UTS UNIVERSITY OF TECHNOLOGY SYDNEY

Cohesive Subgraph Search in Big Graphs

by Conggai Li

Thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

under the supervision of Ying Zhang and Lu Qin

University of Technology Sydney
Faculty of Engineering and Information Technology

November 2020

CERTIFICATE OF AUTHORSHIP/ORIGINALITY

I, Conggai Li declare that this thesis, is submitted in fulfilment of the requirements for the award of Doctoral of Philosophy, in the school of computer science, faculty of engineering and information technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Signature: Production Note: Signature removed prior to publication.

Date: 05/02/2021

ACKNOWLEDGEMENTS

First and foremost, I would like to deliver my sincere gratitude to my supervisor Prof. Ying Zhang for his continuous support and guidance for my PhD study and research, especially for his professionalism, patience, passion, and diligence. He is professional, efficient, patient, and diligent. His guidance broads my knowledge in computer science, improves my scientific research capacity and elevates my love for exploring the significant, undiscovered and challenging fields. Additionally, Ying is a good mentor and friend for me. Thanks to his confidence and encouragement, I am always positive and brave when experiencing challenges and even failures. This thesis could not reach its present form without his illuminating instructions.

Secondly, I would like to express my great gratitude to my co-supervisor A/Prof. Lu Qin, for his guidance and advice, especially for his support and strong confidence in me. He gave me many wonderful ideas and inspirations. His intelligence and guidance significantly extended my knowledge and helped me solve the problems I met during my study. His work efficiency gave me the hope to conduct good research without abandoning too many of other interests, which prevented me to be negative during my PhD study. Lu always has confidence in solving research problems, regardless of their complexities, which encourages me to keep

thinking and challenging myself.

Thirdly, I would like to thank Prof. Fan Zhang for his advice, especially for his guidance and help during my research study. He gave me a lot of great ideas and support during my PhD study. His guidance and help extended my knowledge and helped me solve the problems I met. I learned a lot of writing skills and many other skills for research work from Prof. Fan Zhang. His passion and brilliant ideas always inspired me during my study. Fan always inspires me for my research and gives me many great ideas for my future career. Thanks to Fan's help for this thesis.

Fourthly, I would like to thank Prof. Xuemin Lin and Prof. Wenjie Zhang for supporting the works in this thesis. I thank Prof. Lin for offering an exciting but rigorous research environment. I learned the characteristics of an excellent researcher from Prof. Lin — passion, preciseness, and earnestness. I thank Prof. Wenjie Zhang, her kindness and fantastic research works always inspired me.

Besides, I would also like to thank the following people at UNSW and UTS, Australia: Dr. Dong Wen, Dr. Ouyang Dian, Dr. Wentao Li, Dr. Xin Cao, Dr. Longbin Lai, Dr. Yixiang Fang, A/Prof. Xin Huang, for sharing your brilliant ideas and experiences. Thanks to Dr. Long Yuan, Dr. Xing Feng, Dr. Wei Li, Dr. Kai Wang, Dr. You Peng, Dr. Boge Liu, Dr. Yang Yang, Dr. Xubo Wang, Dr. Haida Zhang, Dr. Fei Bi, Dr. Chen Zhang, Mr. Xuefeng Chen, Ms. Xiaoshuang Chen, Mr. Zhengyi Yang, Mr. Qingyuan Linghu, Mr. Yuren Mao, Mr. Yixing Yang, Mr. Yu Hao, Mr. Chenji Huang, Mr. Kongzhang Hao, Mr. Yizhang He, Dr. Bingqing Lyu, Mr. Michael Ruisi Yu, Dr. Mingjie Li, Dr. Wanqi Liu, Mr. Hanchen Wang, Mr. Yuanhang Yu, Mr. Yilun Huang, Mr. Peilun Yang, Mr. Bohua Yang, Mr. Junhua Zhang, Mr. Yuxuan Qiu, Dr. Adi Lin, Dr. Zhibin Li, Mr. Han Zheng, Mr. Huipeng Xue, Ms. Lu Zhang, Mr. Yongshun Gong, and other colleagues for sharing the happiness and bitterness with me during my PhD study. The time we spent together will never forget.

Finally, I would like to thank my parents for bringing me a wonderful life and other relatives for their understanding, encouragement, and love. Thanks to my husband, Mr. Jianjun Chen, for his company and support in my PhD study.

ABSTRACT

Graphs are widely used to model relationships in various applications, such as social science, biology, information technology, to name a few. Mining cohesive subgraphs is one of the fundamental problems in graph analytics, where the main aim is to find subgraphs with well-connected graph nodes/vertices. A variety of models have been proposed to capture the cohesiveness of subgraphs with different constraints. In this thesis, we study three cohesive subgraph models to investigate various real-life applications better.

Firstly, we would like to detect the critical users whose leave will break the user engagement of the network, i.e., lead many other users to drop out. Accordingly, we propose the collapsed k-truss problem: detect b vertices from a graph G, whose removal will lead to the smallest size k-truss, i.e., identifying some specific users to strengthen the user engagement of the network/graph. From the theoretical side, we deliver the complexity of this problem: NP-hard and inapproximate. From the practical side, we propose an efficient algorithm that can accelerate the computation by vitally reducing the number of candidates. Extensive experiments on real-life networks (graphs) demonstrate the effectiveness and efficiency of our proposed algorithm.

Secondly, we study the minimum k-core search problem. Given a graph G, an integer k and a set of query nodes $Q = \{q\}$, we aim to find the smallest size of k-core subgraph containing all the query node $q \in Q$. As one of the most representative cohesive subgraph models, k-core model has recently received significant attention. It has been shown that this problem is NP-hard with a huge search space, and it is very challenging to find the optimal solution. There are several heuristic algorithms for this problem, but they rely on simple scoring functions, and there is no guarantee as to the size of the resulting subgraph compared with the optimal solution. Our empirical study also indicates that the size of their resulting subgraphs may be large in practice. In this thesis, we develop an effective and efficient progressive algorithm, namely PSA, to provide a good trade-off between the quality of the result and the search time. Novel lower and upper bound techniques for the minimum k-core search are designed. Our extensive experiments on several real-life graphs demonstrate the effectiveness and efficiency of the new techniques.

Finally, we investigate the fortress-like cohesive subgraph, p-cohesion. Morris defines the p-cohesion by a connected subgraph in which every vertex has at least a fraction p of its neighbors in the subgraph, i.e., at most a fraction $(1-p)$ of its neighbors outside. We can find that a p-cohesion ensures not only inner-cohesiveness but also outer-sparseness. The textbook on networks by Easley and Kleinberg shows that p-cohesions are fortress-like cohesive subgraphs that can hamper the cascade's entry, following the contagion model. Despite the elegant definition and promising properties, there is no existing study on p-cohesion regarding problem complexity and efficient computing algorithms to our best knowledge. In this thesis, we fill this gap by conducting a comprehensive theoretical analysis of the problem's complexity and developing efficient computing algorithms. We focus on the minimal p-cohesion because they are elementary units of p-cohesions and the combination of multiple minimal p-cohesions is a larger p-cohesion. We demonstrate that the discovered minimal p-cohesions can be utilized to solve the MinSeed problem: finding the smallest set of initial adopters (seeds) such that all the network users are eventually influenced under

the contagion model. Extensive experiments on several real-life social networks verify this model's effectiveness and the efficiency of our algorithms.

PUBLICATIONS

Fan Zhang, ***Conggai Li****, Ying Zhang, Lu Qin, and Wenjie Zhang, Finding Critical Users in Social Communities: The Collapsed Core and Truss Problems, TKDE*

Conggai Li*, Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin, Efficient Progressive Minimum k-Core Search, PVLDB2020*

Conggai Li*, Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin, Discovering Fortress-like Cohesive Subgraphs, revision submitted.*

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Graphs are widely used to model the relationships of entities in a large spectrum from the real world, such as social science, biology, information technology, and collaboration networks. Given a graph $G = (V, E)$, where V represents the entities that interest from real-world, E represents the relationship between those entities. With the rapid growth of graph applications, a large number of research efforts have been devoted to many fundamental problems in analyzing graph data, in which cohesive subgraph mining has beenwidely studied. Generally speaking, cohesive subgraph mining aims to find one meaningful subgraph from a given graph based on pre-defined cohesive metrics. Given the fact that the graph data is snowballing, algorithms should be designed to handle big graphs efficiently. For example, there were 1.73 billion daily active users for March 2020 on *Facebook* on average [1].

There are several frequently studied cohesive subgraph models, such as k-truss [61], k-core [92], p-cohesion [82], clique [32], k-edge connected component [26], densest subgraph [45], and k-plex [93], to name a few. More details can

[1] https://www.prnewswire.com/news-releases/facebook-reports-first-quarter-2020-results-301049682.html

be founded in Chapter 2. This thesis mainly studied three fundamental models together with two basic components of social networks ((1) tie strength; (2) user engagement). Furthermore, we focus on the improvement of algorithmic efficiency and effectiveness. In this thesis, we mainly focus on the following problems: (1) Collapse k-truss problem: given a graph G, an integer k, and a budget b, detecting a set of b vertices from G, s.t the removal of these b vertices can lead to the smallest size of k-truss; (2) Minimum k-core search problem, given a graph G, an integer k and a set of query vertices $Q = \{q\}$, we aim to find the smallest size of the k-core subgraph containing all the query vertex $q \in Q$; (3) Fortress-like cohesive subgraph discovering, i.e., p-cohesion. A p-cohesion is a connected subgraph where every vertex has at least a fraction p of its neighbors in the subgraph, i.e., at most a fraction $(1 - p)$ of neighbors outside, and a p-cohesion can ensure not only inner-cohesiveness but also outer-sparseness.

1.1 Motivation

With the rapid growth of information and technology, big graph mining is becoming more and more important. Cohesive subgraph mining is one of the fundamental tools in analyzing network (graph) structures, which has attracted significant interests over recent years [25, 124, 130]. Besides, cohesive subgraph mining is vital in many real life applications, such as viral marketing [70], group recommendation [72], and event organization [44].

How to find cohesive subgraphs meeting the pre-defined cohesive metrics becomes an important research area in recent years. There are many aspects to consider, i.e., efficiency, effectiveness, and scalability, because many real-life graphs contain millions or even billions of edges and vertices. Besides, the cohesive subgraph mining needs to analyze the graph structure better.

1.1.1 Collapsed k-Truss Computation

Cohen [35] introduces the cohesive subgraph model k-truss at 2008. The k-truss is defined as a maximal subgraph, in which the support of every edge is $(k-2)$ (i.e., each edge in the k-truss is contained in at least $(k-2)$ triangles). The number of triangles that an edge is involved can capture the strength of the edge [87]. The definition of k-truss ensures that each edge in the k-truss has high embeddedness, i.e., many common friends of two users incident to the edge. Aral and Walker[10] present that, with extensive experiments, the user interactions can be increased with high embeddedness of edges. The definition also deduces that each vertex in the k-truss has at least $(k-1)$ friends inside. Many studies deliver the conclusion that users prefer to engage in communities if a large number of their friends are engaged [77, 34, 18].

Wang *et al.* [105] conduct an in-depth study on 10 representative community detection algorithms. In the study, the k-truss shows high quality on the community metrics and high accuracy on approximating ground-truth communities. The k-truss computation (named M-KMF in the study) is also the most efficient one among all the evaluated algorithms. Ugander *et al.*[102] show that the friends' number in local subgraphs, such as k-truss, can control the social contagion, rather than the total number of friends in the graph. Benefits from the nice properties of k-truss, many studies utilize the k-truss model to find social communities to further analyze the network structures, e.g., [104, 129, 59, 2, 58]. Nevertheless, none of them study the importance of users concerning the k-truss communities.

The leave of users can break down user engagement level and weaken the tie strength inside the communities. Due to the limited budget, we aim to find the most valuable users (i.e., *critical users*) whose leave can significantly decrease the size of the k-truss. The natural problem is that, given an integer k and a

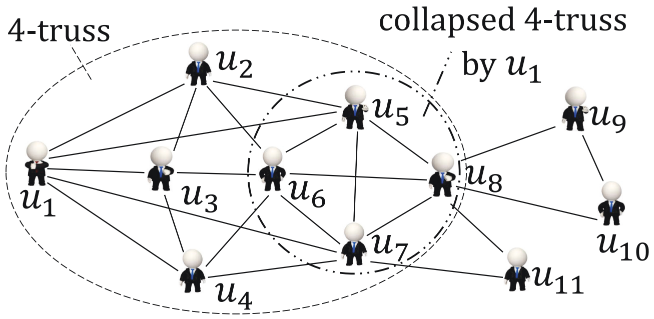


Figure 1.1: Collapsed k-Truss Motivation Example

limited budget b, remove b vertices from a graph so that we can get the smallest k-truss. This problem is named as the collapsed k-truss problem. The resulting critical users play an important role regarding to the engagement perspective of the communities. The problem can be used to reinforce the robustness of k-truss communities against the attack.

Applications. The importance of the collapse k-truss problem in user engagement related applications can be reflected through the following representative example.

Example 1. *Suppose that there is a project team (group), and every member wishes to have enough familiar members in the same team to conduct better collaboration, otherwise he/she may choose to leave.*

Figure 1.1 shows a small workgroup. We model it as a network. 11 *people and their friendship construct this network. Based on the definition of k-truss, an edge will be deleted if it is contained in less than $k-2$ triangles. A vertex will be deleted if it becomes isolated. When $k = 4$, the 4-truss consists of $\cup_{1\leq i\leq 8}u_i$. If users in the 4-truss continue to leave, the 4-truss will be further collapsed. For instance, the leave of u_1 will lead to the quit of u_2, u_3 and u_4. Note that the leave of u_1 deletes all the edges which are incident to u_1. Then the edges (u_2, u_3), (u_2, u_5), (u_3, u_4) and (u_4, u_7) are deleted because each of them is contained in only 1 triangle now. Finally, the collapsed k-truss by deleting u_1 consists of $\cup_{5\leq i\leq 8}u_i$.*

Towards the original k-truss, the leave of u_5 can even lead to the collapse of the whole network. However, the leave of u_8 will not take away any other users. In this sense, we prefer to give u_5 the bonus or other incentives to ensure his/her engagement.

The above example implies an essential need to find the crucial users that may break the group efficiently. As a matter of fact, such a demand arises from many real applications, such as social networks and collaboration networks.

Although there are many other user engagement models, such as Clique [102] and k-core [123], have been widely used to evaluate user engagement widely, we find that the k-truss [57] is being used to more studies especially for finding the cohesive subgraphs when considering the tie strength of the connection between the users. Consequently, we consider these user engagement models as a future extension of this work. There are many unexplored works because of various model analysis networks that come from the demand of corresponding real-life query types, such as diversified top-k problems with clique [117] and k-core [53]. Solving these problems needs a vast demand for the study of user engagement.

Challenges. As far as we know, we are the first to propose and study the collapsed k-truss problem. In Chapter 3, we will deliver the proof of the complexity for the collapsed k-truss problem: (1) NP-hard for any k value, and (2) in-approximate within a factor of $1 - 1/e$ for $k \geq 4$.

A basic exact solution requires to enumerate all possible answer sets with size b. Towards a possible answer set A, we have to conduct the complete k-truss computation with the deletion of A to find the size of collapsed k-truss. Due to the cascade nature in k-truss computation, it is unpromising to estimate the size of collapsed k-truss without the complete k-truss computation. Although the k-truss computation can be done in polynomial time with efficient algorithms, the large number of candidate answer sets makes the exact solutions unaffordable.

We aim to optimize the heuristical solution for the collapsed k-truss problem, where reducing the candidate number is critical and challenging. The computation of k-truss is based on edge deletions with the update of triangles. The triangles' capture in k-truss leads to a larger computation cost while also a more cohesive structure.

The details of this work are presented in Chapter 3.

1.1.2 Minimum k-Core Search

Query processing and mining with cohesive subgraphs are fundamental problems in graph analytics, where the main aim is to find groups of well-connected graph vertices. k-core is an important cohesive subgraph model based on *k-core constraint*: a subgraph is a k-core (subgraph) if every vertex has at least k neighbors in the same subgraph. Problems related to k-core model have been intensively studied in the literature, with many existing research efforts mainly focusing on the *maximum* k-core computation, which aims to find the *largest* induced subgraph satisfying k-core constraint. Indeed, there are many important applications for maximum k-cores, most notably user engagement [18, 77, 18] and influence evaluation [123, 63, 103]. Nevertheless, in some scenarios, especially when one or a set of query vertices are involved, users may prefer a small size group because the group size may closely related to the costs (e.g., verification or advertisement/recommendation costs), and the stableness and homophily of the group. In these scenarios, the maximum k-core may contain many extraneous vertices, and it is more natural to find k-core subgraphs containing all query vertices with the smallest size. From a theoretical perspective, it is also an interesting optimization problem given the degree constraint and optimization goal (i.e., minimize the subgraph size).

In Figure 1.2, the graph G is a k-core with $k = 3$. However, for the given

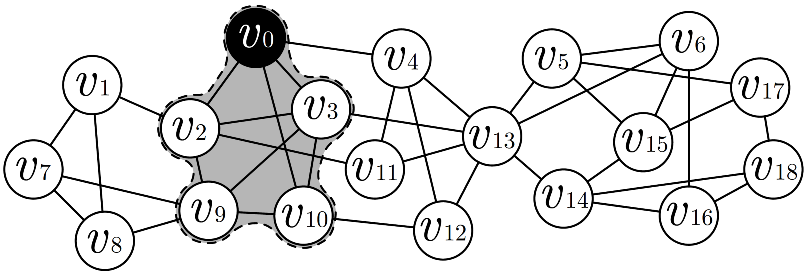


Figure 1.2: A Minimum k-Core Motivation Example, $k = 3$

query vertex v_0, it is more intuitive to return the subgraph $\{v_0, v_2, v_3, v_9, v_{10}\}$ as a k-core subgraph containing v_0 with $k = 3$, instead of using the whole graph G. In this thesis, we study the problem of *minimum* k-core search, which aims to find the smallest k-core subgraph containing the given query vertex. The applications and challenges associated with this problem are discussed below.

Applications. The importance of the minimum k-core search problem can be reflected through some concrete examples in the following representative applications.

Social Networks. It is a common practice to encourage the engagement of group members in the social network by utilizing the positive influence from their friends in the same group (e.g., [111, 43, 18, 77]). In some applications such as group recommendation with one or a set of query vertices (e.g., specific users), a small k-core subgraph may be preferred because a large k-core subgraph may contain many irrelevant vertices for the query vertex. In online social platforms (e.g., Groupon, Pinduoduo and Meetup), the system may recommend an item (e.g., ticket or event) to a user based on the information from a social group containing the user. Usually, the item has been adopted (e.g., bought) by some users in the group, and there are certain interactions among these users. Consider the recommendation cost and the stableness of the group, using the smallest k-core subgraph is more feasible than using the large ones. By doing this, the recommendation is more likely to be adopted, and the group members will be better motivated to take action (e.g., buy items) together due to positive peer influence [74].

Biological networks. It has been reported in [5, 12] that in protein-protein interaction (PPI) networks, the proteins in the same k-core subgraph are likely to have the same functionalities. However, our empirical study reveals that such homophily property only holds for small k-core subgraphs, not for large ones. Thus, it is more reliable to use the minimum k-core subgraph if we aim to find a group of proteins such that, with high probability, they have the same functionality as the query vertex (i.e., homophily property). The cell-assembling theory [55] in neural networks is another example. This theory suggests that information processing in the brain is based on the collective action of groups of neurons, which is essential for acquiring memories (e.g., [55, 20, 109, 88, 98]). As shown in [88, 98], a neuron can be fired/activated if a certain number of neighbor neurons have been activated. This implies that, to activate an area of neurons, we can initially stimulate a small k-core subgraph for cost-effectiveness purpose.

Challenges. As shown in [37, 16, 8], the minimum k-core search problem is NP-hard and cannot be approximated with any constraint factor. We remark that this problem is more challenging than finding a k-clique containing q, although the latter is also NP-hard, in the sense that minimum k-core search needs to explore neighbors with more than one hop. As such, it is cost-prohibitive to find optimal solution for minimum k-core search problem in practice given the huge search space involved.

To circumvent this obstacle, existing studies [37, 16] propose greedy algorithms to incrementally include candidate vertices according to their scoring functions till the resulting subgraph is a k-core subgraph. These algorithms are simple and time efficient, but they do not offer any quality guarantee over the size of the resulting k-core subgraph. Therefore, we propose a progressive algorithm by developing novel techniques to incrementally derive lower and upper bounds for the size of the minimum k-core containing the query vertex. By doing this, we can safely terminate the search once the desired approximate ratio is reached.

The details of this work are presented in Chapter 4.

1.1.3 p-Cohesion Computation

Morris [82] defines the p-cohesion by a connected subgraph in which every vertex has at least a fraction p of its neighbors in the subgraph, i.e., at most a fraction $(1 - p)$ of its neighbors outside. We can find that a p-cohesion ensures not only inner-cohesiveness but also outer-sparseness. The textbook on networks by Easley and Kleinberg [41] shows that p-cohesions are fortress-like cohesive subgraphs which can hamper the entry of the cascade, following the contagion model.

The advantages of the p-cohesion model over other existing cohesive subgraph models are twofold: first, with a large p value, we can find a p-cohesion ensures not only inner-cohesiveness, as the vertices inside a p-cohesion are cohesive; but also outer-sparseness, as the outside neighbors of the p-cohesion have a sparse connection to the p-cohesion; second, in many applications, it is more natural to consider the neighbors' percentage instead of the same number of neighbors (such as the k value in a k-core) within the cohesive subgraph. For instance, in real-life social networks, a large-degree user may need more neighbors than a small-degree user to encourage her/him to adopt a behavior [13].

The p-cohesion is also related to the contagion model, which is introduced in [82] to study the interaction of large populations: given a graph with some initial adopters of a behavior 'A' and a cascading threshold $r \in (0, 1)$, a user

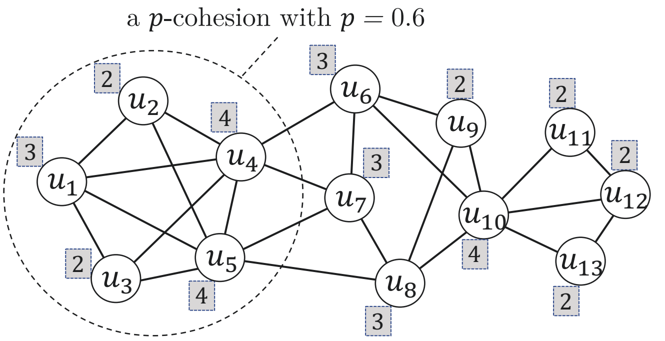


Figure 1.3: A p-Cohesion in A Small Graph, $p = 0.6$

will adopt 'A' if at least a fraction r of his/her neighbors already adopted 'A'. Clearly, with the contagion model, for any p-cohesion S with $p > (1-r)$, none of the users in S will adopt 'A' if S does not contain any initial adopters; in other words, the influence coming from outside of S alone cannot affect any user in S. In this sense, a p-cohesion is a fortress regarding the contagion model.

Example 2. *Figure 1.3 shows a small graph. Suppose* $p = 0.6$*, we use the gray-filled rectangles to label every user* u *with the smallest number of neighbors required for* u *to stay in a* p*-cohesion. A minimal* p*-cohesion* S *is marked in the dashed circle, which is the subgraph induced by vertices* u_1*,* u_2*,* u_3*,* u_4*, and* u_5*. Suppose* $r = 0.5$ *in the contagion model and there is no initial adopter in* S *for product 'A', any user in* S *will not adopt 'A' even if all the other users (not in* S*) adopted 'A'.*

Applications. The fortress-like cohesive subgraphs (i.e., p-cohesions) are critical for information diffusion related applications. They may be information islands in social networks, focalization in viral marketing, etc. We may benefit from exploiting the fortress property of p-cohesions. For instance, in viral marketing, it is usually hard to market a new product to homogeneous users in social groups (e.g., p-cohesions) who use another competing product. In contrast, these users form a business focalization [41]. Offering incentives such as discounts or free product trials to users in p-cohesions may help in the successful marketing of a new product.

In this thesis, we are interested in the minimal p-cohesion problem where we say a p-cohesion S is minimal if there does not exist a proper subgraph S' of S ($S' \subset S$) which is also a p-cohesion. Because: (1) they are elementary units of p-cohesions, and the union of multiple minimal p-cohesions forms a larger p-cohesion; (2) we may avoid enumerating an overwhelming number of p-cohesions; and (3) it is more useful to find small fortresses in influence-related applications. For instance, it is immediate that the whole graph or a connected component is a p-cohesion for any p value, but this is not interesting in these applications.

Minimal p-cohesions also enable us to find good heuristics for the MinSeed problem under the contagion model; that is, given a target graph and a cascading threshold r, find a minimum set of seeds such that the whole graph is eventually influenced. The minimal p-cohesions can hinder the entry of cascades to it, and the vertices inside a p-cohesion are relatively isolated from the outside vertices. Therefore, by giving certain priorities to seed the vertices inside the minimal p-cohesions, we may break the entry barriers of the p-cohesions by the great influence power of the selected seeds.

In this thesis, we study two representative problems with regard to the p-cohesion model: minimum p-cohesion search and diversified p-cohesion enumeration.

> **Minimum p-cohesion search** aims to find the smallest p-cohesion containing the given query vertex, i.e., the p-cohesion with the smallest number of vertices to which a user (query vertex) belongs. We show this problem is NP-hard, and some heuristics are proposed to efficiently identify a p-cohesion with a small size for the given query vertex.

Diversified p-cohesion enumeration aims to find a set of diversified p-cohesions which can cover as many vertices as possible. Here, we consider diversity because, in practice, the user may be overwhelmed by the exponential number of minimal p-cohesions. Thus, in this thesis, we design efficient algorithms to find a set of disjoint minimal p-cohesions.

Challenges. To our best knowledge, there is no existing study on p-cohesion regarding problem complexity and efficient computing algorithms. We are the first to fill this gap by conducting a comprehensive theoretical analysis of the complexity of the p-cohesion discovering problem and developing efficient computing algorithms. We prove minimum p-cohesion search problem does not admit a PTAS for any $p \in (0, 1)$, unless P = NP. For a graph G, it contains an exponential number of minimal p-cohesions, for every fixed $p \in (0, 1)$.

A basic exact solution requires enumerating all minimal p-cohesions to solve the minimum p-cohesion search problem. As shown in our experiment part, the number of minimal p-cohesion containing a certain query for a graph with 70 vertices can be $19,778.1$, which is much larger than the graph size. Furthermore, the number of all minimal p-cohesions for the graph with 70 vertices can be $87,429$, which is impossible to enumerate all for big graphs.

The details of this work are presented in Chapter 5.

1.2 Contributions

In this section, we summarize all the contributions in this thesis. We proposed efficient techniques to deal with the three critical problems. In the following, for each of them, we briefly introduce our contributions.

1.2.1 Collapsed k-Truss Computation

Even though there are polynomial-time algorithms for the k-truss subgraph computation [35, 104], we prove that the problem of finding critical users according to k-truss is NP-hard and in-approximate. A straightforward solution is to enumerate all possible answer sets with a size of b. Towards a possible answer set A, we have to conduct the complete k-truss computation with the deletion of A to find the size of collapsed k-truss. We show that this is not possible because of the large number of combinations of b vertices from the big graph with million or even billion vertices. In this thesis, we show that the performance can be improved by considering the pruning rule. Our technique can reduce the search space dramatically. Our principal contributions for the collapsed k-core computation are as follows.

We propose and investigate collapsed k-truss problem to find critical users according to the well-studied model k-truss. The problem can help reinforce (resp. destroy) the social communities by encouraging the critical users' engagement (resp. the leave).

We give the proof of the complexity for the problem: (1) NP-hard for any k value, and (2) in-approximate within a factor of $1 - 1/e$.

We develop efficient heuristic algorithm CKT to solve the problem. The proposed pruning techniques significantly eliminate the unpromising candidate vertices.

We show the advantages and disadvantages of CKT by experimental comparison. A large number of experiments on real-life networks exhibit the efficiency and effectiveness of our techniques.

Details of this work are presented in Chapter 3.

1.2.2 Minimum k-Core Search

Even though there is a linear time algorithm for the k-core subgraph computation [17], the minimum k-core search problem is NP-hard and cannot be approximated with any constraint factor [8, 16, 37]. Several efficient algorithms were proposed [97, 16, 37]. While, to the best of our knowledge, the subgraphs returned by these solutions do not have any guarantee with regard to the optimal result. In this thesis, we proposed a progressive framework and three lower bounds, one upper bound technique to find the minimum k-core with a certain guarantee. Below are our principal contributions in this thesis for the minimum k-core problem.

We study the problem of the minimum k-core search which aims to find the smallest k-core subgraph containing a given query vertex. An effective and efficient Progressive Search Algorithm, namely *PSA*, is proposed to provide an approximate solution by incrementally computing lower and upper bounds of the optimal solution.

We investigate three approaches to compute the lower bound of the optimal solution after mapping the problem of lower bound computation to the set multi-cover problem. We also design an *onion-layer* based heuristic algorithm to find small k-core subgraphs, and the smallest k-core subgraphs seen so far will serve as the upper bound as well as the approximate solution.

We conduct comprehensive experiments on real-life graphs to evaluate the proposed techniques. The results demonstrate the efficiency and effectiveness of our methods. Particulary, we show that the proposed techniques outperform the state-of-the-art technique *S-Greedy* [16] in two ways: (1) by using our upper bound technique alone, the corresponding greedy algorithm, namely *L-Greedy*, dominates *S-Greedy* under all settings in the experiments because the former can always find smaller k-core subgraphs with less search time; and (2) *PSA* algorithm equipped with both lower and upper bounds techniques can further significantly reduce the resulting subgraph size and provide good trade-off between result quality and the search time.

The details of this work are presented in Chapter 4.

1.2.3 p-Cohesion Computation

Despite the elegant definition and promising properties of the p-cohesion model, there is no study on the problem complexity or the related efficient algorithms to our best knowledge. An exact solution for searching the minimum size p-cohesion is to enumerate all minimal p-cohesions and return the smallest size one. While the tremendous number of minimal p-cohesions make it impossible to compute all of them. In this thesis, we proved that the minimum p-cohesion mining does not admit a PTAS, unless p = NP, and the number of all minimal p-cohesions is exponential. From theory to practice, we developed some techniques to compute one minimal p-cohesion containing a certain query vertex and compute a disjoint set of minimal p-cohesions for graphs. In this thesis, we made the following contributions.

On the theoretical side, we prove that (1) the problem of finding the smallest p-cohesion does not admit a PTAS, unless P = NP; and (2) the number of minimal p-cohesions can be exponential in the graph size.

On the practical side, we propose efficient algorithms to find a p-cohesion with small size containing a query vertex and identify a set of diversified minimal p-cohesions.

Since the MinSeed problem is also NP-hard, we propose a heuristic solution for it considering the property of the minimal p-cohesions, which significantly reduces the number of required seeds for the problem, compared with other feasible solutions.

Comprehensive experiments are conducted to demonstrate the effectiveness of the studied model and the efficiency of the proposed algorithms.

The details of this work are presented in Chapter 5.

1.3 Organization

The thesis is organized as follows:

Chapter 2 provides the related work on cohesive subgraph models, user engagement, tie strength, and the contagion model.

Chapter 3 presents the collapsed k-truss problem, the proof of the complexity, the proposed algorithms, and the experimental results.

Chapter 4 describes the minimum k-core search problem, our lower and upper bounds techniques under the progressive framework, and our experimental results.

Chapter 5 presents the fortress-like cohesive subgraph discovering problems, our theoretical contributions, proposed algorithms, and finally the experimental results.

Chapter 6 concludes our research work and provides future works.

Chapter 2

LITERATURE REVIEW

2.1 Cohesive Subgraph Models

Various cohesive subgraph models are proposed to accommodate different scenarios. Clique [73, 38] is the most cohesive subgraph where every two nodes are adjacent. Because of the over-restriction of the clique model, some clique relaxation models are proposed. For instance, k-core [92, 65], k-truss [35, 125], k-plex [93], k-fami [121], dense subgraph [42, 89], and p-cohesion [82], to name a few. Among all these cohesive subgraph models, k-core and k-truss are the widely studied and can computed in polynomial time.

2.1.1 k-Core

Seidman[92] introduces the cohesive subgraph model k-core to detect a maximal subgraph. In this subgraph, the number of neighbors of each vertex is at least k. The k-core model benefits a lot of vital problems in recent years with broad applications. For instance, social contagion [102, 35], user engagement [18, 77, 120, 122, 124], hierarchical structure analysis [7], network analysis [36, 1], influence studies [63, 103], dense subgraph problems [9, 27], graph visualization [6, 129, 54], event detection [79], anomaly detection [95], internet topology [22], graph clustering [49], structure analysis of software system [126] and protein function prediction [5, 112].

Core decomposition and its related structure have also been widely studied. Batagelj and Zaversnik [17] design a linear in-memory algorithm to derive core numbers of vertices in a graph. Matula and Beck [78] use bin-sorting algorithm to find the core number of vertices. Wen [108] and Cheng [31] present I/O efficient algorithms for core decomposition. Locally estimating core numbers is studied in [84]. Several papers [3, 71, 128] study core maintenance and its efficient algorithms to update core numbers against edge addition or deletion.

Bhawalkar *et al.* [18] propose the anchored k-core problem to prevent unraveling of social networks by retaining some users with additional bonus. This problem is to find b vertices from the graph such that the existence (anchor) of the b vertices can lead to the largest size k-core, where b is a budget. They present an algorithm for bounded tree-width graphs, which are usually inapplicable in real-life social networks. The problem is proved to be harder in [77], which shows its NP-hardness even on the planar graph. Zhang *et al.* [122] come up with an effective and efficient algorithms to solve this problem. The proposed algorithm is mainly based on a specific order of vertices with regard to the core numbers of these vertices. Considering the possibility that a user may choose to leave a community, Zhang *et al.* [123] aim to find the users who can break the user engagement of the k-core community.

Some works aim to discover small-size subgraphs with k-core model. Amini *et al.* [8] study some degree-constrained subgraph problems including the minimum-size k-core search. Cui *et al.* [37] propose efficient algorithms to locally search the k-core. Barbieri *et al.* [16] propose greedy algorithms to search the minimum k-core for one or multiple query nodes. Wood *et al.* [110] study the k-assemblies based on minimal k-core computation.

2.1.2 k-Truss

Since the k-core model does not consider the strength of each tie (i.e., edge) that much, by further considering the tie strength, Cohen [35] proposes the k-truss model where each edge should be involved in at least $(k-2)$ triangles. He also proposes an algorithm for truss decomposition with time complexity of $\mathcal{O}(\sum_{v \in V(G)} deg(v)^2)$ in the paper. Wang *et al.* [104] reduce the time complexity of truss decomposition to $\mathcal{O}(m^{1.5})$. They also propose an I/O efficient algorithm in large graphs that cannot fit in memory.

The definition of k-truss deduces that each vertex has at least $(k-1)$ neighbors in the k-truss subgraph. Compared with definition of k-core, the k-truss model is proposed as an enhanced version of k-core model by further computing and requiring the strength of each tie (edge) [104, 129, 59]. Since k-truss requires each edge inside to be contained in at least $(k-2)$ triangles, which ensures every social tie in the k-truss is strong [87]. Note that, tie strength is a fundamental and important social network characteristic. Due to the fact that each tie in the k-truss is relatively strong, the k-truss can be used to model strong tie communities. Besides, k-truss subgraph can also be used to capture high engagement users from the community [105, 2, 58].

Although many research works are based on the k-truss model, they do not focus on finding critical users. Huang *et al.* [57] propose the online algorithm for the community search problem based on the truss model on large and dynamic graphs. Huang *et al.* [58] also detect the k-truss communities with the largest attribute relevance score from attributed graphs. Akbas *et al.* [2] speed up the search of the truss-based community by a designed truss-equivalence based index. As far as we know, we are the first to utilize k-truss model on discovering essential users from networks.

2.1.3 p-Cohesion

Morris [82] defines the p-cohesion by a connected subgraph, in which every vertex has at least a fraction p of its neighbors in the subgraph, i.e., at most a fraction $(1 - p)$ of its neighbors outside. By definition, we can find that a p-cohesion ensures not only inner-cohesiveness but also outer-sparseness. Easley and Kleinberg [41] show that p-cohesions are fortress-like cohesive subgraphs which can hamper the entry of the cascade, following the contagion model. Despite the elegant definition and promising properties, there is no existing study on p-cohesion regarding problem complexity and efficient computing algorithms to our best knowledge.

Although the above introduced cohesive subgraph models (i.e., clique, k-core, k-truss) hold fantastic properties, none of the above models possess the fortress property to defend outside information cascades. Furthermore, most cohesive subgraph models only consider the cohesiveness inside the subgraphs and ignore the external interactions. The k-defensive alliance [46, 91, 115] has been proposed to find a subgraph S in which a vertex v has at least k more neighbors in S than out of S, which corresponds to the p-cohesion with $p = 0.5$ when $k = 0$. However, there is no such mapping between the two models for an arbitrary p value. We stress that none of the above subgraph models possess the fortress property to defend the enter of information cascades.

2.2 Cohesive Subgraph Search

There is a large body of research on finding cohesive subgraphs within various large graphs. The cohesive subgraph search aims to find subgraphs that contain a set of query nodes, in which the nodes are intensively linked to each other concerning a particular goodness metric. k-core is one of the popular models to capture the structural cohesiveness of a subgraph. Some existing works [97, 37, 16] have been proposed to find the small size k-core subgraphs with different search heuristics. Sozio *et al.*[97] propose a linear-time global algorithm to find a k-core containing a query node with size and distance constraint. Cui *et al.*[37] study the same problem as Sozio [97] but propose a more efficient local search algorithm which works in a local expansion manner that may have a smaller size than that of the global algorithm. The algorithm expands locally to find k-cores based on two scoring functions, often resulting in smaller size subgraphs than the global one. Barbieri *et al.*[16] subsequently present a more accurate and efficient algorithm for the same problem by designing new scoring functions. However, all mentioned works cannot guarantee the size of the returned k-core subgraph, i.e., they just find a smaller k-core, which may be very large as they state in their work. In reality, this usually means the k-core found is smaller than the whole but can still be infeasibly large. Thus, the k-core subgraph identified may be very large in practice.

Size-constrained k-core problem is another popular work. Recently, Ma *et al.*[75] study the size constrained k-core search problem, which is to find a k-core with exact size h and the smallest closeness among all size h subgraphs containing a query vertex on a weighted graph. This model is suitable for applications when the cohesive subgroup size is prefixed.

k-core is not the only goodness metric for cohesive subgraphs search. Huang *et al.* [57] propose an online algorithm and triangle connectivity preserving index to tackle the k-truss query problem. Yuan *et al.* [118] answer a cohesive subgraph search problem with an index-based approach. Their chosen problem is the densest clique percolation subgraph search problem, aiming to find the

k-clique percolation subgraph with the maximum k value that contains a query set. Chang *et al.* [24] propose a compact index structure to answer the k-ECC search problem, which can run in linear time regarding the result size.

2.3 User Engagement

Whenever there is an activity that depends on the involvement of individuals, engagement becomes a primary and important tool to evaluate the robustness of the activity. Behavioral engagement can be observed via individual's actions.

User engagement is an essential tool to evaluate the quality of the user experience that emphasizes the interaction's positive aspects. In particular, the phenomena associated with being captivated by a web application, and numerous studies have been motivated to use it [66]. Other engagements, such as self-reported engagement, cognitive engagement, and online behavior metrics, have also been widely studied. In the self-reported engagement, questionnaires and interviews are used to elicit user engagement attributes [90]. Cognitive engagement occurs when individuals engage in events that are outside their deep emotional range [86]. Online behavior metrics can collect data from millions of users. The users accessing a service daily is a strong indication of a high engagement [66]. Thus those can indicate the engagement metrics in their areas, and measuring the effect on engagement metrics can explain why users engage with service.

Evaluating the user engagement related activities is vital for social networks, which can improve the stickiness among users and can avoid the collapse procedure of the network. Collapse means that the leave of users from a group will lead to the leave of other users, which is very common in real-life. Malliaros and Vazirgiannis [77] verify that the degeneracy property of k-core can provide a way to quantify the engagement incrementally in real social networks. Seki *et al.* [94] present the collapse procedure of the Friendster network and explain its mechanism, and also presents that the collapse usually starts from the center of the core structure. Bhawalkar and Kleinberg *et al.* [21] use the game-theory to show the network unraveling process stops when the remaining engaged individuals correspond to the k-core of the network.

User engagement has attracted vital interests in the past few years [111, 106]. k-core is a simple and popular model based on degree constraint, which has been widely used to measure the network engagement [77, 33, 48, 123]. Suppose that all users in a community are initially engaged. Each user inside has two options: (1) to remain engaged: if at least k of his/her friends are engaged (i.e., degree constraint), (2) and drop out: if a user has less than k friends are engaged. In the second case, the leave of this user may be influential, which will result in a cascade of departure (i.e., collapse) in the network. Once the collapse procedure stops, the rest engaged users correspond to the well-known concept k-core, the maximal induced subgraph in which every vertex has at least k neighbors [123]. Bhawalkar [18] shows that a user seems to keep strong activate engagement if there are a large number of this user's friends engaged.

2.4 Tie Strength

Besides user engagement, the tie strength, which is introduced by Granovetter [52], among users is another fundamental character in social network [40, 50, 87, 96]. The model of k-core fits well with the communities which does not require tie strength. Such as in a study group, students may discuss with other students in the group, even though their relationships are not quite strong. Nevertheless, some community types need to consider the strength of ties, e.g., a company's programming team.

Tie strength is a fundamental character for the social network. A large number of excellent research works has been done in the sociology related areas. In recent times, various researchers working in social network related areas show that the tie strength is an important component for the graph and is worth for further exploring [14, 50, 129]. Onnela *et al.* [85] give a counterintuitive consequence that social networks are robust to removing the strong ties but collapse on the phase when some weak ties have been removed. They observe coupling between tie strengths and the local structure and show that this coupling significantly slows down the information propagation process. Bakshy *et al.* [14] examine the relative roles between strong and weak ties in information diffusion. They show that although strong ties may be more influential towards individuals, the effect of strong ties is not large enough to substitute the abundant information from weak ties in social networks.

Gilbert and Karahalios [50] present a predictive model that maps social media data to tie strength for distinguishing strong and weak ties. They show that the model can enhance social media design elements, including friend recommendation, message routing, privacy controls and information prioritization. Sintos and Tsaparas [96] use the principle of strong triadic closure to characterize the tie strength in social networks. They study the problem of labeling the ties by strong or weak so as to enforce the strong triadic closure property. Rotabi *et al.* [87] present that structural information of graphs is often used to detect strong tie, especially on triangles. They experimentally demonstrate that using only structural network features is sufficient for strong tie detection with high precision. With the minimum triangle number of k for each edge, the k-truss [35] can be regarded as a strong tie community where each edge in the community is a strong tie.

2.5 The Contagion Model

The contagion model. Morris [82] introduces the contagion model and p-cohesion to characterize social choices in local interaction systems. The paper also studies the diffusion of a behavior from a finite set of initial adopters to all network users. Centola [23] studies the strength of weak ties with the contagion model. Ugander *et al.* [102] show that the contagion probability of a user is strongly influenced by his local neighbors, e.g., neighbors in cohesive subgraphs. Zarezade *et al.* [119] study correlated cascades based on the fact that the adoption of a behavior by a user is influenced by the aggregation of the behaviors of his/her neighbors. Young [116] shows that utilizing local clusters greatly enhances the spread of innovations. Easley and Kleinberg [41] make the same observation as the above in various applications. They further emphasize the contagion model and the p-cohesion that it is difficult for new innovations to enter tightly-knit social groups (i.e., p-cohesions), because people tend to interact with their friends or acquaintances.

Other cascade models. In addition to the contagion model, there are some other information cascade models, such as independent cascade model (IC) and linear threshold (LT) model [62], where the influence maximization problem and seed minimization problem have been extensively studied, e.g., [127, 11, 100, 80, 30]. It uses polynomial time to compute the influence spread of given seeds in the contagion model, while this influence computation is NP-hard for both IC and LT models [76]. For a set of seeds, the spread area of their influence is certain in contagion model while the influence spread is uncertain and complex in IC and LT models due to the possible world assumption.

Chapter 3

FINDING CRITICAL USERS IN SOCIAL COMMUNITY

3.1 Overview

User engagement and tie strength are fundamental components in social communities. Many recent studies show that the k-truss model can capture users with high engagement and strong interactions. It is effective and efficient to utilize k-truss on discovering social communities and analyzing the network structure. We observe that the leave of some *critical users* may significantly break the k-truss communities. This motivates us to propose the *collapsed k-truss problem* to find these users. Giving incentives to these users can reinforce the k-truss communities. We propose the collapsed k-truss problem to identify the critical users: given a graph G, an integer k and a budget b, we aim to detect a set of b vertices from G. Such that the removal of the b vertices may result in a small k-truss. In this chapter, we prove the problem is NP-hard and in-approximate within $1 - 1/e$. To solve this problem, an efficient algorithm is proposed. The work is published in [120] and the rest of this chapter is organized as follows.

Section 3.2 gives preliminary definitions and formally defines the problem. Section 3.3 gives the proof for the complexity of this problem. Section 3.4 describes our algorithms for the proposed collapsed k-truss problem. Section 3.5 introduces the evaluation of our proposed algorithms and reports the experimental results. Section 3.6 summarizes the chapter.

3.2 Preliminary

Let $G = (V, E)$ be an unweighted and undirected graph, where V and E denote the set of vertices and edges in G, respectively. Let n (resp. m) denote the number of vertices (resp. edges) in G. Given a subgraph S of G, we denote the adjacent vertex set (i.e. neighbor set) of u in S by $N(u, S)$. We use $G \setminus S$ to represent the subgraph which removes S from G, i.e., G-S. Let $deg(u, S)$ denote the degree of u in S, which is equal to the number of u's neighbors in S, i.e., $deg(u, S) = |N(u, S)|$. In a graph, the cycle with 3 edges is a triangle. When we say a e-containing triangle in the following, we mean a triangle that contains the edge e, i.e., e is a part of the triangle. The *support* of an e in subgraph S is the number of e-containing triangles in S, we denote it as $sup(e, S)$, If a vertex u is one of the endpoints of an edge e, we say a vertex u is incident to an edge e, or e is incident to u. The notations are summarized in Table 3.1. In this chapter, if a vertex is deleted, its incident edges are also deleted accordingly.

Definition 1. ***k-truss**. Given a graph G, a subgraph S is the k-truss of G, denoted by $T_k(G)$, if (i) S satisfies support constraint, i.e., $sup(e, S) \geq k - 2$ for every edge $e \in S$; (ii) S is maximal, i.e., any subgraph $S' \supset S$ is not a k-truss; and (iii) S is non-trivial, i.e., no isolated vertex in S.*[1]

[1] In real-life scenarios, the k value is usually determined by users based on their real requirement for cohesiveness, or can be learned according to ground-truth communities in the network.

Notation	**Definition**
G	an unweighted and undirected graph
k	a non-negative integer
u, v; e, (u, v)	a vertex in G; an edge in G
n; m	the number of vertices (resp. edges) in G
$N(u, G)$	the set of adjacent vertices of u in G
$deg(u, G)$	the number of adjacent vertices of u in G
$sup(e, G)$	the number of e-containing triangles in G
$E(u, G)$	the edge set where each edge is incident to u and each edge is in G
$E(S, G)$	the union set of $E(u, G)$ for each $u \in S$
A	a set of collapsers
$T_k(G)$	the k-truss of G
$T_k(G_A)$	the k-truss of $G \setminus \{A \cup E(A, G)\}$
$\mathcal{F}(A, G)$	the followers in G of the collapser set A
$V_\triangle(e, G)$	the set of vertices where each vertex is from a e-containing triangle in G, and each vertex is not incident to e
$V_\triangle(S, G)$	the union set of $V_\triangle(e, G)$ for every $e \in S$

Table 3.1: Summary of Notations

This definition also deduces that each k-truss vertex has at least $(k - 1)$ neighbors in the k-truss, because a vertex involves in at least $(k-2)$ triangles in the k-truss. As shown in Algorithm 1, to find the k-truss, we recursively delete vertices with less than $(k - 1)$ degree. Then we recursively remove every edge whose support is less than $(k - 2)$ in current graph. We get the k-truss after removing isolated vertices.

We assume that, the collapse of a vertex v in G will lead to the removal of vertext v and its incident edges from k-truss even if v is not isolated. We use **collapsers** to represent the collapsed vertices.

Definition 2. ***collapsed k-truss***. *Given a graph G and a set $A \subseteq G$ of vertices, the collapsed k-truss, denoted by $T_k(G_A)$, is the k-truss of the subgraph $G \setminus \{A \cup E(A, G)\}$.*

Besides the deletion of the collapsers in A, there may be some other vertices

Algorithm 1 ComputeTruss(G, k)

Input: G : a social network, k : support constraint **Output**: $T_k(G)$: the k-truss of G

```
while exists $u \in G$ with $deg(u, G) < k - 1$ do
    $G := G \setminus \{u \cup E(u, G)\}$
while exists an edge $e \in G$ with $sup(e, G) < k - 2$ do
    $G := G \setminus \{e\}$
Delete isolated vertices in $G$
return $G$
```

in $T_k(G)$ which follow the collapsers to leave the k-truss due to the cascade of edge deletions. We name these vertices as **followers** of the collapsers in A, and denote it by $\mathcal{F}(A, G)$. Formally, $\mathcal{F}(A, G) =$ the vertices in $T_k(G) \setminus \{T_k(G_A) \cup A\}$. The number of the followers indicates the importance of the collapsed vertices.

Problem Statement. Given a graph G, a support constraint k and a budget b, the **collapsed k-truss problem** is to find a set A of b vertices in G such that the number of followers, $\mathcal{F}(A, G)$, is maximized; that is, the size of the resulting collapsed k-truss, $T_k(G_A)$, is minimized.

Example 3. *As shown in Figure 1.1, suppose we set $k = 4$ and $b = 1$, the result of the collapsed k-truss problem can be $A = \{u_5\}$ with $T_k(G_A) = \emptyset$ and $\mathcal{F}(A, G) = \{\cup_{1 \leq i \leq 8} u_i\} \setminus u_5$.*

3.3 Complexity

In this section, we give the complexity of the collapsed k-truss problem.

Theorem 1. *The collapsed k-truss problem is NP-hard for any k.*

Proof. We reduce the collapsed k-truss problem from the Minimum Vertex Cover (MVC) problem [29]. Given a graph G, the MVC problem is to find a minimum vertex set such that each edge in G is incident to at least one vertex of the set.

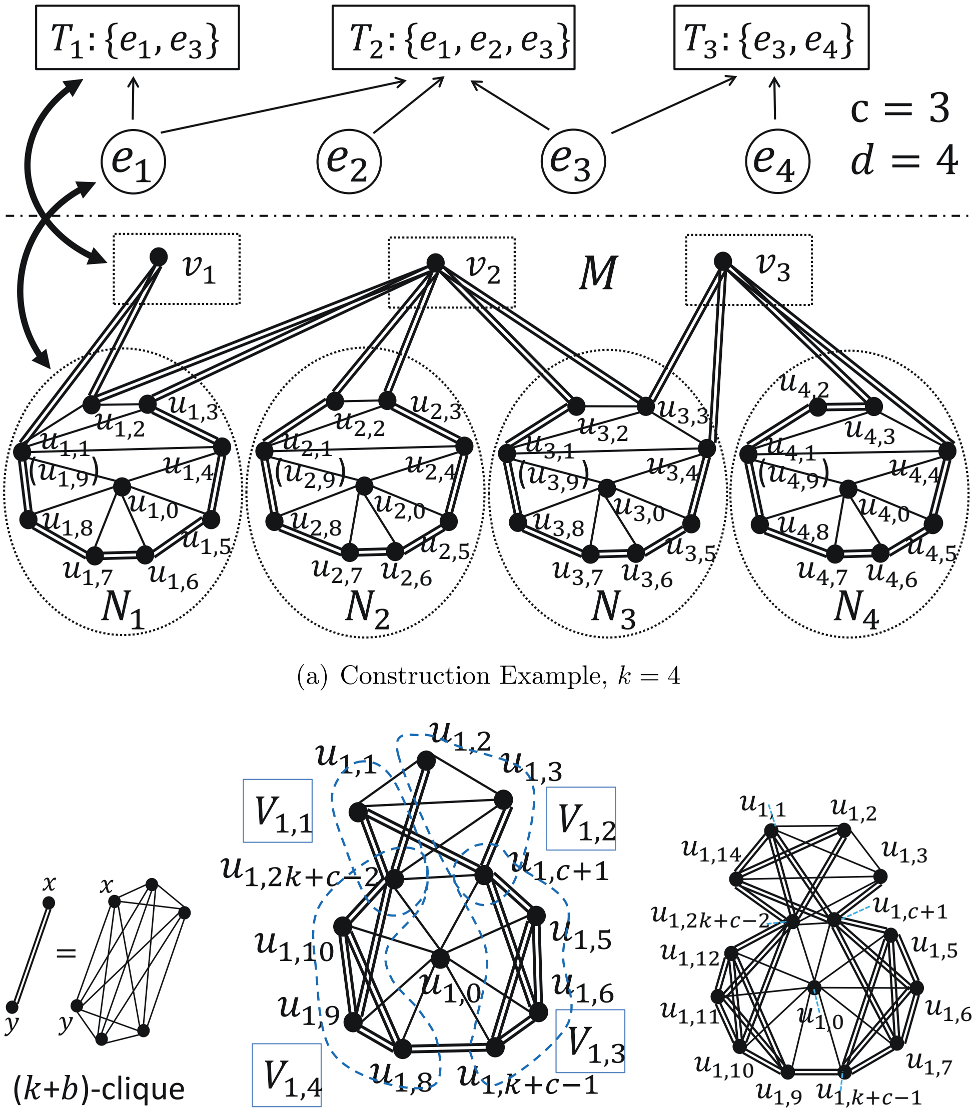
When $k \geq 2$, we construct a graph G' by (1) adding a vertex set U of $(k-2)$ vertices for every edge (u, v) in G; and (2) adding edges to make every $U \cup \{u, v\}$ a clique where every two vertices are adjacent. Since every edge in the induced graph of a k-clique (a clique of k vertices) is contained in $(k-2)$ triangles, a k-clique forms a k-truss.

Then we prove the MVC problem on G is equivalent to finding a minimum vertex set W in G' such that the k-truss of $G' \setminus (W \cup E(W))$ is empty. Let C denote a k-clique in G' which corresponds to an edge in G. To make the k-truss empty, we have to delete at least one vertex in every C. Then we have that the minimum set W comes from G, because deleting a vertex in $G' \setminus G$ cannot destroy more C than deleting a vertex in $C \cap G$. Then every edge in G will be covered by an incident vertex in W. So the above two problems are equivalent. To find the minimum vertex set W, we can try solving the collapsed k-truss problem by at most $(n-1)$ times $(1 \leq b < n)$. So we can say that the collapsed k-truss problem is NP-hard when $k \geq 2$. When $k \leq 1$, the problem is exactly the collapsed 2-truss problem according to the definitions. Then the collapsed k-truss problem is NP-hard for any k. □

Theorem 2. *It is NP-hard to approximate the collapsed k-truss problem within $1 - 1/e$ when $k \geq 4$.*[2]

Proof. For the collapsed k-truss problem, we show a reduction from the Maximum Coverage (MC) problem [56] which is proved to be in-approximate. The MC problem is to find at most b sets to cover the largest number of elements, where b is a given budget [123]. It is NP-hard to approximate the MC problem within $1 - 1/e$ [56]. Let $\gamma > 1 - 1/e$, we prove that if there is a solution with

[2] When $k = 3$, the collapsed k-truss problem can be reduced from the Triangle Vertex Deletion problem [51] whose inapproximability is still an open problem. When $k \leq 2$, the k-truss becomes the connected components except isolated vertices.



(a) Construction Example, $k = 4$

(b) $k = 4$, $b = 2$ (c) N_1, $k = 5$ (d) N_1, $k = 6$

Figure 3.1: Examples for Proving Inapproximability for Collapse k-Truss

γ-approximation on optimal follower number for our problem, there will be a γ-approximate solution on element number for the MC problem.

Considering an arbitrary instance of the MC problem with c sets $T_1, .., T_c$ and d elements $\{e_1, .., e_d\} = \cup_{1 \le i \le c} T_i$. Then we can construct a corresponding instance of the collapsed k-truss problem on a graph G. Figure 3.1(a) shows a construction example from 3 sets and 4 elements with $k = 4$. The set of vertices of G is constructed by two parts: M and N. There are c vertices in M, i.e., $M = \cup_{1\leq i\leq c} v_i$. N is consist of d sets of vertices, i.e., $N = \cup_{1\leq j\leq d} N_j$. For every $j \in [1, d]$, N_j contains $3k+c-6$ vertices, i.e., $N_j = \cup_{0\leq p\leq 3k+c-7} u_{j,p}$. To show the construction clearly, as the example in Figure 3.1(c), we divide N_j into 5 sets: $V_{j,1}$ ($k - 3$ vertices), $V_{j,2}$ (c vertices), $V_{j,3}$ ($k - 1$ vertices), $V_{j,4}$ ($k - 1$ vertices) and $\{u_{j,0}\}$. Specifically, we have $V_{j,1} = u_{j,1} \cup \{u_{j,p} \mid 2k+c-2 \leq p \leq 3k+c-7\}$, $V_{j,2} = \{u_{j,p} \mid 2 \leq p \leq c+1\}$, $V_{j,3} = \{u_{j,p} \mid c+1 \leq p \leq k+c-1\}$ and $V_{j,4} = \{u_{j,p} \mid k+c \leq p \leq 2k+c-2\}$, respectively, (Note that $u_{j,1} = u_{j,2k+c-2}$ when $k = 4$).

The edge construction is as follows: (1) we define the *super edge* as a $(k+b)$-clique, denoted by C, where only two vertices in C have neighbors in $G \setminus C$, as the example in Figure 3.1(a)(b); (2) for every set T_i ($i \in [1, c]$) and every element e_j ($j \in [1, d]$), if $e_j \in T_i$, we add two super edges (v_i,$u_{j,i}$), (v_i,$u_{j,i+1}$) and an edge (non-super) ($u_{j,i}$,$u_{j,i+1}$); if $e_j \notin T_i$, we add a super edge ($u_{j,i}$,$u_{j,i+1}$). (3) there is a super edge between every vertex pair in $V_{j,1}$; (4) for every $p \in [3, c]$, there is an edge between $u_{j,p}$ and each vertex in $V_{j,1}$; (5) we add a super edge between $u_{j,2}$ and each vertex in $V_{j,1} \setminus \{u_{j,1}\}$; (6) we add an edge $(u_{j,c+1}, u_{j,2k+c-2})$, and add a super edge between $u_{j,c+1}$ and each vertex in $V_{j,1} \setminus \{u_{j,2k+c-2}\}$; (7) we add a super edge between every vertex pair in $V_{j,3}$ except the pair ($u_{j,c+1}$, $u_{j,k+c-1}$); (8) we add a super edge between every vertex pair in $V_{j,4}$ except the pair ($u_{j,k+c}$, $u_{j,2k+c-2}$); and (9) we add an edge between $u_{j,0}$ and each vertex in $V_{j,3} \cup V_{j,4}$, and add a super edge between $u_{j,k+c-1}$ and $u_{j,k+c}$. The construction of G is completed. Figure 3.1 (c)(d) show the construction of N_1 ($c = 3$) when $k = 5$ and $k = 6$, respectively.

Here we show the support of each non-super edge in G is exactly $(k-2)$: (1) for a non-super edge $(u_{j,1}, u_{j,2})$, it can only form a triangle with each vertex in

$V_{j,1} \backslash \{u_{j,1}\}$ ($k-4$ vertices), the vertex $u_{j,3}$ and one vertex in M; (2) for a non-super edge $(u_{j,p}, u_{j,p+1})$ $(2 \leq p \leq c)$, it can only form a triangle with each vertex in $V_{j,1}$ ($k-3$ vertices) and one vertex in M; (3) for a non-super edge $(u_{j,c+1}, u_{j,2k+c-2})$, it can only form a triangle with each vertex in $V_{j,1} \setminus \{u_{j,2k+c-2}\}$ ($k-4$ vertices), the vertex $u_{j,c}$ and the vertex $u_{j,0}$; (4) for a non-super edge $(u_{j,p}, u_{j,q})$ between $V_{j,1}$ and $V_{j,2}$, it can form a triangle with each vertex in $V_{j,1} \backslash \{u_{j,p}\}$ ($k-4$ vertices), the vertex $u_{j,q-1}$ and the vertex $u_{j,q+1}$; (5) for a non-super edge $(u_{j,0}, u_{j,c+1})$, it can only form a triangle with each vertex in $V_{j,3} \backslash \{u_{j,c+1}, u_{j,k+c-1}\}$ ($k-3$ vertices) and the vertex $u_{j,2k+c-2}$. Note that it is similar for edges $(u_{j,0}, u_{j,k+c-1})$, $(u_{j,0}, u_{j,k+c})$, and $(u_{j,0}, u_{j,2k+c-2})$; and (6) for a non-super edge $(u_{j,0}, u_{j,p})(c+2 \leq p \leq k+c-2)$, it can only form a triangle with each vertex in $V_{j,3} \setminus \{u_{j,p}\}$ ($k-2$ vertices). Note than it is similar for a non-super edge in $(u_{j,0}, u_{j,p})(k+c+1 \leq p \leq 2k+c-3)$. Now the support of each non-super edge has been verified to be $k-2$. Obviously G is a k-truss.

The key idea is we ensure that: (1) only the vertices $u_{j,0}$ can become a follower of a vertex, since we need to delete at least $b+1$ vertices in a $(k+b)$-clique to produce followers; (2) only vertices in M need to be considered as collapsed vertices, since any vertex in N cannot have more followers than a vertex in M; (3) reduce the support for any non-super edge in N_j $(1 \leq j \leq d)$ will lead to the deletion of $u_{j,0}$ due to the support constraint; and (4) every N_j $(1 \leq j \leq d)$ can only produce one follower $u_{j,0}$. Then, every solution of the collapsed k-truss problem in G corresponds to a solution of the MC problem, where the follower number for our problem equals the element number for the MC Problem. So it is NP-hard to approximate collapsed k-truss problem within $1 - 1/e$ when $k \geq 4$. □

Algorithm 2 GreedyCKT(G, k, b)

```
Input: $G$ : a social network, $k$ : support constraint,
       $b$ : the budget for collapser number
Output: $A$ : the set of collapsers

1: $A := \emptyset$; $i := 0$
2: while $i < b$ do
3:     for each $u \in T_k(G_A)$ do
4:         Compute $\mathcal{F}(A \cup u, G)$
5:     $u^* \leftarrow$ the best collapser in this iteration
6:     $A := A \cup u^*$; $i := i + 1$; update $T_k(G_A)$
7: return $A$
```

3.4 Solution

Due to the NP-hardness and inapproximability of the problem, we adopt a greedy heuristic which iteratively finds the best collapser, i.e., the vertex with the largest number of followers. We only need to consider the vertices in $T_k(G_A)$ as candidate collapsers, because all other vertices will be deleted during k-truss computation. A greedy algorithm is shown in Algorithm 2. The time complexity of this algorithm is $\mathcal{O}(bnm^{3/2})$, where n denotes the number of candidate collapsers in each iteration (Line 3) and m denotes the number of edges in follower computation (Line 4), i.e., k-truss computation.

The follower computation at Line 4 is efficient by conducting k-truss computation. However, the number of candidate vertices in $T_k(G_A)$ at Line 3 is still too large to afford, which motivates us to reduce the candidate vertices in the heuristic algorithm by developing effective pruning rules.

3.4.1 Reducing Candidates

In this section, we introduce the pruning rules on the first iteration in the greedy algorithm (i.e., $A = \emptyset$ and $i = 0$). The pruning rules can be immediately applied

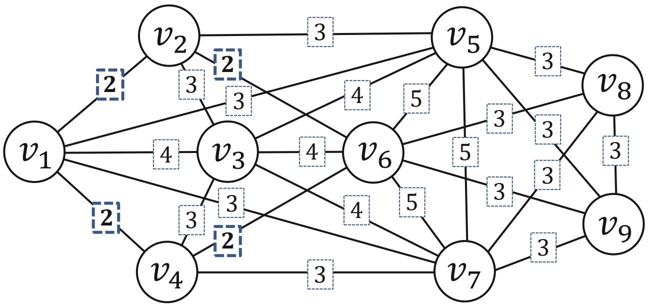


Figure 3.2: Candidate Reduction, $k = 4$

to the following iterations by using $T_k(G_A)$ to replace $T_k(G)$. The following theorem finds the candidate set of collapsers.

Theorem 3. *Given a graph G, let D denote the support $k-2$ edge set in $T_k(G)$, i.e., $D = \{e \mid sup(e, T_k(G)) = k-2\}$, if a collapsed vertex x has at least one follower, x is from $V_\triangle(D, T_k(G))$; that is $|\mathcal{F}(x, G)| > 0$ implies $x \in V_\triangle(D, T_k(G))$.*

Proof. We prove that a vertex $x \in G \setminus V_\triangle(D, T_k(G))$ cannot have any followers. (1) If $x \in G \setminus T_k(G)$, x will be deleted in k-truss computation and hence $|\mathcal{F}(x)| = 0$. (2) If $x \in T_k(G) \setminus V_\triangle(D, T_k(G))$, x survived in k-truss computation. Let $E_\triangle$ denote the edge set where each edge is from a triangle containing x in $T_k(G)$, and let E_x denote the edge set where each edge is incident to x in $T_k(G)$. We have that each edge in $E_\triangle \setminus E_x$ has a support of at least $k-1$, otherwise x will be in $V_\triangle(D, T_k(G))$. If we delete x in $T_k(G)$, the support of every edge in $E_\triangle \setminus E_x$ can only decrease by one, i.e., every edge in $E_\triangle \setminus E_x$ still has a support of at least $k-2$. Consequently, all the edges in $E_\triangle \setminus E_x$ will exist in $T_k(G_x)$, i.e., no cascade of deletion will incur for removing x. So only edges in E_x will be deleted by removing x, and x does not have any followers. Consequently, if x has at least one follower, $x \in V_\triangle(D, T_k(G))$. □

Example 4. *In Figure 3.2, when $k = 4$, the graph is already a k-truss. On* *each edge, we label its support, i.e., the number of triangles containing the edge. According to Theorem 3, the set* $D = \{(v_1, v_2), (v_2, v_6), (v_1, v_4), (v_4, v_6)\}$*. Thus the set* $V_\triangle(D, T_k(G)) = \{v_3, v_5, v_7\}$ *and every other vertex cannot have any followers. The best collapser is* v_3 *since* $\mathcal{F}(v_3) = \{v_1, v_2, v_4\}$*,* $\mathcal{F}(v_5) = \{v_2\}$ *and* $\mathcal{F}(v_7) = \{v_4\}$*.*

The following theorem further reduces the candidates.

Theorem 4. *Given a graph* G*, if a vertex* u *is a follower of* x*, i.e.,* $u \in \mathcal{F}(x)$*, we have* $\mathcal{F}(u) \subset \mathcal{F}(x)$*.*

Proof. Since $u \in \mathcal{F}(x)$, u will be deleted if x is collapsed. For every vertex v in $\mathcal{F}(u)$, v will be deleted if x is collapsed, because u will be deleted and collapsing x cannot increase the supports for edges. So $\mathcal{F}(u) \subseteq \mathcal{F}(x)$. Since $u \in \mathcal{F}(x) \setminus \mathcal{F}(u)$, we have $\mathcal{F}(u) \subset \mathcal{F}(x)$. □

Based on Theorem 4, every vertex which is a follower of another vertex should be excluded from candidate collapser set in the computation of a best collapser. Thus we can reduce even more vertices in the computation by checking promising collapsers first, which may have large number of followers. We say a vertex u *corresponds* to an edge e if $v \in V_\triangle(e, G)$. A vertex corresponds to more edges in the set D is more promising, because all these edges will follow the vertex to be deleted. This edge numbers (en) in D can be accumulatively produced by visiting the triangles of each edge in D. To further reduce the candidate collapsers, we try collapsing vertices in decreasing order of their en.

3.4.2 CKT Algorithm

Algorithm 3 shows the details of CKT algorithm which finds the best collapser for a given graph G (i.e., $b = 1$). Specifically, we firstly compute the k-truss

Algorithm 3 CKT(G, k)

```
Input:  $G$ : a social network, $k$ : support constraint,  Output $x$ : the best collapser

1: $T := \textbf{ComputeTruss}(G,\ k)$
2: $D := \{e \mid sup(u, T) = k - 2\}$
3: $V := V_{\triangle}(D, T)$
4: for each $u \in V$ (Theorem 3) do
5:     Compute $\mathcal{F}(u,\ G)$
6:     $V := V \setminus \mathcal{F}(u, G)$ (Theorem 4)
7: return  the best collapser
```

of graph G (Line 1) and find the set D of edges with support $k-2$ in $T_k(G)$ (Line 2). According to Theorem 3, we find the candidate set $V_{\triangle}(D, T_k(G))$ which correspond to at least one edge in D (Line 3-4). To compute $\mathcal{F}(u, G)$, we continue the k-truss computation in Line 1 with vertex u deleted (Line 5). The best collapser is produced by trying every candidate.

For a general case with $b > 1$, our CKT algorithm can be simply embedded in the greedy algorithm (replacing Line 3 and 4) to find the best collapser in each iteration. In order to avoid the re-computation of D (Line 2) and $V_{\triangle}(D, T_k(G))$ (Line 4) in the following iterations, we incrementally update two sets at the end of each iteration. Specifically, let D_1 denote the edges whose supports are decreased to $k-2$ during the computation and D_2 denote the edges which are discarded during the computation, we have $D = D \cup (D_1 \setminus D_2)$. Towards the set $V_{\triangle}(D, T_k(G))$, we include new vertices correspond to the set D_1 and delete vertices which only correspond to the edges in D_2. We also delete vertices in $V_{\triangle}(D, T_k(G))$ synchronously with the computation of collapsed k-truss in each iteration.

Additionally, if we find a vertex $u \in F(x)$ in one iteration of Algorithm 1, we do not need to consider u as a candidate in all following iterations because x is always a better candidate collapser than u, and u will be excluded from

k-truss whenever x is removed (Theorem 4). Furthermore, in one iteration, if there is a connected k-truss subgraph which does not contain the produced best collapser, the result on this subgraph can be reused. The reason is that the collapsed k-truss of each vertex in the subgraph will keep same. We can share the computation by only recording the largest number of followers for a vertex in this subgraph.

3.4.3 Collapsed k-Core Problem

The collapsed problem first proposed by Zhang [123] with the k-core model. For the fact that, the k-core model do not consider the tie strength of the graph, we use the potential model k-truss to find the crucial users, which is more complex as we showed in the complexity part. In this session, we just give the basic algorithm that proposed by the authors.

Algorithm 4 CKC(G, k)

Input: G : a social network, k : degree constraint, **Output** x : the best collapser

```
1: $C_k(G) :=$ **ComputeCore**$(G,\ k)$
2: $P := \{u \mid deg(u, C_k(G)) = k\}$
3: $T := P \cup \{u \in C_k(G) \& N(u, G) \cap P \neq \emptyset\}$
4: for each $u \in T$ do
5:     Compute $\mathcal{F}(u,\ G)$
6:     $T := T \setminus \mathcal{F}(u, G)$
7: return the best collapser
```

CKC Algorithm. Algorithm 4 illustrates the details of CKC algorithm proposed by the users, which finds the best collapser for a given graph G (i.e., b =1). Particularly, they first compute the k-core of graph G (Line 1) and find the set P of vertices with degree k in $C_k(G)$, the k-core of G (Line 2). Then they find the set T of vertices in P, and vertices which are inside C_k and are neighbors of at least one vertex in P (Line 3). To compute $F(u, G)$, they can continue the k-core computation in Line 1 with vertex u deleted (Line 5). They we can have the best collapser when the algorithm terminates.

3.5 Performance Studies

This section evaluates the proposed model and algorithms through comprehensive experiments.

3.5.1 Experimental Setting

Algorithms. As far as we know, there is no existing work investigating the collapsed k-truss problem and its corresponding algorithms. In the experiments, we implement and evaluate 7 algorithms including the baseline algorithm and our proposed algorithm.

> `BaselineT`. The baseline algorithm (Algorithm 2). In each iteration, it finds a best collapser by computing the collapsed k-truss for every candidate collapser.
>
> `CKT`. The advanced algorithm (Algorithm 3), where Theorem 3 and Theorem 4 are applied.
>
> `CKC`. The advanced algorithm equipped with Algorithm 4.

The other algorithms will be introduced when appear in the experiments for the first time.

Datasets. In our experiments, we deploy 9 real-life social networks and we assume that all vertices in each network are initially engaged. The original data of `Yelp` is from [114], `DBLP` is from [68] and the others are from [67]. In `DBLP`, each vertex represents an author and each edge between two authors represents

Dataset	Vertices	Edges	d_{avg}	k_{max}
Facebook	4,039	88,234	43.7	97
Brightkite	58,228	194,090	6.7	42
Gowalla	196,591	456,830	4.7	23
Yelp	552,339	1,781,908	6.5	73
YouTube	1,134,890	2,987,624	5.3	19
DBLP	1,566,919	6,461,300	8.3	119
Pokec	1,632,803	8,320,605	10.2	20
LiveJournal	3,997,962	34,681,189	17.4	352
Orkut	3,997,962	34,681,189	17.4	352

Table 3.2: Statistics of Datasets

there is at least one co-authored paper of the two authors. The other datasets have existing vertices representing users and edges representing relationships. Table 5.3 shows the statistics of all the datasets used in the experiments, in increasing order of their edge numbers.

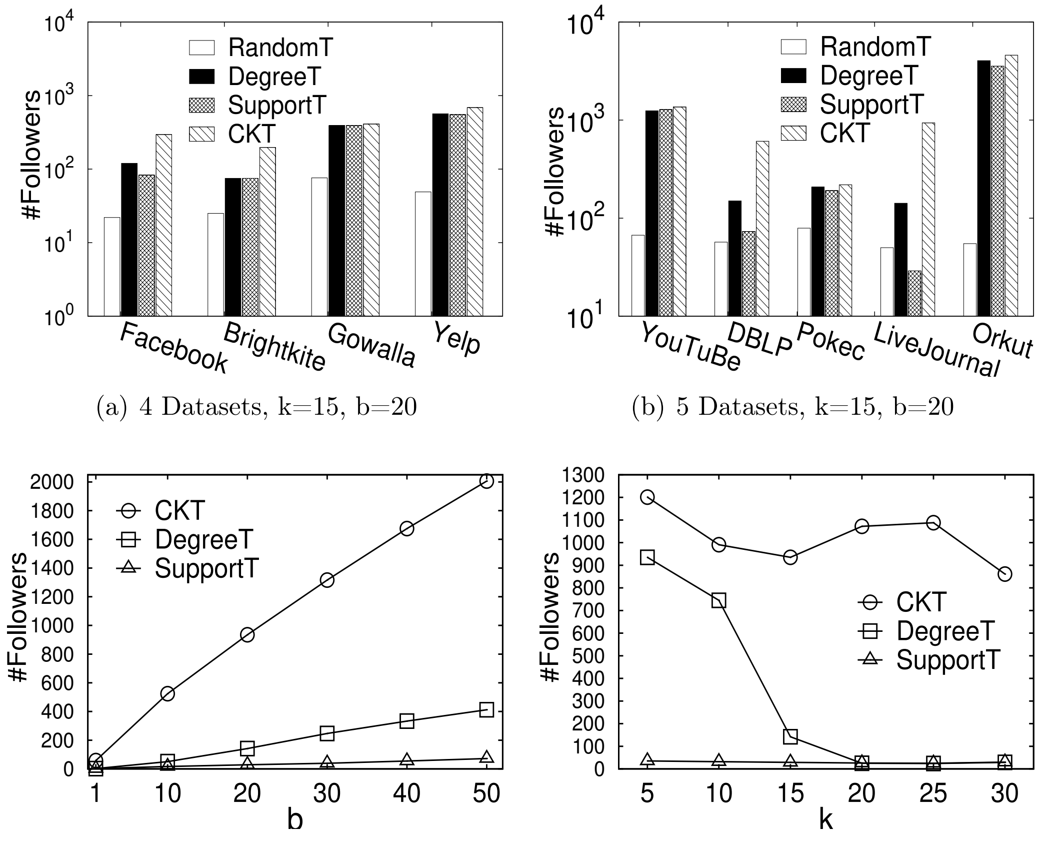
Parameters. We conduct extensive experiments by changing the support constraint value k and the budget of collapsers b. The default value of b is 20. In the experiments, the range of k varies from 5 to 50 and the range of b varies from 1 to 100.

All the programs are implemented in standard C++ and compiled with G++. All experiments are conducted on a machine with Intel Xeon 2.8GHz CPU running Redhat Linux.

3.5.2 Effectiveness

In this section, the number of the followers produced by CKT and other approaches are compared. Two case studies are presented to show an example of the CKC and CKT.

Effectiveness of the Greedy Algorithm. Figure 3.3 compares the number of followers w.r.t b collapsers identified by CKT algorithm with that of three



(a) 4 Datasets, k=15, b=20

(b) 5 Datasets, k=15, b=20

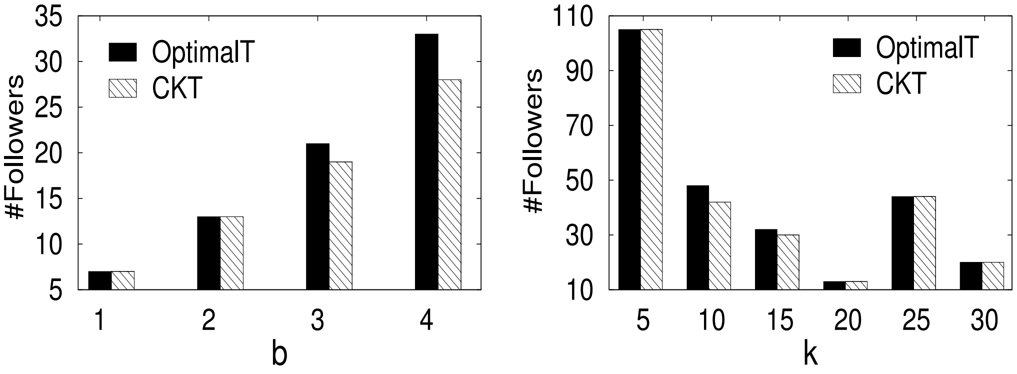
(c) LiveJournal, k=15

(d) LiveJournal, b=20

Figure 3.3: Number of the Followers

other approaches. Specifically, `RandomT` randomly chooses b collapsers in k-truss, `SupportT` chooses b collapsers from vertices with the largest numbers of triangles containing each vertex in k-truss and `DegreeT` chooses b collapsers from vertices with the largest degrees in k-truss. For `RandomT`, we give the average number of the followers for 100 independent testings. Note that the input value of k for `YouTube` is 10 because the collapsed 15-truss is always empty when $b = 20$. Figure 3.3 shows that `CKT` produces significantly more followers than the others.

To further present the effectiveness of `CKT`, we compare its performance with the optimal algorithm (`OptimalT`), which exhaustively computes collapsed k-truss for every combination of b collapsers. Figure 3.4 shows that the greedy



(a) Brightkite, k=20

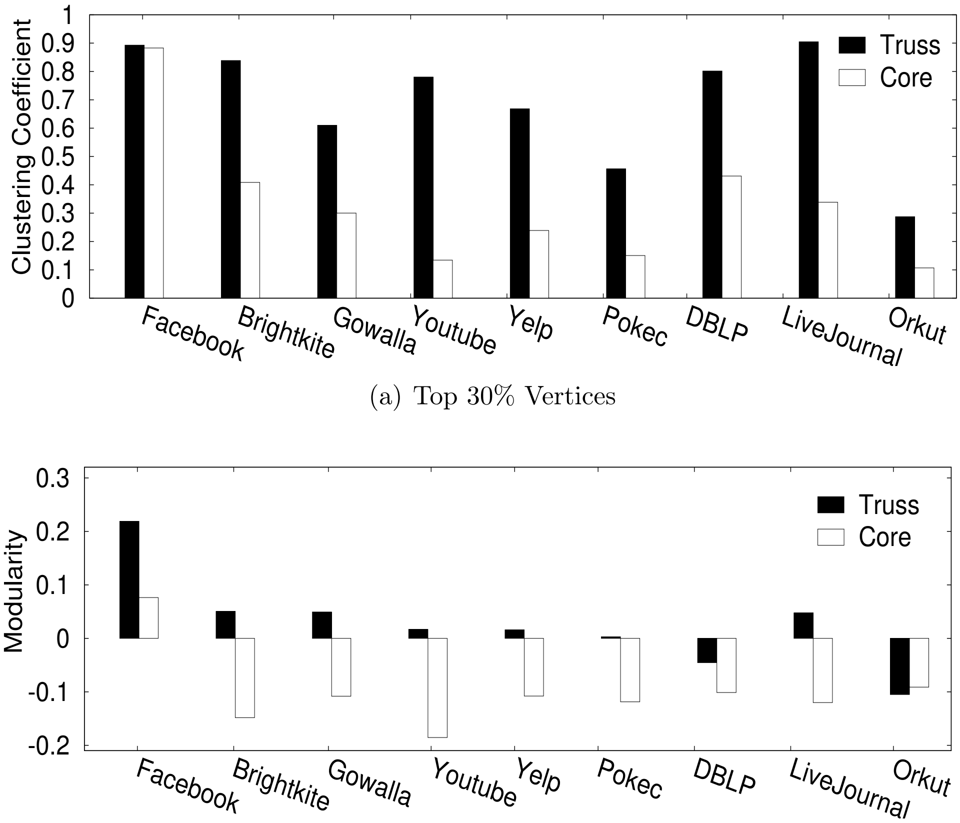
(b) Brightkite, b=2

Figure 3.4: Greedy v.s. Optimal

algorithm achieves comparable results with that of the optimal solution.

Comparison between Core and Truss. For a fair comparison on a dataset, we find a representative set of vertices. For the k-core model, we extract the top 30% vertices from the dataset, whose core numbers are largest among all vertices (resp. truss numbers for k-truss model). In Figure 3.5, for each dataset, we report the community score values: global clustering coefficient and modularity, on the induced subgraphs of the set of extracted vertices for k-core and k-truss, respectively. Note that the representative set of vertices is the user group with the highest user engagement with regard to k-core and k-truss model, respectively. Figure 3.5(a) shows the k-truss vertices possess significantly higher clustering coefficients on all datasets than the k-core vertices. Figure 3.5(b) shows the k-truss vertices also have better modularity values than the k-core vertices. The experiments indicate that utilizing k-truss may be more promising than k-core when the additional computation cost is affordable.

Comparison of Engagement Loss. In Figure 3.6, we report the engagement loss in the result of CKC and the result of CKT. For a given budget b, we consider the reduction percentage of the original noncollapsed subgraph as the engage-



(a) Top 30% Vertices

(b) Top 30% Vertices

Figure 3.5: Comparing Core and Truss

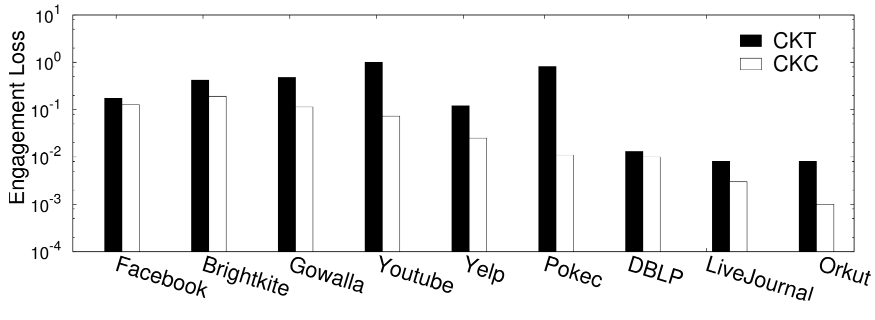


Figure 3.6: Engagement Loss of Collapsing

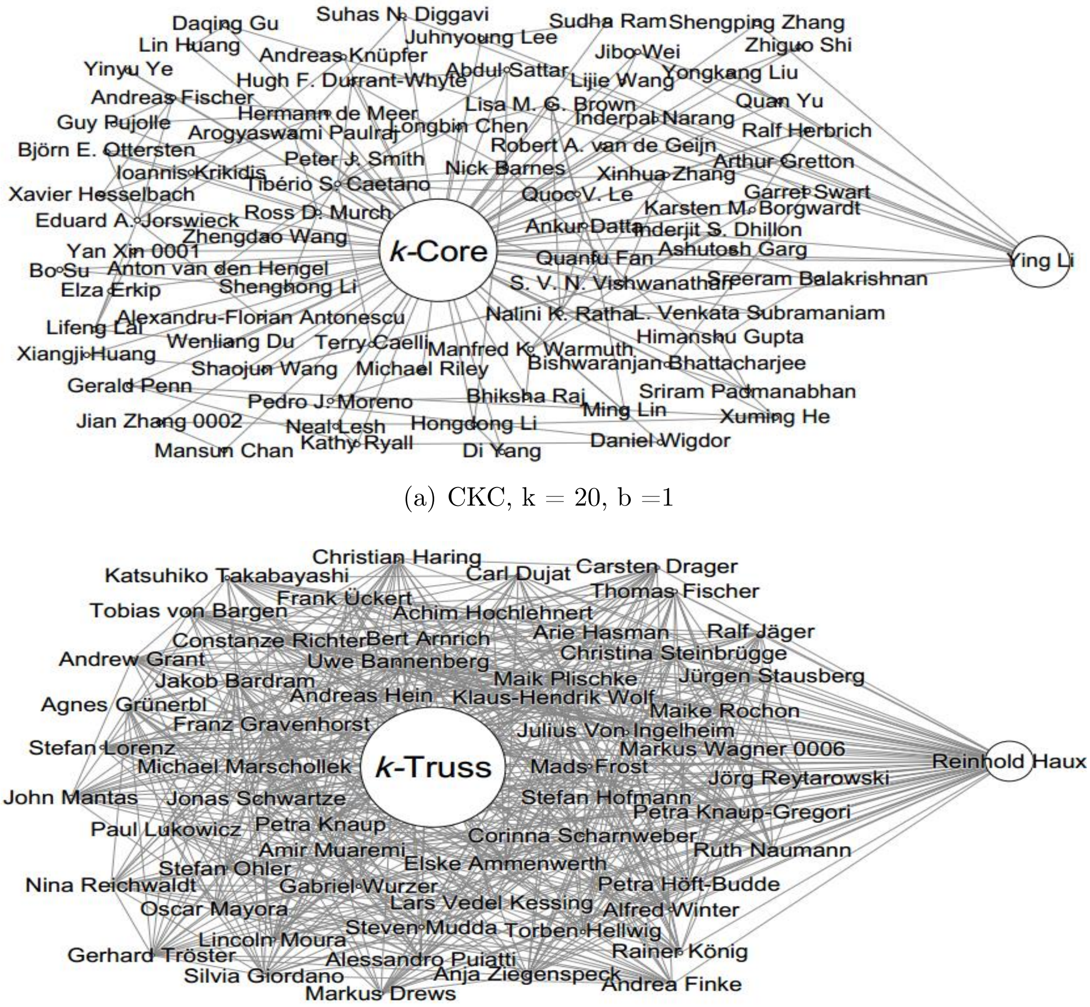
ment loss, i.e., the number of reduced vertices (i.e., followers and collapsers) divided by the number of vertices in the k-core or k-truss. Given the fact that the $(k-1)$-core is usually a super-graph of k-truss, in Figure 3.6, the values of k

we chosen are 14 for CKC and 15 for CKT, respectively. Here, the budget value b is setting as 20 for both algorithms. From the figure we can conclude that, the collapsers of CKT can reduce the original subgraph by a larger extent than CKT under the same collapsing budget. Particularly, for a same budget b that equals to 20, the collapsed 15-truss returned by CKT on Orkut has 4598 followers while there are only 2907 followers in the collapsed 14-core that returned by CKC. It shows the collapsed k-truss model may be more effective on destroying the engagement of social communities.

Case Studies of CKC and CKT. Figure 3.7 depicts the best collapser and the corresponding followers on DBLP identified by CKC and CKT, respectively, where $k = 20$ for CKC and $k = 21$ for CKT. For a clear presentation, the edges between each author and authors in k-core or k-truss are integrated as one edge. In Figure 3.7(a), it is interesting that the author“Ying Li” (the collapser) alone has 74 followers, and only 12 of them are co-authors of Li. This shows the user engagement can be critically damaged along with the leave of a few individuals and the effect of leave cascade. In Figure 3.7(b), the author“Reinhold Haux” (the collapser) has 57 followers and 56 of them are coauthors of Haux. We can see the connection between among followers and the collapser is significantly stronger in the collapsed k-truss than that of the collapsed k-core, which the user leave caused by “Ying Li” replies more on the butterfly effect. The power of collapse by Haux is stronger in the sense that his leave breaks a tighter group than the leave of Li. This shows the CKT may better model the collapse of tight-knit user groups.

3.5.3 Efficiency

Here we report the number of visited vertices and the running time for each technique and each algorithm.

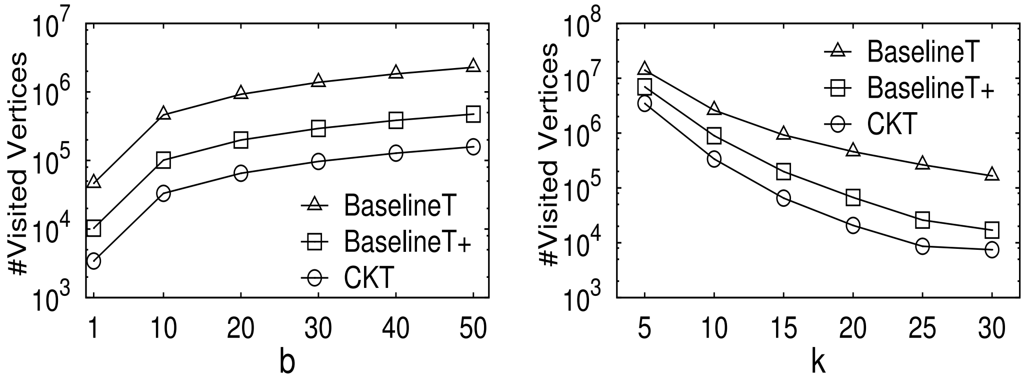


(a) CKC, k = 20, b =1

(b) CKT, k = 21, b = 1

Figure 3.7: Case Studies on *DBLP*

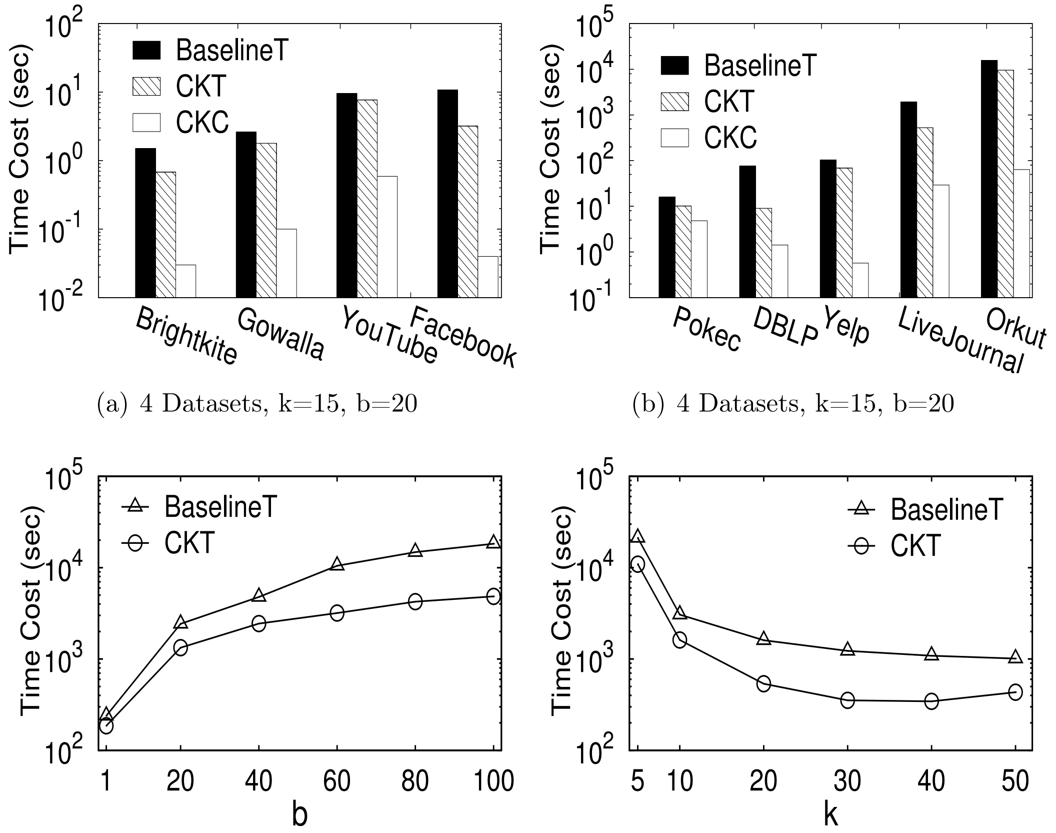
Individual Techniques Evaluation. In Figure 3.8, we report the number of the visited vertices, i.e., the size of candidate collapsers, in three algorithms, where `BaselineT+` is `BaselineT` equipped with Theorem 3. Note that `CKT` is further equipped with Theorem 4 on `BaselineT+`. It is easy to get that the number of visited vertices significantly decreases by both of the theorems on `DBLP` for different k and b.



(a) DBLP, k=15

(b) DBLP, b=20

Figure 3.8: Effectiveness of Reducing Candidate Collapsers



(a) 4 Datasets, k=15, b=20

(b) 4 Datasets, k=15, b=20

(c) LiveJournal, k=15

(d) LiveJournal, b=20

Figure 3.9: Performance of the Algorithms

Performance Evaluation. Figure 5.15 shows performance of three algorithms on all datasets with a logarithmic axis. From Figures 5.15(a)(b) we can see that `CKT` outperforms `BaselineT` on all datasets by several times, where $k = 15$ and $b = 20$. In Figure 5.15(c), the runtime of `CKT` increases when we vary b from 1 to 100. In Figure 5.15(d), the runtime for `CKT` is relatively large for extremely small k and does not change much for other values of k. We can see `CKT` significantly outperforms `BaselineT` under all settings.

3.6 Chapter Summary

In this chapter, we propose and investigate the problem of collapsed k-truss, which aims to find critical users with respect to the k-truss. These users help to sustain or destroy the strong tie communities. We prove the problem is NP-hard and inapproximate. An efficient algorithm is proposed, which significantly reduces the number of candidate vertices. Extensive experiments on real-life social networks demonstrate our model is effective and our algorithm is efficient.

Chapter 4

MINIMUM K-CORE SEARCH

4.1 Overview

k-core model is one of the most representative cohesive subgraph model. In this chapter, we investigate the problem of the minimum k-core search: given a graph G, an integer k and a set of query nodes $Q = q$, we aim to find the smallest k-core subgraph containing all the query node $q \in Q$. It has been shown that this problem is NP-hard with a huge search space, and it is very challenging to find the optimal solution. Although, there are several heuristic algorithms for this problem, they rely on simple scoring functions and there is no guarantee as to the size of the resulting subgraph, compared with the optimal solution. In this chapter, we develop an effective and efficient progressive algorithm, namely *PSA*, to provide a good trade-off between the quality of the result and the search time. Novel lower and upper bound techniques for the minimum k-core search are designed. Our extensive experiments on 12 real-life graphs demonstrate the effectiveness and efficiency of the new techniques

The work is published in [69] and the rest of this chapter is organized as follows.

Section 4.2 gives preliminary definitions and formally defines the problem. Section 4.3 describes our algorithms for the proposed minimum k-core search problem. Section 4.4 introduces the evaluation of our proposed algorithms and reports the experimental results. Section 4.5 summarizes the chapter.

4.2 Preliminary

Notation	**Definition**
G	an unweighted and undirected graph
$V(G), E(G)$	the vertex set and edge set of G
u, v	vertex in the graph
n, m	the number of vertices and edges in G
d_{max}	the largest degree value in G
$N(u, G)$	the set of adjacent vertices of u in G
$deg(u, G)$	$\|N(u, G)\|$
k	the degree constraint for k-core subgraph
$C_k(G)$	k-core of G
$P(G)$	partial solution of G

Table 4.1: Summary of Notations for Minimum k-Core Search

Problem Definition

Let $G = (V, E)$ be an undirected and unweighted graph, where V and E denote the set of vertices (nodes[1]) and edges respectively. Let $n = |V|$ and $m = |E|$ be the number of vertices and edges respectively. We use $N(u, G)$ to denote the set of adjacent vertices of u in the graph G, which is also known as the neighbor set of u in G. Let $deg(u, G)$ denote the degree of u in G, which is the number of adjacent vertices of u in G. Given a subgraph S of G, we use $V(S)$ to denote its vertices. The size of the subgraph, denoted by $|S|$, is the number of the vertices. When the context is clear, $N(u, G)$, $deg(u, G)$ and $V(S)$ are simplified to $N(u)$,

[1] In this thesis, we use node and vertex interchangeably.

$deg(u)$ and S respectively. By d_{max} we denote the largest vertex degree in the graph.

Below we have the definition of k-core subgraph based on the concept of k-core constraint.

Definition 3. ***k*-core subgraph**. *Given a graph G, a subgraph S of G is a k-core subgraph if it satisfies the* ***k*-core constraint***, i.e., every vertex in S has at least k neighbors in S.*

For presentation simplicity, we use k-core to represent k-core subgraph in this chapter when there is no ambiguity[2]. In the literature, many studies focus on the maximum k-core which is the largest induced subgraph which satisfies the k-core constraint. Note that the maximum k-core is also the *maximal* k-core because the maximal k-core of G is unique for a given k, as shown in [122].

Problem Statement. Given an undirected and unweighted graph $G = (V, E)$, a degree constraint k and a query set $Q = \{q_1, q_2, ...\}$, the minimum k-core search problem finds the **smallest** k-core subgraph containing the query set Q. Due to the NP-hardness of the problem and the huge search space involved to find the exact solution, in this chapter we aim to develop an effective and efficient progressive algorithm to enable users to achieve a good trade-off between the quality of the results (i.e., the size of the k-core subgraph retrieved) and the search time.

For presentation simplicity and the ease of understanding, we first focus on computing the minimum k-core containing one query vertex q. The proposed algorithms are adapted for the query of multiple vertices in Section 4.3.4.

[2] Note that, in some existing work, k-core is used to represent the maximal k-core.

Existing Solutions

As discussed in Section 1.1.2, the search space of the exact solution is prohibitively large, and existing solutions resort to simple heuristic algorithms. In particular, existing solutions follow two search strategies: (1) *shrink* strategy [97], called *global search*; and (2) *expansion* strategy [16, 37], called *local search*. In the global search strategy [97], the maximal k-core is the initial result, which can be efficiently computed in linear time [17]. Then the size of the resulting subgraph will be shrunken by repeatedly removing the vertex while keeping q in the resulting k-core subgraph[3]. As shown in [16], the global search method in [97] is ineffective because the size of maximal k-core is usually very large, and the quality of the resulting k-core subgraph is not competitive with the local search approaches. In recent studies [16, 37], the local search strategy is adopted which starts from the query vertex and then expands the resulting subgraph by incrementally including the most promising candidate vertex at each step following some greedy heuristics. Below, we present the state-of-the-art technique proposed in [16].

State-of-the-art. In [16], a greedy algorithm, named *S-Greedy*, is proposed to find the minimum k-core subgraph containing the query vertex. Let P denote the resulting subgraph, which is initialized as $\{q\}$. By C we denote candidate vertices which are neighbors of the vertices in P, not contained by P. In each iteration, the most promising candidate vertex is chosen for inclusion in P, and the candidate set C is updated accordingly. P is returned when every vertex in P satisfies the k-core constraint. The key of the algorithm is the design of the scoring function to measure the goodness of candidates. In [16], the authors employ two functions to qualitatively measure the advantage and disadvantage of

[3] There is no k-core subgraph containing the query vertex q if it is not in the maximal k-core.

including a vertex u in P, denoted by $p^{+}(u)$ and $p^{-}(u)$ respectively. Specifically, $p^{+}(u)$ records the number of neighbors of u in P with a degree still less than k, i.e.,

$$p^{+}(u) = |\{v|v \in N(u,P), deg(v,P) < k\}| \tag{4.1}$$

Intuitively, the larger the $p^{+}(u)$ value, the better chance that u can make more vertices in P to satisfy the k-core constraint. The cost of including u in P is that an extra number of vertices needed to make u have at least k neighbors in P, i.e.,

$$p^{-}(u) = max\{0, k - |N(u,P)|\} \tag{4.2}$$

Then the score of the vertex u is defined as $p^{+}(u) - p^{-}(u)$, where the larger value is preferred.

Discussion. The proposed algorithm is time efficient. The time complexity of the score computation at each iteration is $O(d_{max})$ as only the neighbors of the vertex u are involved and it takes $O(\ln(n))$ time to maintain the most promising candidate vertex. Thus, the time complexity of *S-Greedy* is $O(s(d_{max}+\log(n)))$ where s is the size of the resulting subgraph. In [16], the maximal k-core will be computed for the following computation and hence s is bounded by the size of maximal k-core, which is usually a large number in practice. Experiments show that *S-Greedy* significantly outperforms other competitors by means of the resulting subgraph size. However, our empirical study indicates that there is still a big gap between its resulting subgraph and the optimal solution. Moreover, all existing algorithms do not have the quality guarantee about the size of the resulting subgraph and hence it is difficult for users to make a trade-off between result quality and search time. This motivates us to develop a progressive algorithm in this chapter.

4.3 Progressive Search Algorithm

In this section, we give the solution for the minimum k-core problem.

4.3.1 Progressive Search Algorithm

Motivation and Framework. We devise a progressive search algorithm, namely *PSA*. Given a vertex set V_t as a partial solution, we find that it is feasible to compute the upper/lower bounds of the minimum size of a k-core containing V_t (the details are introduced in later sections). Thus, if we can progressively converge the size upper/lower bounds of the partial solutions in a search, it is possible to compute a k-core with guaranteed size approximation ratio regarding the size of optimally-minimum k-core. Then, the framework of *PSA* is designed as a Best-First Search (BFS) which computes the result in an expansion manner and visits the most promising branch at each search step.

A search tree of BFS is constructed along with the procedure of BFS where the root is the query vertex and every tree node contains one vertex. For each tree node t, its partial solution V_t contains the vertex in the node and all the vertices in the ancestor nodes. In *PSA*, when a tree node t is visited and t contains the vertex u, we add the child nodes of t to the search tree where each child node contains a unique neighbor of a vertex in V_t with vertex id larger than u. Then, the search step at node t is processed as follows:

(i) Lower bound driven search. For the partial solution V_t' of every child node t' of the node t, we compute a size lower bound $s^-(t)$ of the minimum k-core containing V_t' (introduced in Section 4.3.2). The next node to visit (the most promising node) is the one with the smallest lower bound $s^-(\cdot)$ from all the leaf nodes in current search tree.

(ii) Upper bound driven search. For the partial solution V_t' of every child node t' of the node t, we conduct a Depth-First Search (DFS) to compute a minimal k-core containing V'_t by heuristics (introduced in Section 4.3.3), to update the global size upper bound s^+ of the optimally-minimum k-core.

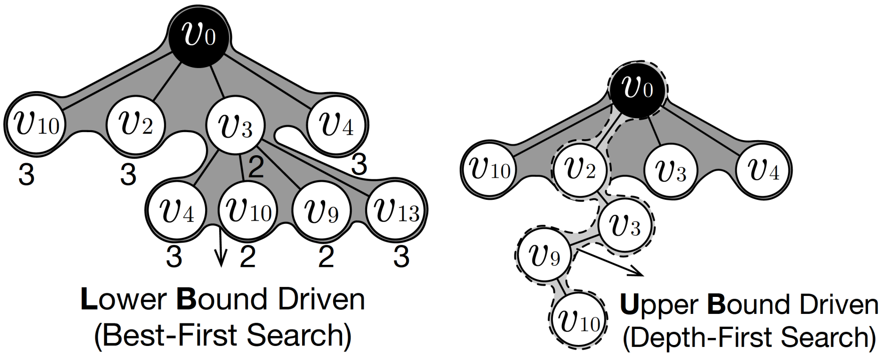
The algorithm *PSA* will return if $\frac{s^+}{s^-} \leq c$ is satisfied for current search node, where c is the approximate ratio.

Example 5. *Figure 4.1 illustrates a part of the search tree $\mathcal{T}$ of our PSA algorithm, where the root is the query vertex v_0. There are 4 neighbors of v_0 with vertex id larger than v_0: v_2, v_3, v_4 and v_{10}. When v_0 is visited, for each neighbor of v_0, we attach a child node to the current visited node in $\mathcal{T}$, where each child node contains exactly one unique neighbor of v_0. The partial solution for the node containing v_3 is $\{v_0, v_3\}$. In the BFS, suppose the node t has the smallest size lower bound of the minimum k-core containing its partial solution, t will be explored in the next search step, e.g., the node containing v_3 in Figure 4.1(a).*

For each child node with partial solution V_t, we also compute a minimal k-core containing V_t in a DFS which heuristically adds vertices to V_t. In Figure 4.1(b), a k-core induced by v_0, v_2, v_3, v_9 and v_{10} is computed for $V_t = \{v_0, v_2\}$, and the size upper bound is updated to $s^+ = 5$ if it is smaller than existing upper bound.

Iteratively, the search will return when it finds a k-core satisfying approximation ratio c regarding the size of the optimally-minimum k-core.

Algorithm 5 shows the pseudo-code of our progressive search algorithm. We use t to denote a tree node in the search tree $\mathcal{T}$. The vertex set V_t of a node t consists of the vertex $t.v$ in t and all the vertices in the ancestor nodes of t. Let s^+ denote the size upper bound of optimally-minimum k-core, and $s^-(t)$ denote the size lower bound of the minimum k-core containing V_t. Similar to the A* search [15, 113], we use a set $\mathcal{Q}$ to denote the the leaf nodes in $\mathcal{T}$ to be visited, where the key of a node t is $s^-(t)$ in ascending order. *GetUpper*(V_t) computes a minimal k-core R containing V_t by heuristics (introduced in Section 4.3.3).



(a) Lower Bound Driven (b) Upper Bound Driven

Figure 4.1: Tree Construction

Lines 2-4 initializes the above notations.

In each iteration (Lines 6-19), the node t with the smallest lower bound value $s^-(t)$ is popped at Line 6. To avoid duplicated computations, for current visited vertex $t.v$, we expand it by each neighbor u of a vertex in V_t with $id(u) > id(t.v)$ (Line 7), where $id(u)$ is the identifier of u. At Lines 8-9, for each child node t' of t, $GetLower(V_t)$ computes the size lower bound of the minimum k-core containing V_t' (introduced in Section 4.3.2), where V_t' contains $t'.v$ and all the vertices in V_t. At Line 10, $s^-(t')$ is assigned by $s^-(t)$ if $s^-(t')$ is smaller, because the size lower bound of V_t' should not be smaller than V_t. For the size upper bound $s^+(t')$, at Line 12, we conduct a heuristic search to incrementally add promising vertices to V_t' till it becomes a k-core subgraph, denoted by R', with $s^+(t') = |R'|$. The global upper bound s^+ and current best solution R will be updated by $s^+(t')$ and R' if $s^+(t') < s^+$, since smaller k-core subgraph is preferred (Lines 13-14). Note that a search branch following t' can be safely terminated (Line 17) if $s^-(t') \geq s^+$ because the resulting subgraph derived from V_t' cannot outperform the current best solution R.

The global lower bound s^- is updated as the smallest $s^-(\cdot)$ among all nodes in $\mathcal{Q}$ at Line 18. The algorithm will return if $\frac{s^+}{s^-} \leq c$ is satisfied at Line 19 or

Algorithm 5 PSA(G, k, q, c)

Input: G : a graph, k : degree constraint, c : approximation ratio, q : query vertex
Output: R : the approximate minimum k-core

```
1:  if $q \notin$ the maximal $k$-core of $G$ then return $\emptyset$
2:  $t \leftarrow$ the (root) node of search tree $\mathcal{T}$, where $t.v = q$
3:  $\mathcal{Q}.push(t)$; $R :=$ GetUpper($V_t$)
4:  $s^+ := |R|$; $s^-(t) := 1$
5:  while $\mathcal{Q} \neq \emptyset$ do
6:     $t \leftarrow \mathcal{Q}$.pop(); //$\mathcal{Q}$ is a priority queue with key on $s^-(t)$
7:     for every $u \in N(t.v)$ with $id(u) > id(t.v)$ do
8:        $t' \leftarrow$ the child node of $t$, where $t'.v := u$
9:        $s^-(t') :=$ GetLower($V_{t'}$)
10:       if $s^-(t') < s^-(t)$ then $s^-(t') := s^-(t)$
11:       if $s^-(t') < s^+$ then
12:          $R' :=$ GetUpper($V_{t'}$); $s^+(t') := |R'|$
13:          if $s^+(t') < s^+$ then
14:             $R := R'$; $s^+ := s^+(t')$
15:          $\mathcal{Q}$.push($t'$); attach child node $t'$ to $t$ in $\mathcal{T}$
16:       else
17:          $s^-(t') \leftarrow +\infty$
18:    $s^- \leftarrow$ smallest key value among nodes in $\mathcal{Q}$
19:    if $\frac{s^+}{s^-} \leq c$ then return $R$
20: return $R$
```

the queue is empty. R is returned as an approximate solution of the minimum k-core containing q.

Time complexity. The time cost of Algorithm 5 is $O(l \times (t_l + t_u))$ where l is the number of iterations and t_l (resp. t_u) denotes the computing cost of the lower (resp. upper) bounds at Lines 9 (resp. Line 12).

Correctness. Every subgraph R' retrieved at Line 12 is a k-core subgraph containing q, and hence the upper bound s^+ is correctly maintained in Algorithm 5. Given the correctness of the lower bound s^-, we have $s^- \leq |R^*| \leq s^+$, where R^* is the optimal solution. Thus, the termination of the search branches at Line 17 is safe as all possible search branches are considered for lower bound computation. When Algorithm 5 terminates, we will return R (best solution obtained so far) with $\frac{s^+}{s^-} \leq c$.

4.3.2 Lower Bounds Computation

Given a partial solution $P = \{v_1, v_2, ...\}$ and a vertex $v \in P$, we use $d(v)$ to denote the *demand* of v, i.e., the number of extra neighbors (supports) needed from vertices outside of P such that v can satisfy the degree constraint, i.e., $d(v) = \max\{k - deg(v, P), 0\}$. For every vertex $v \in P$ with $d(v) > 0$, we must include at least $d(v)$ vertices in $N(v, G) \setminus P$ such that P can be expanded to a k-core subgraph. Let M^* be the minimal subset of vertices from $V \setminus P$, such that there are at least $d(v)$ neighbors in M^* for every vertex $v \in P$. Clearly, $|P| + |M^*|$ is a size lower bound of any k-core subgraph derived from P, denoted by $L(P)$, because M^* is the minimal set of vertices required to satisfy the degree constraint for vertices in P, without considering the degree constraint for vertices in M^*.

We show that the computation of M^* for a given partial solution P can be converted to a variant of the set cover problem as follows.

Set Multi-Cover Problem [28]. Let U be the universe of n elements with $U = \{e_1, e_2, ..., e_n\}$, and there is a count for each element e_i. We use $C = \{c_1, c_2, \ldots, c_n\}$ to denote the counts of the elements. For a family of m subsets of U, $X = \{S_1, S_2, ..., S_m\}$, where S_i is a subset of U. The goal is to find a set $I \subseteq \{S_1, S_2, ..., S_m\}$, such that every element $e_i \in U$ is covered by at least c_i subsets from I.

Mapping of the problem. Each vertex v in the partial solution P with $d(v) > 0$ corresponds to an element e_i of U. The element count c_i is set to $d(v)$, i.e., the demand of v. Let $N(P)$ denote the neighbor vertices of the partial solution

P with $N(P) = (\bigcup_{v \in P \& d(v)>0} N(v,G)) \setminus P$, every neighbor vertex u in $N(P)$ corresponds to a subset S which consists of vertices in P (i.e., U) adjacent to u. By doing this, the optimal solution of the set multi-cover problem, denoted by I^*, corresponds to M^* in the above lower bound computation.

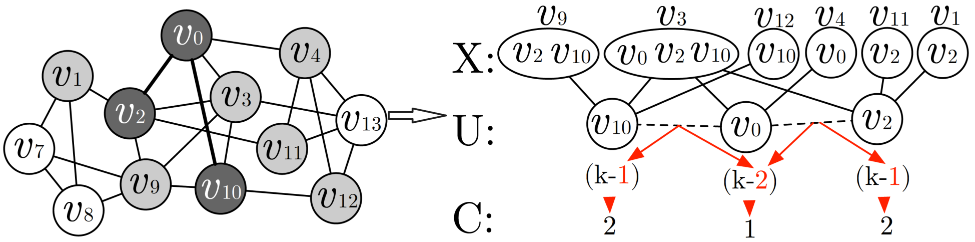


Figure 4.2: Map to Set Multi-cover

Example 6. *Consider the graph in Figure 1.2 with* $k = 3$*, and the partial solution* $P = \{v_{10}, v_2, v_0\}$*. We show the related part of Figure 1.2 in the left of Figure 4.2. For* P*, we have* $d(v_0) = 1$*,* $d(v_2) = 2$*,* $d(v_{10}) = 2$*, and the neighbor set of* P *is* $N(P) = \{v_9, v_3, v_{12}, v_4, v_{11}, v_1\}$*. Mapping to set multi-cover problem, we have* $U = \{v_{10}, v_2, v_0\}$*,* $C = \{2, 2, 1\}$*, and* $X = \{\{v_2, v_{10}\}, \{v_0, v_2, v_{10}\}, \{v_{10}\}, \{v_0\}, \{v_2\}, \{v_2\}\}$*, e.g., the set* $\{v_2, v_{10}\}$ *is associated with the vertex* $v_9 \in N(P)$ *as shown in the figure.*

Lower bound computation. For the given partial solution $P = P_u \cup \{v\}$ at Line 9 of Algorithm 5, we can construct an instance of the set multi-cover problem accordingly. We aim to derive a size lower bound of its optimal solution I^*, denoted by L. Then we can use $|P| + L$ as the lower bound since $|P| + L \leq |P| + |I^*| = |P| + |M^*|$. Here, we stress that our focus is the computation of L, not a feasible solution for the set multi-cover problem.

In the following subsections, we introduce three approaches to compute L: greedy-based approach (L^g), structure relaxation based approach (L^{sr}) and inclusion-exclusion based approach (L^{ie}).

L^g: Greedy-based Lower Bound

In [39], Dobson proposed a greedy algorithm for the set multi-cover problem with an approximation ratio $\ln(\delta)$, where δ is the largest size of the subsets in X. Specifically, the greedy algorithm repeatedly chooses a subset S_i from $X \setminus I$ that covers the largest number of elements in U , which are not yet fully covered by the current sets in I. It stops and returns the chosen subsets in I when every element $e_i \in U$ are covered by at least c_i chosen subsets. Let $|I|$ denote the number of subsets in I when the greedy algorithm terminates, $\frac{|I|}{\ln(\delta)}$ is a *lower bound* of the optimal solution $|I^*|$, i.e., $|I^*| \geq \lfloor \frac{|I|}{\ln(\delta)} \rfloor$. In this chapter, we use L^g to denote this greedy heuristic based lower bound for I^*.

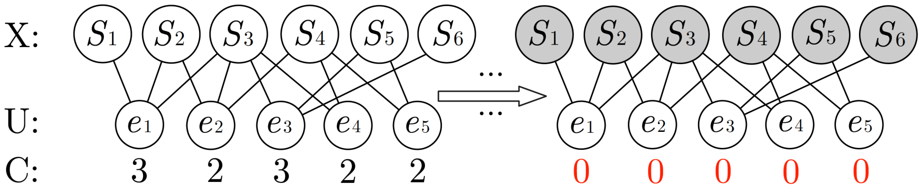


Figure 4.3: Greedy Based Lower Bound

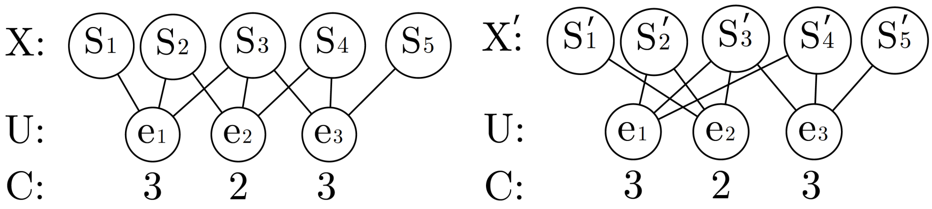
Example 7. *Consider the set multi-cover problem in Figure 4.3, where $U = \{e_1, e_2, e_3, e_4, e_5\}$, $C = \{3, 2, 3, 2, 2\}$, and $X = \{S_1, S_2, S_3, S_4, S_5, S_6\}$. The elements in a set S_i are the ones connected to S_i as shown in the figure. In a greedy manner, S_3 is the first subset to be chosen, which covers e_1, e_2, e_3, and e_4 first. The count set $C = \{2, 1, 2, 1, 2\}$ is updated. Iteratively, S_4, S_5, S_2, S_1 and S_6 are chosen sequentially. The greedy result is $I = \{S_3, S_4, S_5, S_2, S_1, S_6\}$. Since the maximum size subset in X is $|S_3| = 4$ (i.e., $\delta = 4$), the lower bound is $L^g = \lfloor \frac{|I|}{\ln(\delta)} \rfloor = \lfloor \frac{6}{\ln(4)} \rfloor = 4$.*

Discussion. Although the computation of the lower bound can be naturally mapped to the set multi-cover problem, our empirical study indicates that the

δ value is usually not small on real-life graphs, which may lead to the poor performance of the greedy-based lower bound. We note that the main focus of the greedy algorithm in [39] is to find a *feasible solution* I for the set multi-cover problem, which meanwhile can derive the lower bound of $|I^*|$. Considering that our purpose is to derive the lower bound of $|I^*|$, in the following two subsections we develop two new techniques which aim to design some heuristic approaches to *directly* derive a tighter lower bound L for $|I^*|$, without considering the feasible solution to the problem.

L^{sr}: Structure Relaxation Based Lower Bound

In this subsection, we introduce the `Structure Relaxation Based Lower Bound`, denoted by L^{sr}.



(a) (b)

Figure 4.4: L^{sr} Motivating Example

Motivation. The key idea is to directly obtain a lower bound for $|I^*|$ by re-constructing subsets $\{S_i\}$ in X (i.e., a structure relaxation if we treat set U and X as a bipartite graph as shown in Figure 4.4). Suppose we have $U = \{e_1, e_2, e_3\}$, $C = \{3, 2, 3\}$ and $X = \{S_1, S_2, S_3, S_4, S_5\}$ in the set multi-cover problem as shown in Figure 4.4(a). For instance, we have $S_4 = \{e_2, e_3\}$. We have $I^* = \{S_1, S_2, S_3, S_4, S_5\}$ in this example since all S_i must be included, i.e., $|I^*| = 5$. However, suppose we allow each S_i to include arbitrary $|S_i|$ elements (i.e., change with the size constraint), denoted by S_i'. Then we may re-construct

X, denoted by X', as shown in Figure 4.4(b) with $|S_i| = |S'_i|$. For instance, we have $S'_4 = \{e_1, e_3\}$. Now we have $I' = \{S'_2, S'_3, S'_4, S'_5\}$ (i.e., $L = 4$) to cover all elements under the new setting, with $L = |I'| \leq |I^*|$. Note that although I' is not a valid solution for the original set multi-cover problem, $|I'|$ indeed can serve as the lower bound of $|I^*|$.

Lower bound computation. In this subsection, we introduce how to re-construct X such that we can easily derive a valid lower bound for $|I^*|$. Intuitively, we should encourage the set with a large size (i.e., covering power) to include elements with high count values in the re-construction. Algorithm 6 illustrates the details of the structure relaxation based lower bound computation. Initially, Line 1 orders $\{S_i\}$ in X in descending order of their sizes. Meanwhile, the elements are organized by a maximal priority queue Q where the key of an element e_i is its count c_i (Line 2). In Lines 3-11, we will sequentially choose subsets $\{S_i\}$ from X based on their sizes. For each S_i chosen, we will use S_i to cover the first $|S_i|$ elements with the largest count values in Q, denoted by T (Line 5); that is, count c_k will be decreased by one if $e_k \in T$ (Line 7). We remove an element e_k from Q at Line 8 if it has been covered by c_k times, i.e., c_k is decreased to 0. Then the maximal priority queue is updated due to the change of count value at Line 9. Algorithm 6 will be terminated if all elements have been fully covered (Line 11) and l, i.e., $\{S_i\}$ visited so far, will be returned as the lower bound of $|I^*|$ at Line 12.

Example 8. *Figure 4.5 illustrates the procedure of a set multi-cover problem which has* $U = \{e_1, e_2, e_3, e_4, e_5\}$, $C = \{3, 2, 3, 2, 2\}$, *and* $X = \{S_1, S_2, S_3, S_4, S_5, S_6\}$ *initially. To compute* L^{sr}, *the first step is to construct* $F = \{S_3, S_4, S_2, S_5, S_1, S_6\}$ *and* $Q = \{e_1, e_3, e_2, e_4, e_5\}$. S_3 *is processed first because it has the most elements. The counts of the top* $|S_3|$ *elements (*$T = \{e_1, e_3, e_2, e_4\}$*) in* Q *decrease by one:* $C = \{2, 1, 2, 1, 2\}$. *This is followed by the update of the lower bound* l *and* Q.

Algorithm 6 StructureRelaxationLB(U, C, X)

Input: U : a universe of elements,C : the counts of the elements,
X : a family of subsets of U
Output: L^{sr}: the lower bound of $|I^*|$

```
put all $S_i \in X$ to $F$ with descending order of $|S_i|$
$l := 0$; put all $S_i \in X$ to $F$ with descending order of $|S_i|$
put all $e_i \in U$ to a maximal priority queue $Q$ with key $c_i$
for $j = 1$ to $|X|$ do
    $S_i \leftarrow$ the $j$-th subset in $F$; $l := l + 1$
    $T \leftarrow$ the top $|S_i|$ elements in $Q$
    for each $e_k$ in T do
        $c_k := c_k - 1$
        remove $e_k$ from $Q$ If $c_k = 0$
        Update $Q$ due to the change of $c_k$
    if $c_k = 0$ for every $k \in [1, |U|]$ then
        Break
return $l$
```

Iteratively, S_4, S_2, S_5, and S_1 are chosen sequentially until $c_x = 0$ for every x from 1 to $|U|$. So, we have $L^{sr} = l = 5$, which is better than $L^g = 4$.

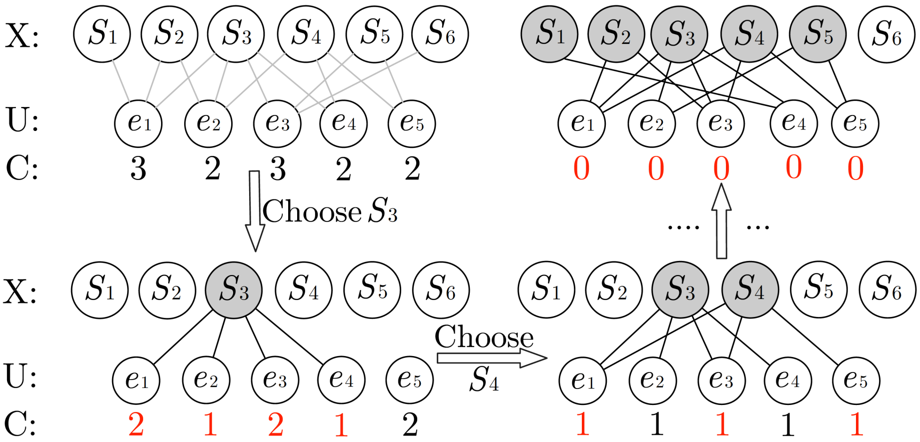


Figure 4.5: Structure Relaxation Based Lower Bound

Time complexity. The dominated cost is the update of Q, which triggers $\sum_{k=1}^{|U|} c_k$ times. The updating of Q costs $\mathcal{O}(\log(|Q|)$. Thus the time complexity of computing L^{sr} is $\mathcal{O}(\log(|U|) * \sum_{k=1}^{|U|} c_k)$.

Correctness. Note that it is not necessary that $e_k \in S_i$ at Line 5. In this sense, we re-construct S_i to cover different $|S_i|$ elements in U, and hence may end up with an invalid result set I in Algorithm 6 for the set multi-cover problem, i.e., not every element e_k in U will be covered by c_k times by I. Nevertheless, the theorem below suggests that $|I|$ obtained by Algorithm 6 is indeed a lower bound of $|I^*|$, which is sufficient for our problem.

By Theorem 5 we prove that the L^{sr} is a lower bound of I^* for the set multi-cover problem.

Theorem 5. *l obtained in Algorithm 6 is a lower bound of $|I^*|$.*

Proof. We consider the set multi-cover problem with regard to $U = \{e_1, e_2, \dots\}$, $C = \{c_1, c_2, \dots\}$, and $X = \{S_1, S_2, \dots\}$. In order to prove the theorem, we define a relaxed problem $\texttt{Rex}(U,\ C,\ H)$ as follows: Given a set of elements $U = \{e_1, e_2, \dots\}$, $C = \{c_1, c_2, \dots\}$, and size constraints $H = \{h_1, h_2, \dots\}$ with $|H| = |X|$ and $h_i = |S_i|$, we want to find a set $I = \{I_{k_1}, I_{k_2}, \dots\}$ where $1 \leq k_1 < k_2 < \dots \leq |H|$, $I_{k_i} \subseteq U$, and $|I_{k_i}| \leq h_{k_i}$, such that e_j is contained in c_j subsets from I and $|I|$ is minimized. Suppose I is the optimal solution of $\texttt{Rex}(U,\ C,\ H)$, and I^* is the optimal solution for the set multi-cover problem regarding U, C and X, we have $|I| \leq |I^*|$, since $\texttt{Rex}(U,\ C,\ H)$ is a relaxation for the set multi-cover problem (each set in I can contain any elements in U with only a size constraint). Next, we prove that our Algorithm 6 obtains the optimal solution I for the problem $\texttt{Rex}(U,\ C,\ H)$.

Suppose w.l.o.g. that $h_1 \geq h_2 \geq \dots$ and $c_1 \geq c_2 \geq \dots$, we show that there exists an optimal solution with $k_1 = 1$ i.e., the size of I_{k_1} in I is bounded by h_1. This is because if $k_1 \neq 1$ we can replace k_1 with 1 and the constraint on I_{k_1} is increased, thus we do not obtain a worse solution. Next, we prove that there exists an optimal solution s.t. I_1 contains the first h_1 elements in U (i.e., the top-h_1 elements with the largest c_i values). Suppose there is an optimal solution with $I' = \{I'_1, I'_2, \dots\}$ s.t. in I'_1 there exists an $x < y$ with $e_i \in I'_1$ for $i < x$, $e_x \notin I'_1$ and $e_y \in I'_1$, we prove that we can construct an optimal solution I with $e_i \in I_1$ for $i \leq x$. Since C is sorted in non-increasing order, we have $c_x \geq c_y$, therefore, we can always find an I'_j $(j > 1)$ that contains e_x but does not contain e_y. So if we move e_x from I'_j to I'_1 and move e_y from I'_1 to I'_j, the constraints on I will not change and in this way we obtain a solution I that is not worse than I' and has $e_i \in I_1$ for $i \leq x$. As a result, statement that I_1 contains the first h_1 elements in U holds. Similarly, we can prove that I_2 contains the first h_2 elements in U' by deducing those covered by I_1. Since the selection procedure is the same as that in Algorithm 6, our algorithm obtains the optimal solution for the problem Rex(U, C, H). Therefore, the theorem holds. □

L^{ie}: Inclusion-exclusion Based Lower Bound

The structure relaxation based approach only considers the size constraint when it re-constructs X, which may end up with a loose lower-bound. In this subsection, we introduce an inclusion-exclusion based lower bound, denoted by L^{ie}.

Motivation. Suppose we have $U = \{e_1, e_2, e_3\}$ and $C = \{3, 3, 3\}$. Let $S(e)$ denote the sets of $\{S_i\}$ in X which contain element e (i.e., $e \in S_i$). Meanwhile, for optimal solution I^*, we use $I^*(e)$ to denote the set of $\{S_i\}$ in I^* which contain the element e. Clearly, for any optimal solution I^* regarding a given X, we must have $|I^*(e_i)| \geq 3$ for elements $\{e_1, e_2, e_3\}$ since $C = \{3, 3, 3\}$. According to the *inclusion-exclusion principle*, we have

$$\begin{aligned}|I^*| &= |I^*(e_1) \cup I^*(e_2) \cup I^*(e_3)| \\ &= |I^*(e_1)| + |I^*(e_2)| + |I^*(e_3)| \\ &- |I^*(e_1) \cap I^*(e_2)| - |I^*(e_1) \cap I^*(e_3)| - |I^*(e_2) \cap I^*(e_3)| \\ &+ |I^*(e_1) \cap I^*(e_2) \cap I^*(e_3)|\end{aligned} \tag{4.3}$$

Given the fact that $I^*(e) \subseteq S(e)$ for any element $e \in U$, we now investigate the possible value of $|I^*|$ if some knowledge about the overlap size among $\{S(e)\}$ in X is available. If there is no constraint regarding the size of $|S(e_i) \cap S(e_j)|$ $(1 \leq i < j \leq 3)$ and $|S(e_1) \cap S(e_2) \cap S(e_3)|$, we can easily come up with a X such that $|I^*| = 3 + 3 + 3 - 3 - 3 - 3 + 3 = 3$ as shown in Figure 4.6(a), in which each S_i $(1 \leq i \leq 3)$ covers all three elements. While if we know that $|S(e_i) \cap S(e_j)| = 0$ $(1 \leq i < j \leq 3)$, Figure 4.6(b) constructs a X such that $|I^*| = 3 + 3 + 3 - 0 - 0 - 0 + 0 = 9$. Note that we have $|I^*(e_i) \cap I^*(e_j)| = 0$ $(1 \leq i < j \leq 3)$ and $|I^*(e_1) \cap I^*(e_2) \cap I^*(e_3)| = 0$ immediately. Using a similar argument, we may have $|I^*| = 3 + 3 + 3 - 1 - 1 - 1 + 0 = 6$ in Figure 4.6(c) if $|S(e_i) \cap S(e_j)| = 1$ $(1 \leq i < j \leq 3)$ and $|S(e_1) \cap S(e_2) \cap S(e_3)| = 0$. In Figure 4.6(d), we have $|I^*| = 3+3+3-1-1-1+1 = 7$ given $|S(e_i) \cap S(e_j)| = 1$ $(1 \leq i < j \leq 3)$ and $|S(e_1) \cap S(e_2) \cap S(e_3)| = 1$.

The above example implies that we may come up with a tighter lower bound of $|I^*|$ if we know the overlap sizes of sets among $\{S(e)\}$. For computing efficiency, we only consider the intersection size of pairwise sets, i.e., $|S(e_i) \cap S(e_j)|$ for all $1 \leq i < j \leq n$, in our inclusion-exclusion based lower bound computation.

Lower bound computation. Algorithm 7 gives details of the inclusion-exclusion based lower bound computation. Initially, the elements in U are organized by a

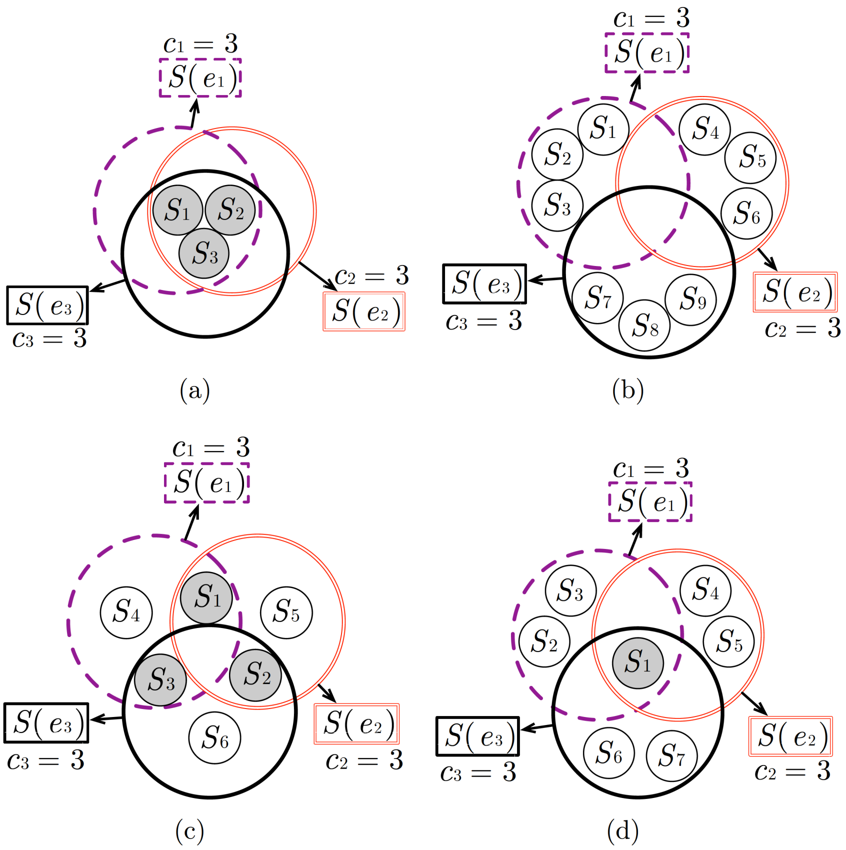


Figure 4.6: L^{ie} Motivating Example

maximal priority queue Q, the key of an element e_i is its count c_i (Line 1). We generate a set $S(e_i)$ for each element. $S(e_i)$ denotes the sets of $S \in X$ which cover the element e_i, i.e., $e_i \in S$. In Lines 3-11, we sequentially choose an element e_i from Q based on its count. For each e_i chosen, the other elements in Q, that is, the count c_k will be decreased by Δ, where $\Delta = \min\{S(e_i) \cap S(e_k), c_i\}$. We remove e_k from Q at Line 8 if $c_k \leq 0$, i.e., e_k has been covered c_k times. Then the maximal priority queue is updated due to the change in the count value (Line 11). Algorithm 7 will be terminated if all elements have been fully covered, i.e., Q is empty and l will be returned as the lower bound of $|I^*|$ at Line 12.

Algorithm 7 Inclusion-Exclusion LB(U, C, X)

Input: U : a universe of elements need to be covered,C : the count of every element in U, X : a family of subsets of U
Output: L^{ie} : *Inclusion-exclusion based lower bound*

```
 1: put all $e_i \in U$ to a maximal priority queue $Q$ with key $c_i$
 2: generate $S(e_i)$ for each element $e_i \in U$; $l := 0$
 3: while $Q \neq \emptyset$ do
 4:     $e_i \leftarrow Q$.top(); $Q$.pop(); $l := l + c_i$; $c_i := 0$
 5:     for each $e_j \in Q$ do
 6:         $\Delta := \min\{|S(e_j) \cap S(e_i)|, c_i\}$
 7:         if $\Delta \geq c_j$ then
 8:             remove $e_j$ from $Q$
 9:         else
10:             $c_j := c_j - \Delta$
11:         Update $Q$
12: return $l$
```

Example 9. *Consider the set multi-cover problem in Figure 4.7, where* $U = \{e_1, e_2, e_3, e_4, e_5\}$*,* $C = \{3, 2, 3, 2, 2\}$*, and* $X = \{S_1, S_2, S_3, S_4, S_5, S_6\}$*, where* $S_i \subseteq U$*. Let* $S(e_i)$ *denote the sets of* $\{S_j\}$ *in* X *that cover the element* e_i *(i.e.,* $e_i \in S_j$*), shown as* $S(e_1) = \{S_1, S_2, S_3\}$ *in the figure. We construct the priority queue* $Q = \{e_1, e_3, e_2, e_4, e_5\}$*. Here* e_1 *is processed first and* $l = 0 + c_1 = 3$*. The set* C *is updated accordingly:* $\Delta_2 = \min\{|S(e_1) \cap S(e_2)|, c_1\} = 2$*. Thus* $\Delta_2 \geq c_2$*,* e_2 *is removed from* Q*. We have* $C = \{0, 0, 2, 1, 2\}$ *and* $Q = \{e_3, e_5, e_4\}$ *after completing the first round. Next,* e_3 *is processed and* l *is updated to* $l = 3 + c_3 = 5$*. Similarly, we have* $C = \{0, 0, 0, 0, 1\}$ *and* $Q = \{e_5\}$ *after finishing this round. Then* e_5 *is processed in the next step, and* $l = 6$*, followed by* $c_5 = 0$ *and* $Q = \emptyset$*. As a result, the lower bound is* $L^{ie} = l = 6$*.*

Time complexity. The dominant cost of Algorithm 7 is the computation of $S(e_i) \cap S(e_j)$ at Line 6. With the help of the data structure HashMap in c++ STL, we can get $|S(e_i) \cap S(e_j)|$ in $|X| \times \mathcal{O}(1)$ time, where $\mathcal{O}(1)$ is the time

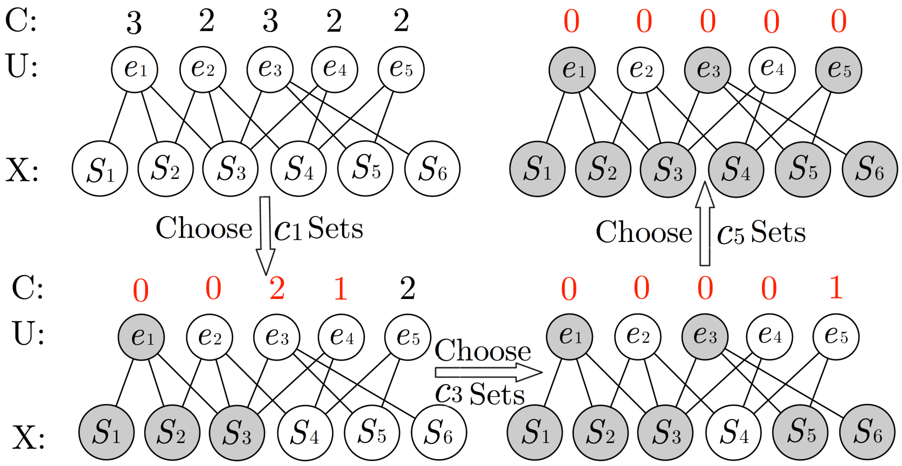


Figure 4.7: Inclusion-exclusion Based Lower Bound

complexity of one search in HashMap. As each $S(e_i)$ can be pre-sorted, the cost is $O(|U|^2|X|)$.

Space complexity. As each set $S(e_i)$ takes $\mathcal{O}(|X|)$ space at most. Each HashMap takes $\mathcal{O}(|X|)$ space. The space complexity is $O(|U||X|)$ at most.

Correctness. The following theorem shows the correctness of the inclusion-exclusion based lower bound computation.

Theorem 6. *l obtained in Algorithm 7 is a lower bound of $|I^*|$.*

Proof. Suppose the set multi-cover problem is related to $U = \{e_1, e_2, e_3, \ldots, e_h\}$, $C = \{c_1, c_2, c_3, \ldots\}$, and $X = \{S_1, S_2, \ldots\}$, and the optimal solution for this problem is I^*. We use $S(e_i)$ to denote the sets of $\{S_j\}$ in X which contain element e_i (i.e., $e_i \in S_j$).

We prove the theorem by induction. Suppose the theorem holds for any U with $h-1$ elements, we prove the theorem holds for any U with h elements. We start from an element $e_1 \in U$: Suppose in the optimal solution I^*, we use l_1^* subsets to cover e_1, apparently, we have $l_1^* \geq c_1$. Let l_0^* be the number of subsets to cover all other elements in $U \setminus \{e_1\}$ in the optimal solution. So we have $|I^*| = l_1^* + l_0^*$. Suppose after selecting l_1^* subsets $I^*(e_1)$ to cover e_1, other elements in U need to be covered $C^* = \{c_2^*, c_3^*, \dots\}$ times. We have $c_2^* = c_2 - |I^*(e_1) \cap S(e_2)|$, $c_3^* = c_3 - |I^*(e_1) \cap S(e_3)|$, $\dots$.

For Algorithm 7, firstly, we select l_1 subsets to cover e_1, i.e., $l_1 = c_1$, apparently, $l_1^* \geq l_1$. Let l_0 be the number of subsets that we get from Algorithm 7 to cover all elements in $U \setminus \{e_1\}$. Thus, we have $l = l_1 + l_0$. Suppose after selecting l_1 subsets to cover e_1, the elements in $U \setminus \{e_1\}$ need to be covered by $C' = \{c'_2, c'_3, \dots\}$ times, and according to the algorithm, we have $c'_2 = c_2 - |S(e_1) \cap S(e_2)|$, $c'_3 = c_3 - |S(e_1) \cap S(e_3)|$, $\dots$. Since $I^*(e_1) \subseteq S(e_1)$, we have $c_2^* \geq c'_2$, $c_3^* \geq c'_3$, $\dots$.

According to the induction condition, l_0 is the lower bound of the problem with regard to $U' = \{e'_2, e'_3, \dots, e'_h\}$ with $S(e'_i) = S(e_i) \setminus S(e_1)$, and $C' = \{c'_2, c'_3, \dots\}$. We know that l_0^* is the optimal solution to the problem with regard to $U' = \{e'_2, e'_3, \dots, e'_h\}$ with $S(e'_i) = S(e_i) \setminus S(e_1)$, and $C^* = \{c_2^*, c_3^*, \dots\}$. Therefore, l_0 is also a lower bound of the problem with regard to U' and C^*, i.e., $l_0 \leq l_0^*$.

Consequently, we have $l = l_1 + l_0 \leq l_1^* + l_0^* = |I^*|$. The theorem holds. □

(4) Putting the lower bounds together

Intuitively, we can use three lower bounds L^g, L^{sr} and L^{ie} together and choose the maximal one as the lower bound at Line 9 of Algorithm 5, i.e., $L = \max(L^g, L^{sr}, L^{ie})$. However, as shown in our empirical study, the greedy-based lower bound L^g is not useful because of its poor performance in terms of pruning power and computing cost. Thus, we only use the structure relaxation based lower bound L^{sr} and the inclusion-exclusion based lower bound L^{ie} in our implementation of *PSA*, i.e., $L = \max(L^{sr}, L^{ie})$.

4.3.3 Upper Bound Computation

Any k-core subgraph containing query q can immediately serve as an upper bound of the optimal solution. Thus, we may continuously maintain an upper bound s^+ at Line 14 of Algorithm 5 by keeping the smallest size k-core containing q obtained so far. In this subsection, we present a heuristic algorithm to find a k-core starting from a partial solution P in Algorithm 5. The key idea is to incrementally expand the partial solution based on the *importance* of the vertices. Particularly, we use the concept of the *onion layer* proposed in [122] to prioritize the access order. The onion layer can be regarded as a refinement of the k-shell.

Given a graph G, we use $C_k(G)$ to denote the maximal k-core of G. The **coreness** of a vertex u, denoted by $cn(u)$, is the largest value of k such that $u \in C_k(G)$. The k-shell of G, denoted by $S_k(G)$, is the set of vertices with core number k, i.e., $S_k(G) = C_k(G) \setminus C_{k+1}(G)$.

Coreness has been used to measure the importance/influence of the vertices in a variety of applications (e.g., [18, 122]). Given the k-shell $S_k(G)$ of graph G, where every vertex in $S_k(G)$ has the same coreness value k, the onion layer further partitions these vertices into different layers using an *onion-peeling-like* algorithm. Given the maximal k-core $C_k(G)$, we use the $L_{k,0}$ to denote the vertices in $C_k(G)$ which do not satisfy the $k+1$ degree constraint, i.e., $L_{k,0} = \{u \mid deg(u, C_k(G)) < k+1\}$. Then we use $L_{k,1}$ to denote the vertices which do not satisfy the $k+1$ degree constraint after removing the vertices in $L_{k,0}$. The above procedure is repeated until $L_{k,l+1}$ is empty, and we have $S_k(G) = \cup_{0\leq j\leq l} L_{k,j}$. Through this procedure, each vertex $u \in G$ will be assigned to a unique onion layer $L_{k,j}$. Given any two vertices u and v in onion layers L_{k_1,j_1} and L_{k_2,j_2} respectively, we say u has *higher onion layer* than v if $k_1 > k_2$ or $j_1 > j_2$ given $k_1 = k_2$.

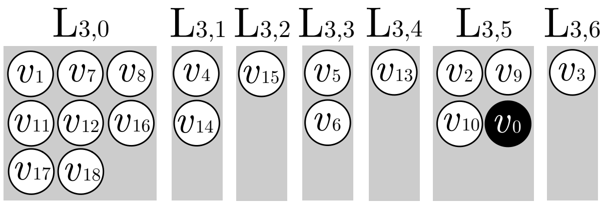


Figure 4.8: Onion Layer Structure, $k = 3$

Example 10. *Consider the graph in Figure 1.2 with* $k = 3$*. The graph itself is a 3-core. We start to peel the graph: firstly,* $L_{3,0} = \{v_1, v_7, v_8, v_{11}, v_{12}, v_{16}, v_{17}, v_{18}\}$*, because these vertices do not satisfy the* $k + 1$ *degree constraint. After removing the vertices in* $L_{3,0}$*,* v_4 *and* v_{14} *do not satisfy the* $k + 1$ *degree constraint, so we have* $L_{3,1} = \{v_4, v_{14}\}$*; Repeating the above procedure, we have* $L_{3,2}, \ldots, L_{3,6}$ *as shown in Figure 4.8 because* $v_{13} \in L_{3,4}$ *and* $v_1 \in L_{3,0}$*,* v_{13} *have a higher onion layer than* v_1*.*

Onion Layer Based Upper Bound. Algorithm 8 illustrates details of the onion-layer based upper bound computation. In Lines 1-6, we sequentially assess the vertices $\{v\}$ in partial solution P which still do not satisfy the degree constraint, i.e., $deg(v, P) < k$. For each vertex v, we add $r = k - \deg(v, P)$ neighbors of v that are not in P with the highest *onion-layer* values, denoted by X (Line 4). The partial solution P will be updated accordingly with $P := P \cup X$ (Line 5). At Line 6, we ensure the resulting subgraph is a small k-core subgraph by removing the redundant vertices in P. Note that we say a vertex $u \neq q$ is redundant if every neighbor v of u in P has $deg(v, P) \geq k + 1$. Algorithm 8 will be terminated if P grows to a k-core subgraph (Line 1), and P is a minimal k-core since we remove the redundant vertices. Then P is returned at Line 7, which can serve as the upper bound of the minimum k-core containing q.

Note that the construction of onion layer is independent to the query vertex q, which can be pre-computed in $O(m + n)$ time as shown in [122].

Algorithm 8 L-Greedy(P)

Input: P : the partial solution **Output**: a k-core subgraph containing P

```
1: while $P$ is not a $k$-core subgraph do
2:     $v \leftarrow$ the vertex in $P$ with $deg(v, P) < k$
3:     $r := k - deg(v, P)$
4:     $X \leftarrow r$ neighbors of $v$ not in $P$ with highest onion-layer values
5:     $P := P \cup X$
6:     remove redundant vertices from $P$
7: return $P$
```

Time complexity. For each vertex v accessed at Lines 1-6, it takes $O(d_{max} \log(d_{max}))$ time to find the set X at Line 4, where d_{max} is the highest degree in graph G. Meanwhile, we only need to consider the redundancy for its neighbor vertices, with cost $O(d^2_{max})$. Therefore, the time complexity of Algorithm 8 is $O(|U| \times d^2_{max})$.

Correctness. The correctness of the algorithm is immediate because the set P returned is a k-core subgraph.

4.3.4 Processing Multiple Query Vertices

In addition to one simple query vertex q, it is interesting to consider to find a minimum k-core subgraph containing a set of query vertices $Q = \{q_1, q_2, ...\}$, which is also NP-hard. The *PSA* algorithm proposed in this chapter can be easily extended to tackle this problem by enforcing every partial solution to contain all query vertices. The computation of the lower and upper bounds are the same. Algorithm 5 can be adjusted as follows.

(i) Replace the input with "$PSA(G, k, Q, c)$";

(ii) Replace Line 1 with "**if** $\exists q \in Q$ *and* $q \notin$ the maximal k-core of G **then**";

(iii) Replace the initialization (Lines 2-4) part with "(1) $u \leftarrow$ the vertex in Q with the largest id; $t \leftarrow$ the (root) node of search tree $\mathcal{T}$; $t.v = u$; (2) $V_t \leftarrow Q$

with increasing order of $id(x)$; $s^-(t)$:= **GetLower**(V_t); $\mathcal{Q}.push(t)$; and (3) $R :=$ **GetUpper**(V_t); $s^+ := |R|$".

Time complexity. The time cost of Algorithm 5 is $\mathcal{O}(l \times (t_l + t_u))$, where l is the number of iterations. The main difference between the multiple query vertices and one query vertex is the number of iterations involved and the computing cost of lower (resp. upper) bounds is the same at each iteration. Thus, the time cost of updated Algorithm 5 for multiple query vertices is also $\mathcal{O}(l \times (t_l + t_u))$.

4.4 Performance Studies

In this section we evaluate the efficiency and effectiveness of our proposed algorithms.

4.4.1 Experimental Setting

Algorithms. We evaluate the following algorithms:

`S-Greedy`: The state-of-the-art technique [16] for the problem of minimum k-core search, which is outlined in Section 4.2. Authors in [16] kindly provided the source code implemented by Java and we rewrite it in C++ for fair comparison of search time.

`L-Greedy`: A greedy algorithm which only uses the upper bound technique proposed in Section 4.3.3; that is, Algorithm 8 is invoked with partial solution $P = \{q\}$ and the resulting k-core subgraph will be returned.

`PSA`: The progressive search framework (Algorithm 5) proposed in Section 4.3.1, equipped with two lower bound techniques L^{sr} (Section 4.3.2-(2)) and L^{ie} (Section 4.3.2-(3)) and the upper bound technique *L-Greedy*.

Dataset	Nodes	Edges	d_{avg}	d_{max}	k_{max}
Email	36,692	183,831	5.01	1383	43
Epinion	75,879	405,740	5.3	3044	67
Gowalla	99,563	456,830	21.9	9967	43
DBLP	510,297	1,186,302	2.3	340	25
Yelp	249,440	1,781,885	7.1	3812	105
Yeast	12,782	2,007,134	157.1	3322	277
YouTube	1,134,890	2,987,624	2.6	28754	51
Google	875,713	4,322,051	4.9	6332	44
Wiki	2,394,385	4,659,565	1.9	100029	131
Flickr	1,715,255	15,555,041	9.1	27236	568
UK2002	18,483,186	292,243,663	15.8	194955	943
Webbase	118,142,155	1,019,903,190	8.6	816127	1506

Table 4.2: Statistics of Datasets for Minimum k-Core Search

Note that *PSA* does not use the greedy-based lower bound L^g (Section 4.3.2-(1)) due to its poor performance in terms of effectiveness and efficiency.

PSA-S: The progressive search framework equipped with *S-Greedy* as the upper bound and L^g (Section 4.3.2) as the lower bound.

PSA-L: The progressive search framework equipped with *L-Greedy* as the upper bound and L^g as the lower bound.

Datasets. Twelve real-life networks are deployed in our experiments. Yeast is a protein-protein interaction network, which can be found in [99]. Flickr is the network for sharing content [81]. DBLP is a co-author network, where each vertex represents an author and there is an edge between two authors iff they have co-authored at least 3 papers. Gowalla is a location-based social network. Vertices without check-ins and their incident edges are removed. We transfer directed edges to undirected edges. UK2002 and Webbase are downloaded from WebGraph[4]. Yelp is downloaded from Yelp[5]. The remaining datasets are

[4] http://webgraph.di.unimi.it

[5] https://www.yelp.com.au/dataset/challenge

downloaded from SNAP[6]. Table 5.3 shows the statistics of all the datasets.

Parameters and query generation. Algorithms are evaluated by varying the degree constraint k and the approximation ratio c. The default values for k and c are 10 and 1.8 respectively. In the experiments, k varies from 5 to 25 and c varies from 4.0 to 1.6. In the experiments, each query vertex is randomly selected from the k-core. In each test, 100 queries are randomly generated and their average result size or search time is reported. Each computation is terminated if it cannot finish within half an hour.

All the programs are implemented in C++ and compiled with G++. All experiments are conducted on a machine with Intel Xeon 2.3GHz CPU running Redhat Linux.

4.4.2 Effectiveness

Metrics	**diameter**	**degree**	**density**	**CC**	**avg. size**	**Engage**
k-Core	8.12	24.45	0.001	0.32	23196.29	35%
k-truss	4.61	6.92	0.63	0.33	13020.85	43%
k-Ecc	7.62	25.18	0.02	0.33	22561.75	36%
k-Clique	5.18	15.49	0.38	0.63	15775.21	42%
Graph Clustering	13.87	8.59	0.11	0.32	81232.53	30%
Min k-Core	3.39	10.09	0.37	0.57	52.02	49%

Table 4.3: Comparison for CS solutions on Gowalla

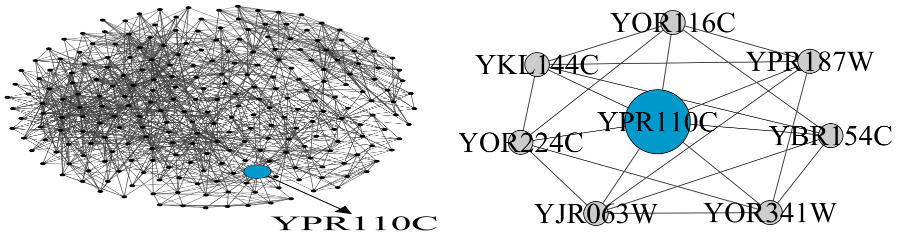
Comparison with community search methods

We compare several representative algorithms introduced in the survey of community search (CS) [44] (i.e., k-Core, k-truss, k-Ecc, k-Clique) and the structural graph clustering [107] with our *PSA* on Gowalla dataset. For each query vertex v, the setting of the input k is same to the evaluation in the survey [44], e.g.,

[6] http://snap.stanford.edu

k is the coreness of v for k-core and k is the trussness of v for k-truss. For the clustering algorithm, the similarity threshold for two nodes is set as $\epsilon = 0.6$ (the default value in [107]) and the threshold for the cardinality of ϵ-neighborhood is set as $\mu = cn(v)$. To evaluate the result, we use all the quality metrics used in the survey [44] and two additional important metrics: diameter, average degree, density, clustering coefficient (CC) and average size of the communities, as well as the user engagement (Engage). Engage is the proportion of active users in a community where a user is active if the user has at least 1 check-in during 2018-08-04T00:00:01 and 2018-08-10T23:59:59. Note that the chosen query vertices are also active during this period.

Table 4.3 shows the communities returned by our *PSA* achieve high scores in the evaluation. Particularly, the average size of our communities is much smaller than the others where the size is reasonable in real-life. Besides, our k-cores have the best diameter and Engage scores, benefiting from the k-core constraint and the small community size.



(a) S-Greedy (b) PSA

Figure 4.9: Case Studies on Yeast, $k = 5$

Case studies

We compare *S-Greedy* and *PSA* ($k = 5$, $c = 1.8$) on `Yeast` to show different results from two approaches. As shown in Figure 4.9(a), *S-Greedy* returns a large k-core subgraph where some vertices are not closely connected. Besides, only 32 of the 254 proteins detected by *S-Greedy* have at least one common function with the query protein YPR110C. In Figure 4.9(b), *PSA* identifies 7 nearby proteins of YPR110C where each of these proteins has at least one common function with the query protein.

Alg.	***k*-Core**	***k*-truss**	***k*-Ecc**	***k*-Clique**	**Graph Clustering**	**Min *k*-Core**
%	61%	70%	61%	86%	52%	95%

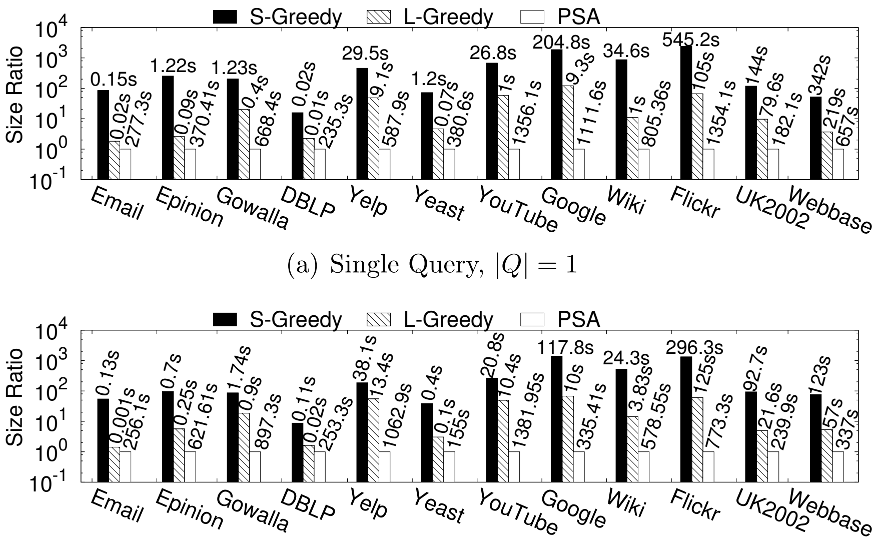
Table 4.4: Percentage of Homogeneous Communities

To further evaluate the homophily property of vertices in a minimal k-core subgraph, we compare it with the community search methods introduced in Section 4.4.2, in terms of the common protein functions in the communities. Specifically, we say a community C is homogeneous iff the query protein has the function f which is the most common function among all the proteins in C. It can be verified by the enrichment analysis in David[7] [60]. Table 4.4 reports the percentage of homogeneous protein communities over all detected communities for each method, where our *PSA* has the best performance.

Evaluation on result size

Evaluation on different graphs. Figure 4.10 reports the average size ratios of k-core subgraphs returned by *S-Greedy*, *L-Greedy* and *PSA* on all the datasets, where $k = 10$ and $c = 1.8$. The average size of the k-cores returned by *PSA* is regarded as the base value, i.e., 1. The sizes of the k-core subgraphs found by *S-Greedy* and *L-Greedy* are much larger than *PSA*, because of the theoretical approximation guarantee of *PSA*. Regarding result size, *L-Greedy* outperforms *S-Greedy* due to better heuristics. On `Flickr` with $|Q| = 1$, the size of a k-core returned by *S-Greedy* (resp. *L-Greedy*) is usually larger than $80,000$ (resp. 2000),

[7] https://david.ncifcrf.gov



(a) Single Query, $|Q| = 1$

(b) Multi Queries, $|Q| = 4$

Figure 4.10: Average Result Size, $k = 10$, $c = 1.8$

while the size of a k-core from *PSA* is usually smaller than 100. In Figure 4.10, we also mark the running time of each setting. To find a k-core with a practically small size, *PSA* costs more runtime in computation than the other algorithms. Given the benefit of consistently retrieving small size k-cores, it is cost-effective to apply the *PSA* algorithm. Given a larger dataset, we observe that the time cost of *PSA* is not necessarily higher. This is because, although the increase of graph size leads to larger search space, *PSA* may have a larger possibility to fast complete the query on a $(k-1)$-clique in the search, in which we only need to explore 1-hop neighbors.

Varying the approximate ratio c. Figures 4.11 shows the average sizes of k-core subgraphs returned by *PSA* on all the datasets when c varies from 4.0 to 1.6. Note that the size of a k-core returned by *S-Greedy* or *L-Greedy* does not have a approximation guarantee, and is irrelevant with the value of c. As expected, the average size decreases when c became smaller. It implies a reasonable ratio can be applied to trade-off the efficiency and the quality.

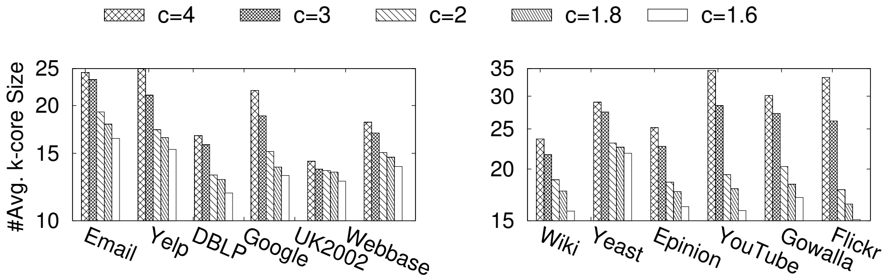
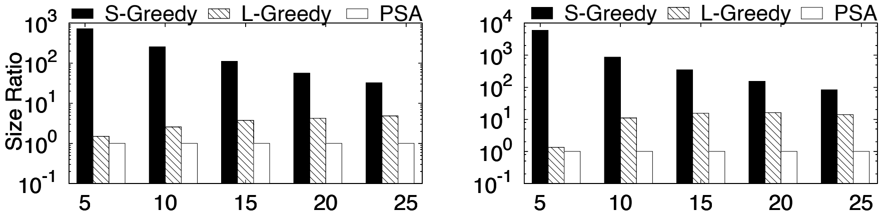


Figure 4.11: Effect of c, $k = 10$



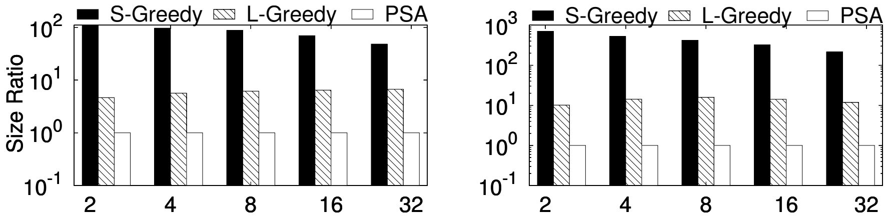
(a) Epinion

(b) Wiki

Figure 4.12: Effect of k, $c = 1.8$

Varying the degree constraint k. Figure 4.12 reports the average size of a k-core returned by *S-Greedy*, *L-Greedy* and *PSA*, by varying k from 5 to 25 on `Epinion` and `Wiki`. The margin of *PSA* and *S-Greedy* becomes smaller when the input of k grows, because the size of a k-core from *S-Greedy* is related to the size of maximal k-core which decreases with a larger k, and our *PSA* performs well on different k. The margin of *PSA* and *L-Greedy* becomes larger when the input of k grows, because it is harder for *L-Greedy* to constraint the size of returned k-core given a larger k.

Varying the size of query set $|Q|$. Figure 4.13 shows the results of three algorithms by varying the size of query set from 2 to 32 on `Epinion` and `Wiki`. The average size of a k-core returned by *L-Greedy* or *S-Greedy* is much larger than *PSA*. As expected, the minimal k-core found by *PSA* becomes larger given more query vertices. Since the minimal k-core returned by *S-Greedy* is more

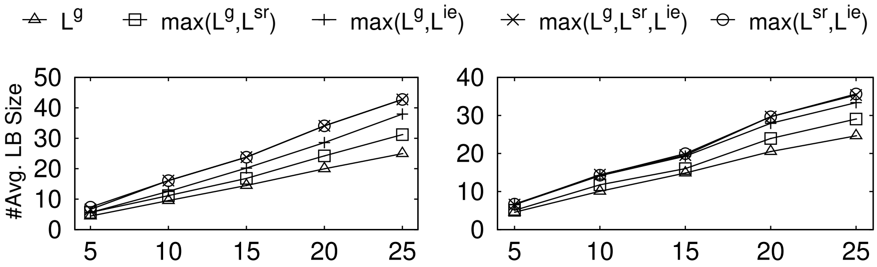


(a) Epinion (b) Wiki

Figure 4.13: Effect of $|Q|$, $k = 10$

related to the size of maximal k-core, the margin between *PSA* and *S-Greedy* becomes smaller with a larger query set.

4.4.3 Efficiency



(a) Epinion (b) UK2002

Figure 4.14: Effect of Lower Bounds, $c = 1.8$

Evaluating lower bounds. Figure 4.14 shows the size of different lower bounds derived from the lower bound technique L^g, L^{sr} and L^{ie}. We report the average size over 100 independent queries. Note that a large value is preferred in the evaluation of the lower bound. The performance of the single L^g is beaten by all the other methods for all the evaluated settings. This is not surprising because it is difficult for a greedy algorithm to get a good approximation due to the factor $\ln(\delta)$, where δ is the largest size of the subsets in X (Section 4.3.2). The bound $\max\{L^g, L^{sr}\}$ is to choose the larger one from L^g and L^{sr}, which outperforms

L^g. As analyzed in Section 4.3.2, sometimes the lower bound derived by L^{sr} is not tight enough, while it can improve the result from L^g. We further evaluate $\max\{L^g, L^{sr}, L^{ie}\}$ which adds L^{ie} to $\max\{L^g, L^{sr}\}$. Figure 4.14 shows that $\max\{L^g, L^{sr}, L^{ie}\}$ achieves the best results. It implies that L^{ie} produces a tighter lower bound, compared with L^{sr}. The reason is that L^{ie} considers more information of the subsets in X. To further validate the effectiveness of different bounds, we evaluate $\max\{L^g, L^{ie}\}$. Compared with $\max\{L^g, L^{sr}\}$, $\max\{L^g, L^{ie}\}$ produces a tighter bound. Considering the computing cost of L^g is expensive, we also investigate the performance of $\max\{L^{sr}, L^{ie}\}$, which shows similar performance with $\max\{L^g, L^{sr}, L^{ie}\}$. Because the time cost of $\max\{L^{sr}, L^{ie}\}$ is dominated by L^{ie}, we deploy $\max\{L^{sr}, L^{ie}\}$ in our *PSA* algorithm.

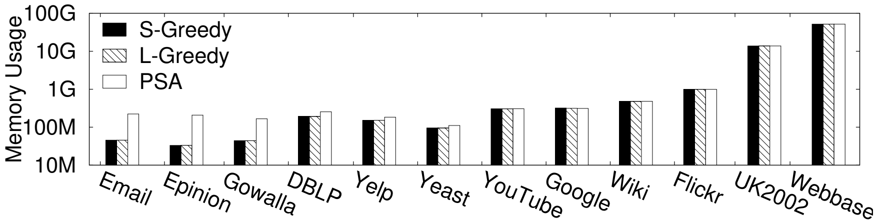
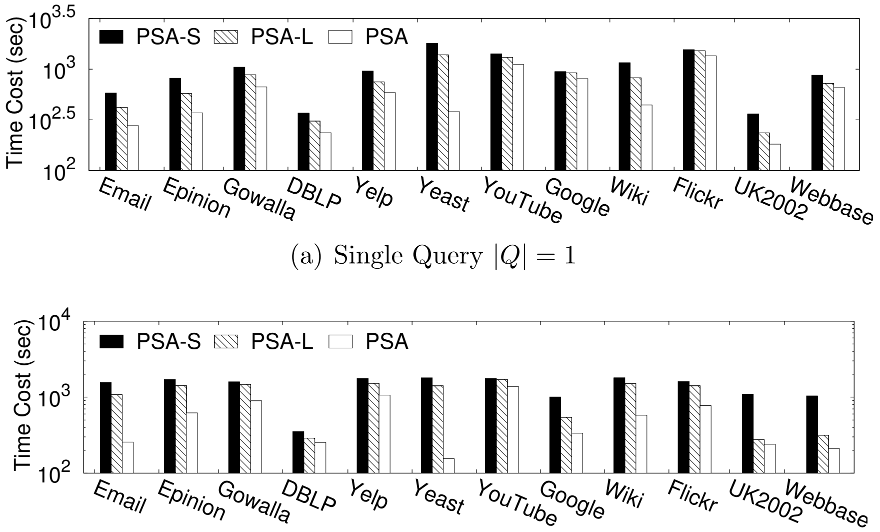


Figure 4.15: Memory Cost, $k = 10$, $|Q| = 1$

Evaluating Memory Cost. Figure 4.15 shows the memory cost of *S-Greedy*, *L-Greedy*, and *PSA*, respectively. The memory cost gradually grows with a larger graph, because the graph data dominates the memory cost. For instance, when the edge number of a graph is larger than $3M$, the memory cost of three methods is almost the same. For graphs with very small size, the lower bound computation dominates the memory cost, while it helps to fast return a minimal k-core.

Evaluation on different graphs. Figure 4.16 shows the time cost of *PSA-S*, *PSA-L* and *PSA*, respectively, when $k = 10$ and $c = 1.8$. *PSA-L* always outperforms *PSA-S* because the upper bound technique *L-Greedy* can better

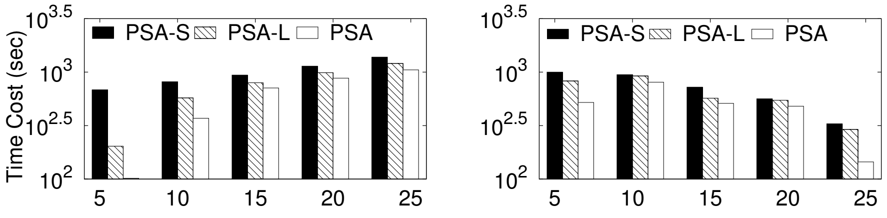


(a) Single Query $|Q| = 1$

(b) Multi Queries $|Q| = 4$

Figure 4.16: Performance of PSA, $k = 10$, $c = 1.8$

reduce the size of the resulting k-core subgraphs, compared with *S-Greedy*. As shown in Figure 4.14, the combination of L^{sr} and L^{ie} performs better than L^{g}. Thus, in Figure 4.16, *PSA* outperforms *PSA-L* in all the settings, benefiting from the superior upper bound technique (*L-Greedy*) and two lower bound techniques L^{sr} and L^{ie}.

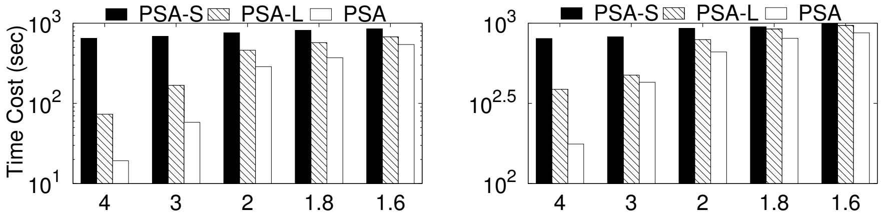


(a) Epinion

(b) Google

Figure 4.17: Varying k, $c = 1.8$

Varying degree constraint k. Figure 4.17 shows the average running time of the three algorithms when k varies from 5 to 25. The result is the average from 100 queries of a random vertex. Similar to Figure 4.16, *PSA-L* always outperforms *PSA-S*. The increase of degree threshold k may affect the time cost in two ways: (1) more vertices need to be explored s.t. the runtime increases; and (2) the number of candidate vertices decreases because the size of maximal k-core becomes smaller s.t. the runtime decreases. In Figure 4.17(a), (1) dominates the effect on different k. In Figure 4.17(b), when $k > 10$, (2) becomes the major effect for different k values. Thus, the trend of a larger input of k is different on different settings.



(a) Epinion (b) Google

Figure 4.18: Varying c, $k = 10$

Varying the approximate ratio c. In this experiment, we vary c from 4.0 to 1.6 and set $k = 10$. Figure 4.18 shows the average time cost for 100 independent queries on `Epinion` and `Google`. *PSA* is the most efficient among the three algorithms, and *PSA-L* consistently outperforms the *PSA-S*. With a relatively large c, *PSA* can find a proper result in a reasonable time. For example, when $c = 4.0$, the time cost of *PSA-L* or *PSA* on `Epinion` is less than 100 seconds. The size of the resulting k-core can be smaller if we allow more time cost for the algorithm, which leads to a trade-off between result quality and efficiency.

Varying the query size $|Q|$. Figure 4.19 shows the results of the three algorithms when $|Q|$ varies from 2 to 32 on `Wiki`. The figure shows that *PSA* outperforms *PSA-L* and *PSA-S* under all the settings. *PSA-S* performs the worst due to the limited performance of its upper/lower bound techniques. Given a larger query set, both the upper bound and the lower bound values may increase

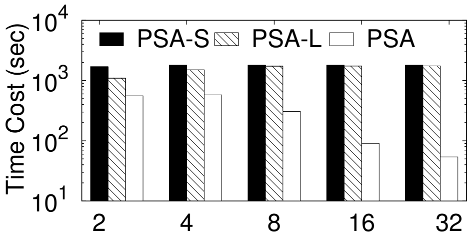


Figure 4.19: Effect of $|Q|$

while usually the lower bound has a larger increase ratio due to its smaller value. Thus, the runtime of *PSA* decreases with a larger query set Q in this experiment.

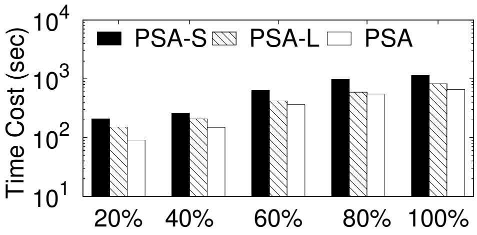


Figure 4.20: Varying $|V|$

Scalability evaluation on `Webbase`. In scalability test, we vary the graph size by randomly sampling vertices from 20% to 100% in the original graph. For each sampled vertex set, we obtain the induced subgraph of the vertex set as the input data. Figure 4.20 shows the running time of *PSA-S*, *PSA-L* and *PSA* under different percentages, respectively. It is shown that the grows of running time of the algorithms are not very significant with respect to the growth of the graph size. This is because, although the increase of graph size leads to larger search space, we may have a higher chance to quickly end up the query with a $(k-1)$-clique in the search, in which we only need to explore 1-hop neighbors.

Evaluating different kinds of queries. We compare the efficiency of our *PSA* algorithm for three kinds of query vertices: *Hubs*, *Nonhubs*, and *Random*. *Hubs*

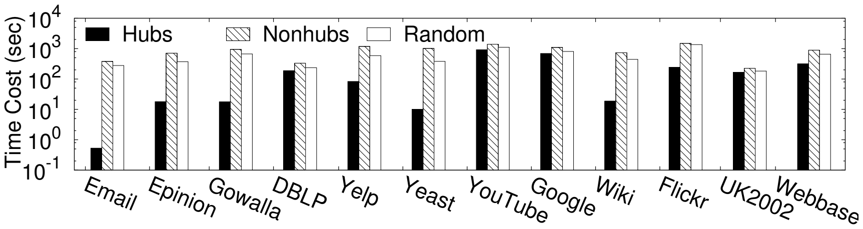


Figure 4.21: Different Query Types, $k = 10$, $c = 1.8$

contains the hub vertices which are the top 10% vertices of the k-core w.r.t the vertex degree in k-core. *Nonhubs* contains the other 90% vertices in the graph. *Random* is to randomly choose query vertices from the k-core. We randomly select 100 vertices from each category. Figure 4.21 shows that the running time for queries on *Hubs* is much faster, because the query on hub vertices is more likely to quickly end up with a clique, in which only direct neighbors need to be accessed. Queries on *Random* are faster than queries on *Nonhubs* because there are both hub and non-hub vertices in *Random*.

4.5 Chapter Summary

In this chapter, we investigated the problem of the minimum k-core search which aims to find the smallest k-core subgraph containing the query vertex set. Some existing studies on this problem are based on greedy heuristic following a variety of scoring functions, while they cannot provide any theoretical guarantee on the quality of the results. Motivated by these issues, we propose a progressive search algorithm *PSA*, based on novel lower and upper bound techniques. The proposed algorithm achieves a good trade-off between the quality of the result and the search time. Our extensive experiments on 12 real-life graphs demonstrate the effectiveness and efficiency of our proposed techniques. Our progressive framework and lower/upper bound heuristic techniques can shed light on other cohesive subgraph/community mining problems. In addition to the k-core model, the framework can be used by other cohesive subgraph models such ad k-truss and k-ecc due to different focuses of the modes. We will leave this as future work.

Chapter 5

DISCOVERING FORTRESS-LIKE COHESIVE SUBGRAPHS

5.1 Overview

Given the elegant property of the p-cohesion, in this chapter, we conduct a comprehensive theoretical analysis on the complexity of the problem and developing efficient computing algorithms. We focus on the minimal p-cohesion because they are elementary units of p-cohesion and the combination of multiple minimal p-cohesions is a larger p-cohesion. In this chapter, we study two representative problems with regard to the p-cohesion model: minimum p-cohesion search and diversified p-cohesion enumeration.

> **Minimum p-cohesion search** aims to find the smallest p-cohesion containing the given query vertex, i.e., the p-cohesion with the smallest number of vertices to which a user (query vertex) belongs. We show this problem is NP-hard, and some heuristics are proposed to efficiently identify a p-cohesion with a small size for the given query vertex.

> **Diversified p-cohesion enumeration** aims to find a set of diversified p-cohesions which can cover as many vertices as possible. Here, we consider diversity because, in practice, the user may be overwhelmed by the exponential number of minimal p-cohesions. Thus, in this chapter, we design efficient algorithms to find a set of disjoint minimal p-cohesions.

Besides, we demonstrate that the discovered minimal p-cohesions can be utilized to solve the MinSeed problem: finding a smallest set of initial adopters (seeds) such that all the network users are eventually influenced, under the contagion model. Extensive experiments on 8 real-life social networks verify the effectiveness of this model and the efficiency of our algorithms. The work is under revision, and the revision has been submitted.

The rest of this chapter is organized as follows.

Section 5.2 gives preliminary definitions and formally defines the problem. Section 5.3 gives the proof of the complexity for the MPCS problem and introduces our solution for the MPCS problem. Section 5.4 gives the proof of the complexity for the diversified enumeration problem, introduces our solution for the disjoint minimal p-cohesion enumeration, and gives our solution for the MinSeed problem. Section 5.5 evaluates our proposed algorithms. Section 5.6 summarizes the chapter.

5.2 Preliminary

Assume there is an unweighted and undirected graph $G = (V, E)$ with n vertices and m edges, where V (resp. E) represents the set of vertices (resp. edges). S denotes a subgraph of G. Let $N(u, S)$ denote the set of adjacent vertices (i.e., neighbors) of u in S. Let $deg(u, S)$ denote the number of vertices in S which are adjacent to u. We may omit the target graph in notations when the context

Notation	**Definition**
G	an unweighted and undirected graph
S	a subgraph of G
$V(S)$	the vertex set of S
$E(S)$	the edge set of S
$G(V)$	the induced subgraph of a vertex set V on G
u, v; e	a vertex in G; an edge in G
n, m	the number of vertices and edges in G
$N(u, S)$	the adjacent vertices of u in S
$deg(u, S)$	the number of adjacent vertices of u in S
$E(u)$	the set of incident edges to u in G
D	a set of seed vertices

Table 5.1: Summary of Notations

is clear, e.g., using $deg(u)$ instead of $deg(u, G)$. We assume that if a vertex is deleted, its incident edges are also deleted accordingly.

The contagion model in [41, 82] defines the following cascading condition when some vertices are activated and the others are not.

Definition 4. ***Cascading Condition***. *Given a graph G and a cascading threshold $r \in (0, 1)$, an inactivated vertex $u \in V(G)$ will be activated iff the number of activated neighbors of u is at least $\lceil r \times deg(u, G) \rceil$.*

The cascades in a graph may imply some interesting subgraphs which are named p-cohesions.

Definition 5. ***p-Cohesion***. *Given a graph G and a real number $p \in (0, 1)$, a connected subgraph S is a p-cohesion of G, denoted by $C_p(G)$, where $deg(u, S) \geq \lceil p \times deg(u, G) \rceil$ for every vertex $u \in S$.*

Easley and Kleinberg [41] prove that the p-cohesion subgraphs have the "fortress" property to hamper the progression of information cascades.

Property 1. ***"Fortress"***. *Given a graph G and a real number $p \in (0, 1)$, for an arbitrary p-cohesion S of G, according to the Cascading Condition with threshold $r > 1 - p$, no vertex in S can be activated when all vertices in S are inactivated initially, even if all the vertices in $V(G) \setminus V(S)$ are activated.*

Definition 6. ***Minimal p-cohesion***. *Given a graph G and a real number $p \in (0, 1)$, a p-cohesion S of G is called minimal if it is an elementary unit of p-cohesion, i.e., every proper subgraph $S' \subset S$ is not a p-cohesion.*

Problem Statement. Given an undirected and unweighted graph G, and a real number $p \in (0, 1)$, we aim to develop algorithms for the following two representative problems: (1) $\underline{\text{M}}$inimum $\underline{p}$-$\underline{\text{C}}$ohesion $\underline{\text{S}}$earch (MPCS): Given a query vertex q, find the smallest p-cohesion containing q in G; and (2) $\underline{\text{D}}$iversified $\underline{p}$-$\underline{\text{C}}$ohesion $\underline{\text{E}}$numeration (DPCE): Enumerate a set of disjoint minimal p-cohesions.

5.3 Minimum p-Cohesion Search

In this section, we study the minimum p-cohesion search problem, i.e., MPCS. We firstly analyze the complexity of the problem, and then present the solution.

5.3.1 Problem Analysis

In this section, we give the proof of the complexity for the MPCS problem.

Theorem 7. *The MPCS problem does not admit a PTAS, unless P = NP.*

Proof. We prove the hardness of MPCS by the hardness of minimum p-cohesion computation (denoted by MinPC) which is to find the smallest p-cohesion in G (without query vertices). If there is a polynomial solution for MPCS, we can immediately come up with a polynomial solution to find the minimum p-cohesion in G (MinPC) by conducting MPCS on every vertex.

For MinPC, we show a reduction from Vertex Cover in cubic graphs which does not admit a PTAS (Polynomial Time Approximation Scheme) in cubic graphs, unless P = NP [47, 4]. Given an arbitrary cubic graph H as an instance of vertex cover, with $|V(H)| = n$, we construct an instance G of MinPC as follows.

We may assume that $|E(H)| = \frac{3n}{2} = 3 \times 2^l$ for some integer l, without loss of generality [8]. As shown in Figure 5.1, we construct a rooted tree T with a height of $l + 1$, where the root vertex has 3 child vertices. Except for the root and leaves, every vertex in T has 1 parent vertex and 2 child vertices. So T contains 3×2^l leaf vertices, where the leaf set is denoted by L. Then we add a copy of L, denoted by F, and add a Hamiltonian cycle on L and F as in the figure. The elements in F are identified with the elements in $E(H)$. Now we add a set A which is a copy of $V(H)$ with the identifications. Then we add an edge between a vertex $u \in A$ and a vertex $e \in F$ (i.e., corresponding to an edge in H) if and only if u is incident to e in H.

Let ST denote a *star-like* subgraph which is induced by a center vertex and its $\lceil \frac{|V(T)|+|V(F)|+|V(A)|}{p} \rceil$ neighbors. Then for every vertex u in T (resp. F), we connect u to every center vertex in $k - 3$ (resp. $max(k - 4, 0)$) copies of ST, where $k = \lfloor \frac{2}{p} \rfloor + 1$. The construction is completed.

Every vertex in the star-like subgraph has a degree of 1 except the center vertices. When $\frac{2}{3} < p < 1$, we have that every vertex in F has a degree of 4 in G and every vertex in $V(G) \setminus F$ has a degree of 3 in G. When $0 < p \leq \frac{2}{3}$, every vertex in T and F has a degree of k in G. If a p-cohesion C_p contains a vertex in one of the star-like subgraphs, then C_p has to contain more than $|V(T)| + |V(F)| + |V(A)|$ vertices which makes C_p not a minimum p-cohesion, as shown in the following.

According to the definition of p-cohesion, if a vertex $u \in T \cup F$ is in a minimum p-cohesion (C_p) of G, u has at least 3 neighbors in C_p. Based on the construction, if a minimum p-cohesion contains a vertex in $T \cup F$, it has to contain every vertex in $T \cup F$. Furthermore, a minimum p-cohesion cannot only contain the vertices in $T \cup F$ or A. So a minimum p-cohesion is induced by all the vertices in $T \cup F$ and a smallest subset of vertices in A such that each vertex in F has at least a degree of 3 in the p-cohesion. Clearly, such a p-cohesion contains at most $|V(T)| + |V(F)| + |V(A)|$ number of vertices, where $|V(T)| = 1 + 3 * 2^0 + 3 * 2^1 + \cdots + 3 * 2^l = 3 * 2^{l+1} - 2 = \frac{6n}{2} - 2$, $|V(F)| = L = 3 * 2^l = \frac{3n}{2}$, and $|V(A)| = n$. Then the minimum p-cohesion search problem on G is exactly the Vertex Cover problem in H. Thus, we have that

$$OPT_{MinPC}(G) = OPT_{VC}(H) + |V(T)| + |V(F)| = OPT_{VC} + \frac{9n}{2} - 2 \qquad (5.1)$$

where $OPT_{MinPC}(G)$(resp. $OPT_{VC}(H)$) is the optimal solution for the MinPC in our constructed graph G (resp. Vertex Cover in cubic graph H). We may omit the target graph in notations when the context is clear, e.g., using OPT_{MinPC} instead of $OPT_{MinPC}(G)$. Note that, any solution of MinPC in G of size $SOLN_{MinPC}$ induces a solution of Vertex Cover problem in H of size $SOLN_{VC} = SOLN_{MinPC} - \frac{9n}{2} + 2$. Suppose that we assume MinPC admits a PTAS, i.e., for any $\alpha > 0$ we can find a solution for MinPC in polynomial time in graph G of size $SOLN_{MinPC} \leq (1 + \alpha) * OPT_{MinPC}$. Therefore, we can find a solution of Vertex Cover in H in polynomial time with size

$$SOLN_{VC} = SOLN_{MinPC} - \frac{9n}{2} + 2 \leq (1 + \alpha) * OPT_{MinPC} - \frac{9n}{2} + 2 \qquad (5.2)$$

Based on Equation 5.1 and Equation 5.2, we have

$$SOLN_{VC} \leq (1+\alpha) * OPT_{VC} + \alpha * (\frac{9n}{2} - 2) \tag{5.3}$$

Since H is a cubic graph, any solution of Vertex Cover in H has at least $\frac{|E(H)|}{3} = \frac{n}{2}$ nodes, i.e., $\frac{n}{2} \leq OPT_{VC}$. Using this in Equation 5.3, we have

$$SOLN_{VC} \leq (1+\alpha) * OPT_{VC} + \alpha * (\frac{9n}{2} - 2) \leq (1 + 10 * \alpha) * OPT_{VC} \tag{5.4}$$

In all, the existing of a PTAS for MinPC would imply the existence of a PTAS for Vertex Cover in the cubic graphs, which is impossible unless P = NP [4].

□

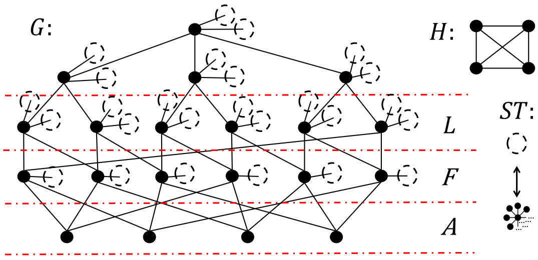


Figure 5.1: Construction Example, $p = \frac{1}{2}$

5.3.2 The Search Algorithms

Exact Search Algorithm

In order to solve the MPCS problem, enumerating all minimal p-cohesion containing the query vertex would be the way to find the exact solution.

Algorithm 9 and Algorithm 10 show the pseudo-code of our algorithms for enumerating all p-cohesions containing the query vertex. The p-cohesions with the minimum size is the solution for our MPCS problem. Algorithm 9 is based

Algorithm 9 SubPCExact(R, P, X, p, c)

Input: R, P, X : three data sets, p : a real number in $(0, 1)$, c : a positive integer
Output:$MinC$: a p-cohesion containing q

```
if all $v \in R$ are with $deg(v, G(R)) \geq \lceil p \times deg(v, G) \rceil$ then
    if $|R| < c$ then
        $MinC = G(R)$, $c = |R|$
        return
for each $v \in P$ do
    $R \leftarrow R \cup \{v\}$, $X \leftarrow X \cup \{v\}$
    SubPCExact($R$, $(P \cup \{N(v) - R\} - X)$, $X$, $p$, $c$)
    $R \leftarrow R - \{v\}$
return $MinC$
```

on the framework proposed by Bron and Kerbosch [19], which is a backtracking algorithm to solve the maximal clique enumeration problem.

In Algorithm 9, we use R to denote the intermediate vertex set of a p-cohesion containing the query vertex q. $G(R)$ is a p-cohesion when every vertex in R satisfy the definition of p-cohesion, i.e., $deg(v, G(R)) \geq \lceil p \times deg(v, G) \rceil$. Set P is the candidate set that the combining of P and R may be a p-cohesion. By X, we denote the vertices that have been processed. p is the parameter for p-cohesion and c is the minimum size of all p-cohesions we found. Within each recursive call, the algorithm considers vertices in P in turn. If there are no such vertices, it backtracks. For each vertex v chosen from P, it makes a recursive call in which v is added to R and X respectively, and in which P are the added with $N(v) - R - X$, which finds and reports all p-cohesions containing vertex v. Then, it moves v from R and P to exclude it from consideration in future p-cohesions and continues with the next vertex in P.

In Algorithm 10, we invoke Algorithm 9 with $R = \{q\}$, $P = N(q)$, $X = \{q\}$ and $c = |V(S)|$, where S is the input graph. It is clearly that, the p-cohesion returned by Algorithm 9 is the minimum p-cohesion containing query vertex

q. Note that, with the help of Algorithm 9, we can also enumerate all the p-cohesions of a graph.

Algorithm 10 ExactPC(S, p, q)

Input: S : a graph, p : a real number in (0,1), q : a query vertex
Output:$MinC_p$: the minimum p-cohesion containing q

```
$MinC_p \leftarrow \emptyset$, $c = |V(S)|$, $R \leftarrow \emptyset$, $P \leftarrow \emptyset$, $X \leftarrow \emptyset$
$MinC_p \leftarrow$ SubPCExact($R \cup \{q\}$,$P \cup N(q)$, $X$, $p$, $c$)
return $MinC_p$
```

As shown in the experiment part, the number of minimal p-cohesions is enormous, and given the fact that the MPCS is NP-hard, and does not admit a PTAS, unless P = NP, we design heuristic algorithms to fast retrieve a small minimal p-cohesion.

Global Search Algorithm

As previously mentioned, the whole graph or a connected component is a p-cohesion. By removing a vertex from a p-cohesion, it may cause the `collapse` of the p-cohesion, resulting in a smaller p-cohesion or an empty set. In Algorithm 11 we show the removal of a vertex may lead to the shrink of the p-cohesion.

Let u be a vertex that should be removed from a graph S. After removing u and the edges incident to u, if there is a vertex v in S violating the p-cohesion constraint (Line 2), we remove v and its incident edges from S at Line 3. The algorithm returns a smaller p-cohesion or an empty set.

Time Complexity. If a vertex u is deleted at Line 1 or 3, only the neighbors of u may violate the definition of p-cohesion (Line 2). Thus, each vertex is visited once for deletion and each edge is visited once for degree update and vertex marking. The time complexity of Algorithm 11 is $\mathcal{O}(m + n)$.

Algorithm 11 Collapse(S, p, u)

Input: S : a graph, p : a real number in (0,1), u : a vertex
Output:C_p : a smaller p-cohesion or an empty set

```
1: $S \leftarrow S \setminus \{u \cup E(u)\}$
2: while $\exists v \in V(S)$ with $deg(v,S) < \lceil p \times deg(v,G) \rceil$ do
3:     $S \leftarrow S \setminus \{v \cup E(v)\}$
4: return $S$
```

Space Complexity. The subgraph S, and the neighbor set take $\mathcal{O}(m+n)$ space respectively. The degree set, to-delete set and vertex index take $\mathcal{O}(n)$ respectively. The space complexity of Algorithm 11 is $\mathcal{O}(m+n)$.

Example 11. *Figure 1.3 shows a graph with the label of the smallest number of neighbors required for every vertex in a p-cohesion, when $p = 0.6$. If the input graph S is induced by $\{u_1, \ldots, u_{13}\}$, and $u = u_{12}$, in Algorithm 11, the deletion of u leads to the removal of u_{11}, u_{13} and u_{10} (Lines 2-3) according to the label (p-cohesion constraint). The returned S is induced by $\{u_1, \ldots, u_9\}$. In Algorithm 11, if the input graph S is induced by $\{u_1, u_2, u_3, u_4, u_5\}$, and $u = u_4$, the deletion of u leads to the removal of u_5, u_2, u_3 and u_1, such that the returned S is an empty set.*

Based on the above `collapse` procedure, we propose a global search algorithm to compute a minimal p-cohesion containing q in a top-down manner. Let S be the connected component containing the query vertex q, which is a p-cohesion, we iteratively remove a vertex from S without violating the p-cohesion constraint until no such vertex exists.

Algorithm 12 shows the details of the global search algorithm. Let S be a copy of the connected component in G which contains q (Line 1). At Line 2, T ensures each vertex in S is visited once only. Let S' be a copy of S (Line 4). We select an unvisited vertex u with the largest degree in S (Line 3) and compute

Algorithm 12 GlobalSearch(G, p, q)

Input: G : a graph, p : a real number in (0,1), q : a vertex
Output:C_{min} : a p-cohesion containing q

```
1: $S \leftarrow$ the connected component containing $q$ in $G$
2: $T \leftarrow \emptyset$; $T \leftarrow T \cup \{q\}$
3: while $\exists u \in V(S) \setminus T$ do
4:     $S' \leftarrow S$; $T \leftarrow T \cup \{u\}$
5:     $S \leftarrow$ Collapse($S, p, u$)
6:     if $S = \emptyset$ or $q \notin S$ then $S \leftarrow S'$
7: return $S$
```

the p-cohesion on S after deleting u by invoking Algorithm 11 (Line 5). If the returned subgraph is empty or q is deleted from S, we recover S (Line 6). When every vertex in current S is visited, S is a minimal p-cohesion. In this chapter, the vertex with the largest degree in current S will be chosen first at Line 3, because a large degree vertex may fast reduce the size of current p-cohesion (S).

Time Complexity. The visit of every vertex in $V(S)$ takes $\mathcal{O}(n)$ (Line 3). Algorithm 11 (Line 5) and the recover of S (Line 6) take $\mathcal{O}(m+n)$ for one iteration respectively. Thus, the time complexity of Algorithm 12 is $\mathcal{O}(n(m+n))$.

Space Complexity. The subgraph S, S' and the neighbor set take $\mathcal{O}(m+n)$ space respectively. The set T and $deg(\cdot)$ take $\mathcal{O}(n)$ space respectively. The space complexity of Algorithm 12 is $\mathcal{O}(m+n)$.

Example 12. *Figure 1.3 shows a graph with the label of the smallest number of neighbors required for every vertex in a p-cohesion, when p* $= 0.6$. *If* $q = u_1$, *Algorithm 12 may firstly choose* u_{10} *at Line 3, and then delete* u_{11}, u_{12} *and* u_{13} *by the collapse procedure (Algorithm 11). At the next loop, it deletes* u_4 *which leads to the deletion of all the vertices according to the constraint of p-cohesion, i.e.,* $S = \emptyset$. *Thus, we recover* S, *and try another unvisited vertex in* S *until no more vertices can be deleted. Finally, the vertices in* $\cup_{1 \leq i \leq 5} u_i$ *induce a minimal*

Algorithm 13 LocalSearch(G, p, q)

Input: G : a graph, p : a real number in (0,1), q : a vertex
Output:C_{min} : a p-cohesion containing q

```
1: $D \leftarrow \{q\}$; $T \leftarrow \emptyset$
2: $f(u) \leftarrow$ compute score for every vertex in $G$
3: while $\exists v \in D \setminus T$ do
4:     $T \leftarrow T \cup \{v\}$; $b \leftarrow |N(v,G) \cap D|$
5:     $P \leftarrow \{max(\lceil p \times deg(v,G) \rceil - b, 0)$ vertices in $N(v,G) \setminus D$ with the largest $f(u)\}$
6:     Update score $f(u)$;    $D \leftarrow D \cup P$
7: $S \leftarrow$ GlobalSearch$(G(D), p, q)$
8: return $S$
```

p-cohesion containing q.

Local Search Algorithm

Due to the giant component phenomenon [41], the connected component containing the query may occupy a large part of the graph. In such cases, Algorithm 12 is inefficient on large graphs. To improve algorithm efficiency, we can compute the minimal p-cohesion on a reduced S. In a bottom-up manner, we repeatedly expand the vertices starting from the query vertex to form a p-cohesion subgraph which may be much smaller than the initial connected component S. Algorithm 13 shows the local search procedure from q on a graph G. The set D records the to-expand vertices and set T ensures that each chosen vertex is expanded only once (Lines 1-4). When we expand a vertex v, b is the number of v's neighbors in D (Line 4). We add $max(\lceil p \times deg(v,G) \rceil - b, 0)$ neighbors of v to D, such that v can stay in the resulting p-cohesion (Line 5).

In Algorithm 13, at Line 3, the chosen vertex v is the vertex with the largest degree in D at that time. At Line 2 we compute a score $f(u)$ for every vertex. At Line 5, we choose the vertices in $N(v,G)$ with the largest score $f(u)$ in G-$G(D)$

since the existence of such vertices can get a good trade-off between the gain and penalty effect of adding one vertex to D. All these chosen vertices are added to D for further expansion (Line 6). After the expansion of every vertex in D, the vertices in D can induce a p-cohesion subgraph. By invoking Algorithm 12 (Line 7), we can get the minimal p-cohesion containing the query vertex q.

<u>*Score Function.*</u> A straightforward score definition for a vertex $u \notin D$ is the gain effect of adding u to D, denoted by $f^+(u)$, specifically, $f^+(u)$ records the number of u's neighbor v in D with $deg(v, G(D)) < \lceil p \times deg(v, G) \rceil$:

$$f^+(u) = |\{v | deg(v, G(D)) < \lceil p \times deg(v, G) \rceil, v \in N(u, G(D))\}| \tag{5.5}$$

Intuitively, $f^+(u)$ denotes the inclusion of u into D could contribute to increasing the degrees of some vertices in D s.t. they are closer to satisfy the p-cohesion constraint.

Besides, there is also a penalty effect of adding u to D, since u may need extra neighbors outside of D to make it have at least $\lceil p \times deg(u, G) \rceil$ neighbors in D. We denote the penalty by $f^-(u)$:

$$f^-(u) = max\{0, \lceil p \times deg(u, G) \rceil - |N(u, G(D))|\} \tag{5.6}$$

By considering both the gain and penalty effect, we define a score for a vertex u to determine which neighbor should be selected in Line 5 with the trade-off between its gain and penalty. The ultimate score of a vertex u is defined as:

$$f(u) = f^+(u) - f^-(u) \tag{5.7}$$

As for Algorithm 13, in Line 5, for an expanding vertex v, we choose its neighbors $u \notin D$ with the largest scores $f(u)$.

Example 13. *Figure 1.3 shows a graph with the label of the minimum number of neighbors required for every vertex in a p-cohesion, when* $p = 0.6$*. If* $u = u_1$*, Algorithm 13 may firstly add* u_2*,* u_3 *and* u_4 *to* D*, s.t.,* u_1 *satisfies the threshold for existing in a p-cohesion. Then* u_2 *and* u_3 *are expanded with no vertex pushed into* D*. Then* u_4 *is expanded, which adds* u_5 *to* D*, because* $f(u_5) = f^+(u_5) - f^-(u_5) = 1 - 0 = 1$ *is larger than* $f(u_6) = 1 - 2 = -1$ *and* $f(u_7) = 1 - 2 = -1$*. When all the vertices in* D *have been expanded, the algorithm returns the induced subgraph by* $\cup_{1 \leq i \leq 5} u_i$*.*

Time Complexity. Let $\hat{n}$ and $\hat{m}$ denote the number of vertices and edges of $G(D)$, respectively. In Line 3 of Algorithm 13, the total number of visited vertices in D is $\hat{n}$. For each vertex in D, the value b can be retrieved by visiting its neighbors, which takes $\mathcal{O}(m+n)$ for one iteration. The updating of $f(\cdot)$ for the neighbors of a vertex v takes $\mathcal{O}(deg(v, G) * log(deg(v, G)))$. The retrieval of P takes $\mathcal{O}(m+n)$ for one iteration. The score computation at each iteration takes $\mathcal{O}(m+n)$ because $deg(\cdot, G(D))$ and $N(\cdot, G(D))$ can be maintained when each vertex is added to D by visiting the neighbors of the vertex, and each vertex is added to D at most once. At Line 7, Algorithm 12 takes $\mathcal{O}(\hat{n}(\hat{m}+\hat{n}))$. As shown in our experiments, usually $\hat{n} \ll n$ and $\hat{m} \ll m$. Thus, the time complexity of Algorithm 13 is $\mathcal{O}(\hat{n}(m+n))$.

Space Complexity. The sets D, T, P, $f(\cdot)$ and $deg(\cdot)$ take $\mathcal{O}(n)$ space respectively. G and $N(\cdot)$ take $\mathcal{O}(m+n)$ space respectively. The space complexity of Algorithm 13 is $\mathcal{O}(m+n)$.

Algorithm Correctness. In Algorithm 13, every vertex v in D is expanded once at Line 3, which ensures v has sufficient neighbors in the partial set D by adding enough neighbors of v into D (Lines 5-6). When every vertex v in D has been expanded, every v in D satisfies $deg(v, G(D)) >= \lceil p \times deg(v, G) \rceil$, i.e., the returned $G(D)$ is a p-cohesion containing u. Then in Line 7, we invoke Algorithm 12. Since the Algorithm 12 has been proved to be correct, Algorithm 13 is correct.

5.4 Diversified Enumeration

In this section, we study the diversified p-cohesion enumeration problem (DPCE): Enumerate a set of disjoint minimal p-cohesions.

5.4.1 Problem Analysis

Firstly, we show the number of minimal p-cohesion for a graph can be exponential by the following theorem.

Theorem 8. *There exists a graph G that contains an exponential number of minimal p-cohesions, for every fixed $p \in (0, 1)$.*

Proof. We prove the theorem based on the exponential number of maximal cliques in a graph G. Suppose G is empty initially, we add a vertex set $O = \cup_{1 \le i \le n} v_i$ to G where $n = 2x$ and $x \in N^+$. For every vertex $v_i \in O$, we connect v_i to every other vertex in O except the opposite vertex v_j where $|j - i| = \frac{n}{2}$, i.e., the degree of every vertex $v \in O$ is $deg(v, G) = n - 2$.

For every vertex $v \in O$, we add y extra vertices and connect each of them to v:

$$y = \begin{cases} \lceil (n-2) \cdot (\frac{1}{2p} - 1) \rceil, & 0 < p < \frac{1}{2} \\ \lceil (n-2) \cdot (\frac{1}{2(1-p)} - 1) \rceil, & \frac{1}{2} \le p < 1 \end{cases} \tag{5.8}$$

The construction is completed. We have $|V(G)| = n + n \cdot y$. For every vertex

$v \in O$, we have $deg(v, G) = (n-2)$ $+y$:

$$deg(v, G) = \begin{cases} \lceil \frac{(n-2)}{2p} \rceil, & 0 < p < \frac{1}{2} \\ \lceil \frac{(n-2)}{2(1-p)} \rceil, & \frac{1}{2} \leq p < 1 \end{cases} \tag{5.9}$$

For both cases of p, we have $p \cdot deg(v, G) \geq \frac{n-2}{2}$. So for each vertex in O to be in a minimal p-cohesion S, at least $\frac{n-2}{2}$ of its neighbors are also in S. Note that, for a maximal clique of $G(O)$, the degree of every vertex in the clique is $\frac{n-2}{2}$. If we count each minimal p-cohesion in which each vertex contained in O has exactly $\frac{n-2}{2}$ neighbors in O, the number of the minimal p-cohesions is at least the number of maximal cliques in $G(O)$ which is $2^{n/2}$. Thus, the number of minimal p-cohesions in G is exponential. □

According to Theorem 8, the number of minimal p-cohesions may be overwhelming to users. Moreover, the p-cohesions discovered may heavily overlap with each other. Thus we are interested in tackling the diversified minimal p-cohesion enumeration problem (DPCE) for graph G: we prefer to find a set of disjoint minimal p-cohesions for G.

5.4.2 Pivot based Local Search (PLS)

In this section, we efficiently find a set of disjoint minimal p-cohesions based on the algorithms for MPCS.

Baseline Algorithm

A straightforward method to find a set of disjoint p-cohesions for a graph is to repeatedly find a minimal p-cohesion and remove it. Thus, we propose an algorithm in a top-down manner: For a graph G, starting with a connected component S, we can find a minimal p-cohesion by Algorithm 12: "C_p = *GlobalSearch(S, p, $\emptyset$)*". We remove C_p and the vertices violating the p-cohesion constraint after the removal of C_p. Repeatedly when all vertices are removed from graph G, we can get a set of disjoint minimal p-cohesions.

A Pivot based Local Search Algorithm

Finding one minimal p-cohesion in a top-down manner without a pivot is time-costly. Motivated by the MPCS, we can improve the efficiency of the `BaseLine Algorithm` by adding a pivot u and finding a minimal p-cohesion containing u by Algorithm 13. The details are shown in Algorithm 14.

At Line 1, we record the degree of every vertex in G. The set T is used to ensure every vertex is checked exactly once. Then we compute a minimal p-cohesion on each connected component S of current graph G from Line 2. A minimal p-cohesion C_p can be computed by Algorithm 13 with subgraph S, threshold p and a vertex $u \in S$. We delete C_p from S and record C_p in C (Line 6). We delete the vertices violating the fraction threshold of p-cohesion in S to further reduce S (Lines 7-8). Algorithm 14 returns the set of minimal p-cohesions in C.

At Line 3, we select the vertex with the smallest degree in S because a small degree pivot can fast expand to a p-cohesion and keep the advantage of the pivot that prunes more vertices at the early stage. Besides, the chosen pivot allows us to compute the minimal p-cohesion on a reduced initial S at Line 5 of Algorithm 14, which can improve the efficiency significantly.

Time Complexity. The visit of the connected components in the graph takes $\mathcal{O}(n)$ at most (Line 2). Algorithm 13 takes $\mathcal{O}(n(m+n))$ for one iteration (Line 5). The update of S and C takes $\mathcal{O}(m+n)$ at most (Lines 6-8). Thus, the time complexity of Algorithm 14 is $\mathcal{O}(n^2(m+n))$ in the worst case.

Algorithm 14 PLS(G, p)

Input: G : a graph, p : a real number in $(0, 1)$
Output:C : a set of disjoint minimal p-cohesions

```
deg(v) ← deg(v, G) for every $v \in V(G)$; $T \leftarrow \emptyset$
while ∃a non-empty connected component $S \in G$ do
    while $\exists u \in S \setminus T$ do
        $T \leftarrow T \cup \{u\}$
        $C_p \leftarrow$ LocalSearch($S, p, u$)
        $S \leftarrow S\text{-}C_p$; $C \leftarrow C \cup \{C_p\}$
        while $\exists v \in V(S)$ with $deg(v, G) < \lceil p \times deg(v) \rceil$ do
            $S \leftarrow S \setminus \{v \cup E(v)\}$
return C
```

Space Complexity. Algorithm 13 takes $\mathcal{O}(m + n)$ space. The G, C_p and C take $\mathcal{O}(m + n)$ space. The $deg(\cdot)$ and T takes $\mathcal{O}(n)$ space. The space complexity of Algorithm 14 is $\mathcal{O}(m + n)$.

5.4.3 An Application on MinSeed

In this section, we study an application of the minimal p-cohesion subgraphs on the contagion model introduced by [82, 41] along with the p-cohesion. To promote the sale of a product B, the company may give incentives to some seed users, such as a discount or free product trial. These seed users are regarded as activated (i.e., influenced) for using B, which may influence (i.e., activate) their friends to use B. The influence will further cascade to the friends of the activated users. In the following, we formally introduce the cascading rule.

Problem Statement

In this section, we give the definition of cascade rule, the problem statement of the MinSeed problem, and a simple describe for the complexity of the MinSeed problem.

Definition 7. ***Cascading Rule.*** *Given a graph G, the set of activated vertices $A \in V(G)$ and a cascading threshold $r \in (0,1)$, we have (1) a vertex $u \in (V(G) \setminus A)$ is immediately activated iff there are at least $\lceil r \times deg(u,G) \rceil$ activated neighbors of u in G, i.e., $deg(u, G(A)) \geq \lceil r \times deg(u,G) \rceil$; and (2) a seed vertex u is always activated.*

Given a target user group (which induces a graph) to cascade, a company may wish to find the fewest seed users (initial adopters) such that all the target users are activated while the promotion expense is minimized. Thus, the MinSeed problem is defined.

MinSeed Problem. Given a target graph G with no activated vertices, and a cascading threshold $r \in (0,1)$, the MinSeed problem is to find a set of seed vertices D in $V(G)$, such that (1) all the vertices in $V(G)$ are activated by applying the *Cascading Rule* repeatedly, and (2) $|D|$ is minimized.

We prove that MinSeed is NP-hard by a reduction from Vertex Cover problem. Given a graph G, when r is large enough, e.g., $r = |V(G) - 1|/|V(G)|$, the activation of a vertex needs all its neighbors to be activated first. To activate all the vertices in G, each edge in G should be incident to at least one seed vertex. Thus, MinSeed with such a r is exactly Vertex Cover which is NP-hard.

Solutions for MinSeed

Heuristic Seed Selection. Algorithm 15 shows the basic framework to find an approximate solution for MinSeed. For every vertex v in G, we use $c(v)$ to record the number of activated neighbors (Line 1). Set D records the selected seeds and set A records the activated vertices except the seeds (Line 2). We select an inactivated vertex u as a seed and use L to record the activated vertices by seed u (Lines 3-4). We update the $c(\cdot)$ value for the inactivated neighbors of each vertex

Algorithm 15 SeedSelection(G, r)

Input: G : a target graph, r : a real number in $(0, 1)$
Output:D : the seed set

```
 1: $c(v) \leftarrow 0$ for every $v \in V(G)$
 2: $D \leftarrow \emptyset$; $A \leftarrow \emptyset$
 3: while $\exists u \in V(G) \setminus \{D \cup A\}$ do
 4:     $D \leftarrow D \cup \{u\}$; $L \leftarrow \{u\}$
 5:     for each $u \in L$ do
 6:         for each $v \in N(u, G) \setminus \{D \cup A\}$ do
 7:             $c(v) \leftarrow c(v) + 1$
 8:             if $c(v) \geq \lceil r \times deg(v, G) \rceil$ then
 9:                 $L \leftarrow L \cup \{v\}$; $A \leftarrow A \cup \{v\}$
10: return $D$
```

in L (Lines 6-7). An inactivated neighbor v is activated if $c(v) \geq \lceil r \times deg(v, G) \rceil$ (Lines 8-9). The algorithm returns D as the seed set.

Time Complexity. Algorithm 15 activates each vertex in G by exactly one time and updates the $c(\cdot)$ value of its neighbors. So the time complexity of Algorithm 15 is $\mathcal{O}(m + n)$ if the seed selection at Line 3 takes up to $\mathcal{O}(m + n)$.

Space Complexity. Sets $c(\cdot)$, D, A, L and $deg(\cdot)$ take $\mathcal{O}(n)$ space respectively. G and $N(\cdot)$ take $\mathcal{O}(m+n)$ space respectively. The space complexity of Algorithm 15 is $\mathcal{O}(m + n)$.

Algorithm Correctness. Every vertex in G is either pushed into D as a seed, or pushed into A as an activated vertex by the seeds. Since the seeds are regarded as activated, the D returned by Algorithm 15 is a feasible solution of MinSeed.

Selection Order. The vertex selection order in Line 3 decides the number of resulting seeds. Because the minimal p-cohesions prevent outside influence spread according to the Fortress property, the vertices in the p-cohesions are relatively isolated from the non-p-cohesion vertices. By giving certain priorities to fortress vertices, we may break through the barrier of influence spread from the non-p-cohesion vertices to the p-cohesion vertices. Thus, the number of seeds required may be reduced. For MinSeed, we explore the following seed selection orders.

IC-Deg Selection. We select a vertex u with the largest degree in G at Line 3, because such a u can increase the $c(\cdot)$ values by a great extent and such a u can relax the fortress property when it is in a minimal p-cohesion.

IC-Core Selection. We select a vertex u with the largest coreness in G at Line 3, because the coreness of a vertex means the importance/influence [69]. The coreness of a vertex u is the largest value of k_{max} such that $u \in k_{max}$-core. k-core is a subgraph that every vertex in this subgraph has at least k neighbors inside. Thus, such a u may increase the $c(\cdot)$ values.

IC-Truss Selection. Similar to the IC-Core Selection, we select a vertex u with the largest trussness in G at Line 3, because trussness of a vertex also means the importance of the vertex [104]. Here we define the trussness of a vertex u is the largest value of trussness of the edges incident to this vertex. The trussness of an edge e is the largest value of k_m such that $e \in k_m$-truss. The $k-$truss is a subgrpah where every edge e is contained in at least $k - 2$ triangles in the subgraph. For some insight, such a u may increase the $c(\cdot)$ values.

IC-BF Selection. We select a vertex u at Line 3 such that the size increase of A is the largest with the selection of u. Such a u is "best" in a greedy view. When there are ties, we choose the one with the largest degree in G, as in the degree based selection. We hope these "best" vertices can effectively break the boundaries of the fortresses.

IC-PC Selection. We find that the vertices with extremely large degrees have great influence power. So firstly we select $\alpha \times |V(G)|$ vertices as seeds which have the largest degrees in G. Then we retrieve a set C of minimal p-cohesions on G-$G(D \cup A)$ by the `BUTD` algorithm where $p = 1-r+\varepsilon$ and ε is an infinitesimal positive number. We compute a weight $\beta \times deg(u, G)$ for each vertex u in C. For a vertex v not in C, its weight is just $deg(v, G)$. Then we continue to select a vertex u as a seed which has the largest weight in G at Line 3. In this way, the fortress property may be relaxed by giving priorities to the vertices in the minimal p-cohesions.

Example 14. *Figure 1.3 shows a graph where $r = 0.5$ and all the users are inactivated. By IC-Deg, Algorithm 15 may select u_{10} and u_4 sequentially. By IC-BF, Algorithm 15 may select u_{10}, u_7 and u_5 sequentially. By IC-PC, when $\alpha = 0.01$ and $\beta = 2$, Algorithm 15 may select u_4 and u_5.*

5.5 Performance Studies

5.5.1 Experimental Setting

In this section, we give the experimental settings for the evaluation of our proposed algorithms and techniques.

Algorithms. As far as we know, there is no existing work investigating the p-cohesion computation. We implement and evaluate 6 algorithms for p-cohesion computation and 3 algorithms for MinSeed as shown in Table 5.2.

Datasets. 8 real-life graphs are deployed in our experiments. The original data of *Yelp* is from [114], *DBLP* is from [68] and the others are from [67]. In *DBLP*, each vertex represents an author and each edge between two authors represents the two authors have at least 5 co-authored papers. The other datasets have existing vertices and edges. For *Gowalla* and *Brightkite*, we remove the vertices without check-ins and their incident edges. We transfer directed edges to undirected edges. Table 5.3 shows the statistics of the datasets.

Settings. All programs are implemented in standard C++ and are compiled

Algorithm	**Description**
GlobalS	Finding a minimal p-cohesion containing a query vertex q by Algorithm 12
LocalS	Local search algorithm (Algorithm 13) equipped with Equation (5.5)
LocalS*	Local search algorithm (Algorithm 13) equipped with Equation (5.7)
BaseTD	The top-down heuristic algorithm to find disjoint minimal p-cohesions (5.4.2)
BaseTD+	BaseTD equipped with the pivot vertices
BUTD	The pivot based local search algorithm (Algorithm 14)
IC-Deg	Algorithm 15 equipped with the degree based vertex selection
IC-Core	Algorithm 15 equipped with the coreness based vertex selection
IC-Truss	Algorithm 15 equipped with the trussness based vertex selection
IC-BF	Algorithm 15 equipped with the best first vertex selection
IC-PC	Algorithm 15 equipped with the minimal p-cohesion based vertex selection

Table 5.2: Summary of Algorithms

with G++ in Linux. All experiments are performed on a machine with Intel Xeon 2.3GHz CPU and Redhat Linux system. The runtime of an algorithm is set to INF if it cannot finish in 1 hour.

5.5.2 Effectiveness

In this section, we extensively evaluate the resulting p-cohesions regarding the size and the fortress property. Different cohesive subgraph models are compared with the p-cohesion. The fortress property of p-cohesion is evaluated on different influence spread models. We also show the minimal p-cohesions are more promising than the non-minimal p-cohesions. Case studies are depicted to visually show the effectiveness of p-cohesion. We also report the seed numbers for MinSeed using the p-cohesion based selection order and other selection orders.

Exact Number of p-cohesions. With the help of Algorithm 9 and Algo-

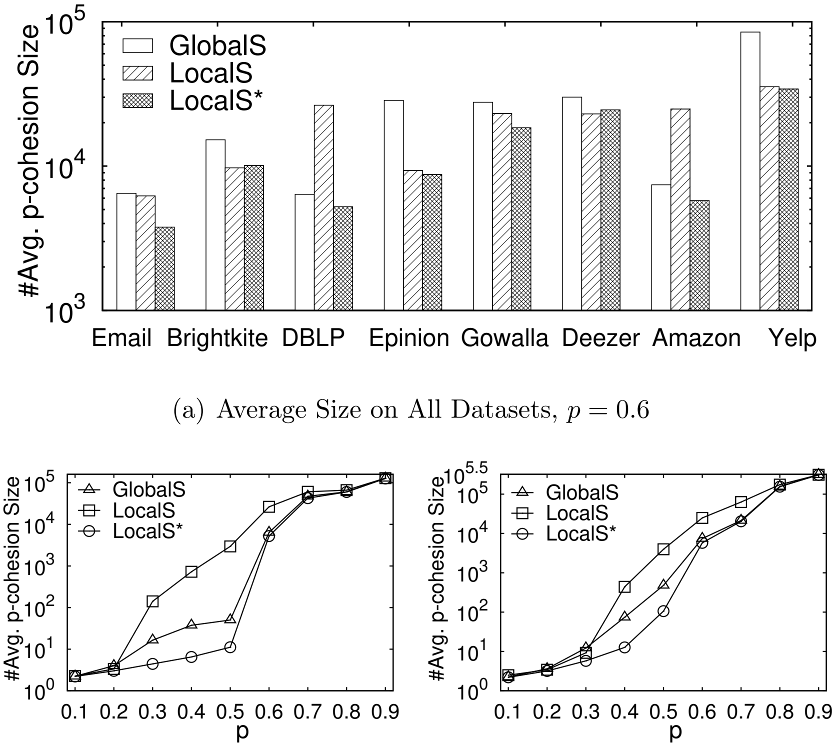
Dataset	**Vertices**	**Edges**	d_{avg}	d_{max}
Email	36,692	183,831	10	1,383
Brightkite	50,111	194,090	7.7	1,098
DBLP	210,840	363,299	3.4	159
Epinion	75,879	405,740	10.7	3,044
Gowalla	99,563	456,830	9.2	9,967
Deezer	54,573	498,202	18.3	420
Amazon	334,863	925,872	5.5	549
Yelp	249,440	1,781,908	14.3	3,812

Table 5.3: Statistics of Datasets

rithm 10, we can compute the number of minimal p-cohesions for a graph. Due to the high time and space complexity, we only compute the number of the p-cohesions on a small graph with $p = 0.6$. We extract a small graph G with 70 vertices from the *Yelp* dataset. Since the extracted graph with 80 vertices cannot finish in one week, we only compute the minimal p-cohesions of an extracted graph with 70 vertices.

We compute the exact number of minimal p-cohesions containing a query vertex with Algorithm 10, which can solve the MPCS problem. In order to do this, we randomly choose 10 vertices as the queries q and repeat "*ExactPC(G, p, q)*" for 10 rounds. The results show that the average number of minimal p-cohesions containing a query vertex is $19,778.1$, and the average smallest size is 9.2. We also compute the total number of minimal p-cohesions of the extracted graph with Algorithm 9, i.e. "*SubPCExact(*$\emptyset$*,* $V(G)$*,* $\emptyset$*,* p*,* $|V(G)|$*)*". The number of minimal p-cohesions of graph G with 70 vertices is $87,429$, which is much larger than the graph size.

Score Function Evaluation for MPCS. Figure 5.2 reports the average minimal p-cohesion subgraph size returned by GlobalS, LocalS, and LocalS* over 100 runs. One query vertex is selected randomly from all the vertices. Figure 5.2(a) shows the result on 8 datasets with $p = 0.6$. The score function based



(a) Average Size on All Datasets, $p = 0.6$

(b) DBLP

(c) Amazon

Figure 5.2: Average Size of Minimal p-Cohesions

algorithm `LocalS`* can significantly outperform `GlobalS` regarding average size, because the search space of `GlobalS` is larger. Some large degree vertices may be chosen in the execution of `GlobalS`, and they usually need more neighbors to stay in a p-cohesion. On most of the datasets, `LocalS`* can significantly outperform `LocalS`. For example, on *DBLP*, the number of vertices returned by `LocalS` is almost 5 times greater than the number of vertices returned by `LocalS`*. This implies that considering only the gain effect (Equation (5.5)) during the expansion procedure cannot guarantee a good performance. Figures 5.2(b)(c) report the results of three algorithms by varying the constraint p from 0.1 to 0.9 on *DBLP* (5.2(b)) and *Amazon* (5.2(c)) respectively. In both figures, the sizes of the minimal p-cohesion subgraphs returned by `GlobalS`, `LocalS`, and `LocalS`*

increase as p grows, since a large p inherently requires more vertices in a p-cohesion subgraph. When p is very large, the results of the three algorithms are similar, because nearly the whole graph is returned.

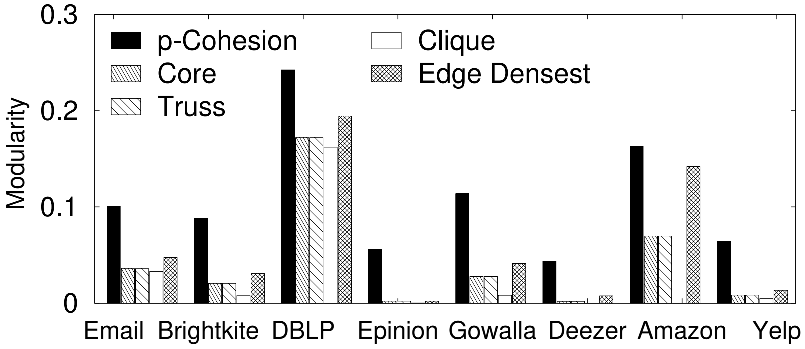


Figure 5.3: Comparing Modularity of Different Models, $p = 0.6$

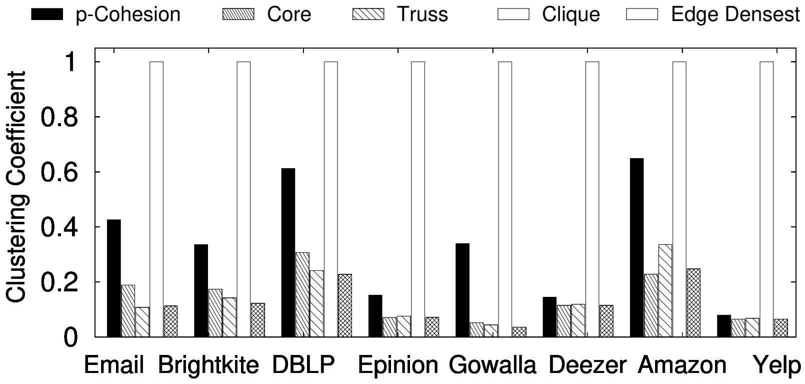


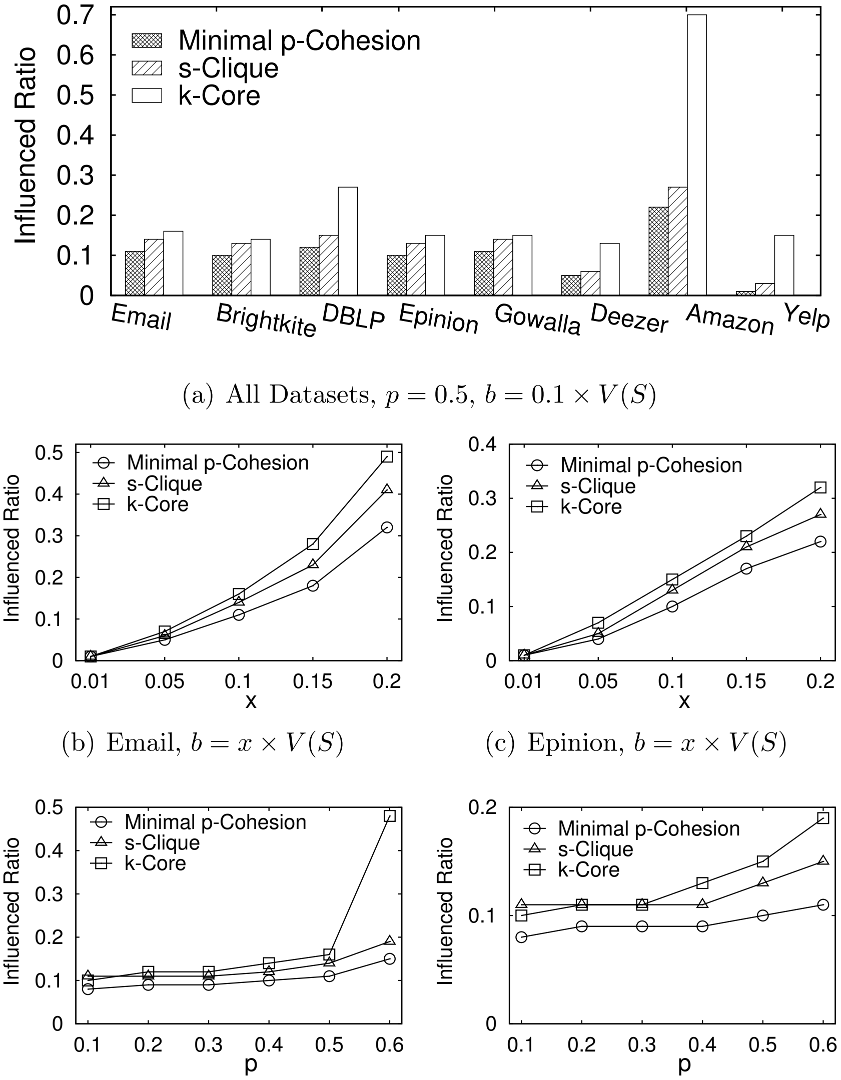
Figure 5.4: Comparing Clustering Coefficient of Different Models, $p = 0.6$

Different Cohesive Subgraph Models. For each query vertex v, we compute the minimal p-cohesion by `LocalS`*, the $\lceil p \times deg(v,G)\rceil$-core by [17], the ($\lceil p \times deg(v,G)\rceil$+1)-truss by [35], the maximal clique by [101], and the edge densest subgraph by [45] where every computed subgraph contains v. The query vertex is randomly selected from all the vertices in the graph. In Figure 5.3, we report the modularity scores [83] on the evaluated subgraph and the subgraph of the outside, for each of the above models, when $p = 0.6$. The scores are the average values from 100 independent tests. Consistent with the definition, the minimal p-cohesion shows better modularity scores because it holds both inner-cohesiveness and outer-sparseness. In Figure 5.4, we report the clustering coefficient values on the induced subgraphs of representative set of vertices for these models. Figure 5.4 shows the p-cohesion vertices possess significantly higher clustering coefficients on all datasets than other models except "Clique" model which is always 1. Considering the meaning of clustering coefficient, i.e., the high value of clustering coefficient of a group means the vertices inside tend to create tightly knit groups, which is corresponding with the definition of p-cohesion.

Evaluation of Fortress Property. Figure 5.5 reports the effectiveness of the minimal p-cohesions on the fortress property, compared with k-core [92] and s-clique [64]. The query vertex is randomly selected on the graph. The minimal p-cohesion is computed by `LocalS`*. For the s-clique containing the query vertex, we choose the size of the minimal p-cohesion as the parameter s. For k-core, the input of k is the largest value of k such that the k-core containing the query vertex is not empty.

We report the average influenced (i.e., activated) vertex ratio under the contagion model [41, 82] over 100 independent tests with $p = 0.5$ and influence ratio $r = 1.0 - p + 0.001$. For each influence test, we randomly choose $b = x \times |V(S)|$ seeds where S is the connected component containing the query vertex in the graph.

In Figure 5.5, `Minimal p-Cohesion` represents the average ratio of "the number of influenced vertices in the p-cohesion (denoted by S)" divided by "the number of all the vertices in S". It is similar for `k-Core` and `s-Clique`. The figure



(a) All Datasets, $p = 0.5$, $b = 0.1 \times V(S)$

(b) Email, $b = x \times V(S)$

(c) Epinion, $b = x \times V(S)$

(d) Email, $b = 0.1 \times V(S)$

(e) Epinion, $b = 0.1 \times V(S)$

Figure 5.5: Evaluating the Fortress Property

shows that the s-clique and k-core subgraphs have more difficulty in hindering the influence spread from outside, because the different definitions and they do not guarantee the outer-sparseness. The minimal p-cohesion shows a stronger fortress property than the others on all the settings.

Fortress Property on Different Influence Models. We also examine the

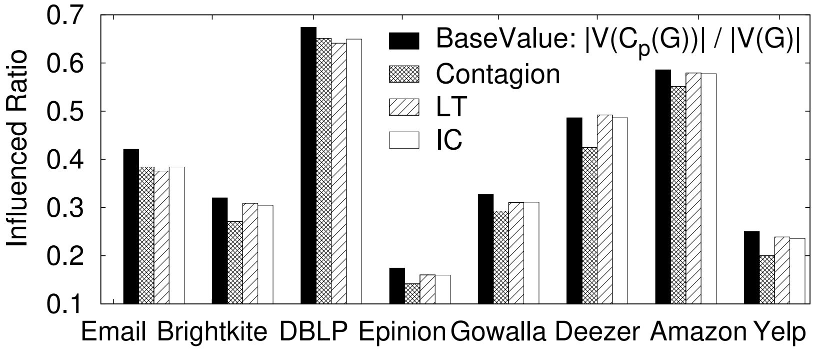


Figure 5.6: Fortress Property of Different Models

fortress property of p-cohesions under the independent cascade (IC) model and the linear threshold (LT) model [62], respectively. The target minimal p-cohesions are computed by `BUTD`. For each influence test, we randomly choose $b = 0.1 \times |V(G)|$ seeds. We follow the settings in [62] to compute the influence spread. During the cascade procedures of the LT and IC models, each edge (u, v) is split to two directed edges (u, v) and (v, u). The existing probability of each directed edge, i.e., (u, v) is set as $1/deg(v)$. For the LT model, we randomly distribute a threshold from 0 to 1 for each vertex v as the influence threshold. We generate 10 possible worlds on both the IC and LT models, to compute the influenced vertices and generate the average influence ratios.

In Figure 5.6, `Contagion`, `IC`, and `LT` represent the average influence ratio of "the number of influenced p-cohesion vertices" divided by "the number of all the influenced vertices" for contagion, IC model, and LT model, respectively. We use `BaseValue`, i.e., $|V(C_p(G))/|V(G)||$, to represent "the number of p-cohesion vertices" divided by "the number of all the vertices", as a base value for influence comparison. The setting is $p = 0.6$ and $r = 1 - p + 0.001$. `BaseValue` is basically larger than `Contagion`, `IC` and `LT`, among different settings, which implies that the vertices in a p-cohesion have a relatively smaller possibility to be influenced than the other vertices, under every evaluated influence model. The p-cohesion holds the fortress property on all the evaluated influence models, because a p-cohesion maintains a sparse connection to the outside s.t. the influence spread from outside is hard to enter the p-cohesion on these influence spread models.

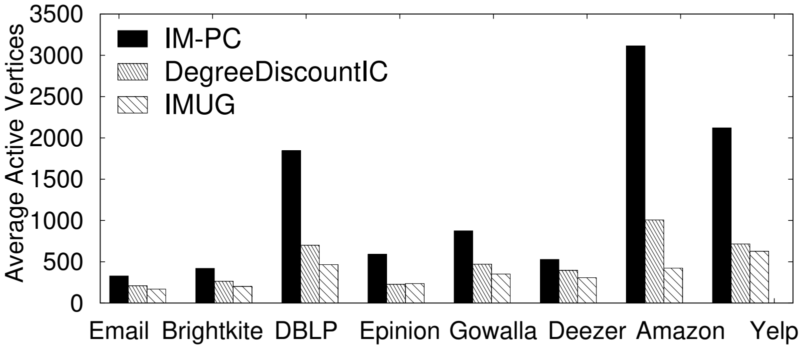


Figure 5.7: Influence Maximization of Different Methods

Evaluation with Influence Maximization Algorithms. To further verify the fortress property of our p-cohesion model, we test the spread of the influence under the influence maximization scenario. We use the IC model as the influence cascade model. The target minimal p-cohesions are computed by `BUTD` with $p = 0.6$. For the influence spread probability of the IC model we simply use $r = 1 - p = 0.4$. We follow the settings in [80] to compute the influence spread. We will run $R = 100$ rounds, and in each round we select $b = 50$ inactive vertices as seeds.

During the seed selection of our `IM-PC`, we compute the influence spread with the following steps: (1) choose an inactive vertex from all minimal p-cohesions with the largest degree as the seed; (2) compute the spread of the influence of this seed; (3) repeat step (1) and step (2) for $R * b$ times; (4) report the average number of active vertices. The `IMUG` [80] is for unknown graph, following the settings in [80], we prob $m = \lceil 0.001 * |V(G)| \rceil$ vertices and select b seeds for each round, s.t the expected number of active vertices in the Rth round is maximized [80]. Since the IMUG is for unknown graph, first of all, we probe the vertex with the largest degree, which will be selected as a seed. Because the vertex with the largest degree results in larger influence spread than other heuristics [62]. The DegreeDiscountIC [30] is based on degree discount. The general idea is as follows. Let v be a neighbor of vertex u. If u has been selected as a seed, then when considering selecting v as a new seed based on its degree, we should not count the edge (u, v) towards its degree.

In Figure 5.7, we report the average number of active vertices for different seed selection methods. IM-PC is larger than DegreeDiscountIC and IMUG, which implies that the vertices in a p-cohesion have a relatively smaller possibility to be influenced than other vertices. Since the p-cohesion holds the fortress property, seed from p-cohesions can break the entry barriers of the p-cohesions under IC model. While, the DegreeDiscountIC and IMUG do not consider this property, thus the influence spread is hard to enter the p-cohesions, s.t the influenced vertices of these two methods are smaller than our IM-PC method.

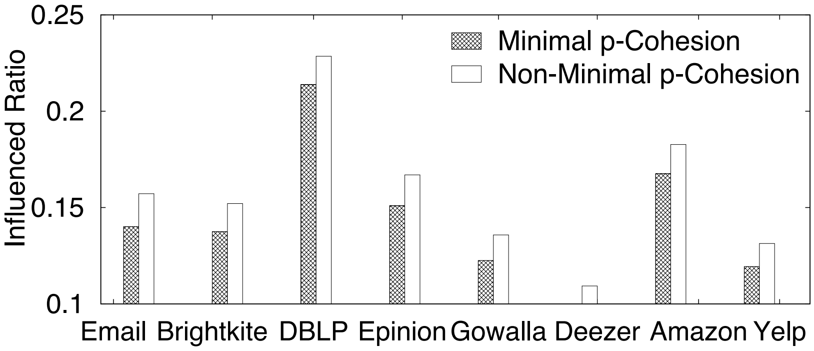


Figure 5.8: Minimal vs Non-Minimal p-Cohesions

Evaluation of Minimal and Non-Minimal p-Cohesions. Figure 5.8 evaluates the effectiveness of "minimal" constraint on p-cohesion, by comparing the fortress property of minimal p-cohesions and non-minimal p-cohesions. Each time for a random query vertex in G, we compute a minimal p-cohesion by `LocalS*` and a corresponding non-minimal p-cohesion by the expansion procedure of `LocalS*`. We conduct 100 influence tests under the contagion model. In each test, we randomly choose $b = 0.1 \times |V(S)|$ seeds to compute the influenced vertices, for 10 times, where S is the connected component containing the query vertex.

In Figure 5.8, `Minimal p-Cohesion` represents the average ratio of "the number of influenced vertices in the minimal p-cohesion (denoted by S)" divided by "the number of all the vertices in S". `Non-Minimal p-Cohesion` represents the average ratio of "the number of influenced vertices in the non-minimal p-cohesion (denoted by S')" divided by "the number of all the vertices in S'". The minimal p-cohesion shows stronger fortress property than the non-minimal p-cohesion, because the smaller diameter and size may benefit the defend of information cascades coming from the outside.

Case Study 1: MPCS on Email. In Figure 5.9, we depict a minimal p-cohesion S containing the query vertex "20317" found by `LocalS*` with $p = 0.6$ on *Email*. The p-cohesion contains all the grey vertices. In Figure 5.10, we find a s-clique containing the query vertex "20317" where $s = |V(S)|$. The s-clique contains all the grey vertices. We also compute the $\lceil p \times deg(v, G) \rceil$-core and the ($\lceil p \times deg(v, G) \rceil$+1)-truss containing vertex"20317", the sizes of the subgraphs found are "11,538" and "10,097" respectively. Since these two subgraphs are too large to show, we omit them in this chapter. We also depict all the 1-hop neighbors of one vertex in the p-cohesion or the s-clique, to show the outer connections to the subgraph. In Figure 5.9, we can see the vertices in S are sparsely connected to their outside neighbors. However, in Figure 5.10, vertex "24944" has a dense connection with its neighbors outside of the s-clique. The outside influence cascades may enter the s-clique through "24944".

In Figure 5.11 and Figure 5.12, we depict the minimal p-cohesion S found by `LocalS`* with $p = 0.6$ and a $|V(S)|$-clique containing the query vertex "251583" on *DBLP*. The p-cohesion S and $|V(S)|$ contain all the grey vertices respectively. We also compute the $\lceil p \times deg(v, G)\rceil$-core and the ($\lceil p \times deg(v, G)\rceil$+1)-truss containing vertex"251583", the sizes of the subgraphs found are "35,281" and "22,499" respectively. Since these two subgraphs are too large to show, we omit them in this chapter either. In the p-cohesion S or the $|V(S)|$-clique we also depict all the 1-hop neighbors of one vertex inside these subgraphs, to show the outer connections to the subgraph. The results on *DBLP* show the similar results as on *Email*.

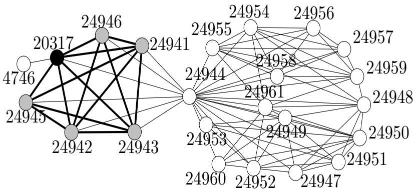


Figure 5.9: Minimal p-cohesion (*Email*)

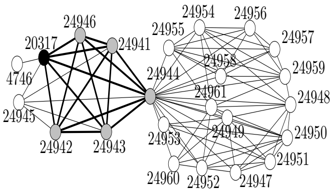


Figure 5.10: s-Clique(*Email*)

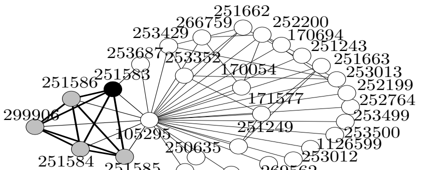


Figure 5.11: Minimal p-cohesion (*DBLP*)

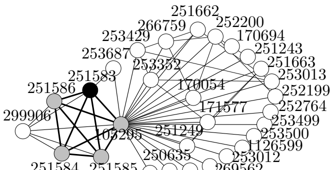


Figure 5.12: s-Clique(*DBLP*)

Case Study 2: DPCE on DBLP. Figure 5.13 depicts a part of `DBLP` with the minimal p-cohesions discovered by `BUTD` when $p = 0.8$. The p-cohesions are marked by the black edges and their incident vertices. We also show all the vertices which are within the 5-hop neighborhood of one vertex in the p-

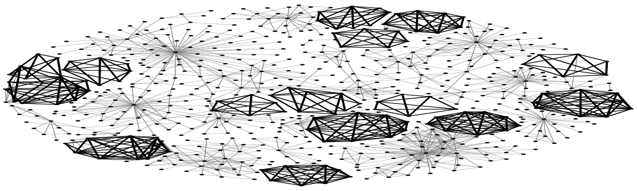


Figure 5.13: BUTD on *DBLP*, $p = 0.8$

cohesions. We can see the minimal p-cohesions have a loose connection to their outside neighbors.

	Email					Brightkite				
r	IC-Deg	IC-Core	IC-Truss	IC-BF	IC-PC	IC-Deg	IC-Core	IC-Truss	IC-BF	IC-PC
0.2	**1,129**	1,130	1,130	1,140	**1,129**	**457**	459	461	459	**457**
0.4	2,410	2397	2377	2,343	**2,083**	1,262	1291	1338	1,310	**1,253**
0.6	4,925	5107	5153	4,909	**2,156**	7,699	8280	8442	7,382	**1,312**
0.8	11,145	11,426	11,498	10,535	**2,292**	17,501	18,032	18,513	15,960	**1,264**

Table 5.4: Seed Numbers of *Email* and *Brightkite* with Different Methods

	Gowalla					Amazon				
r	IC-Deg	IC-Core	IC-Truss	IC-BF	IC-PC	IC-Deg	IC-Core	IC-Truss	IC-BF	IC-PC
0.2	**1,088**	**1,088**	1,091	1,102	**1,088**	1,126	1119	**1112**	1,125	1,126
0.4	2,722	2714	2700	2,902	**2,692**	21,122	21,445	21,487	-	**4,549**
0.6	16,131	17,123	17,082	15,487	**3,023**	76,215	77,633	78,845	-	**4,013**
0.8	39,264	40,663	41,358	-	**3,043**	149,177	150,616	155,377	-	**4,115**

Table 5.5: Seed Numbers of *Gowalla* and *Amazon* with Different Methods

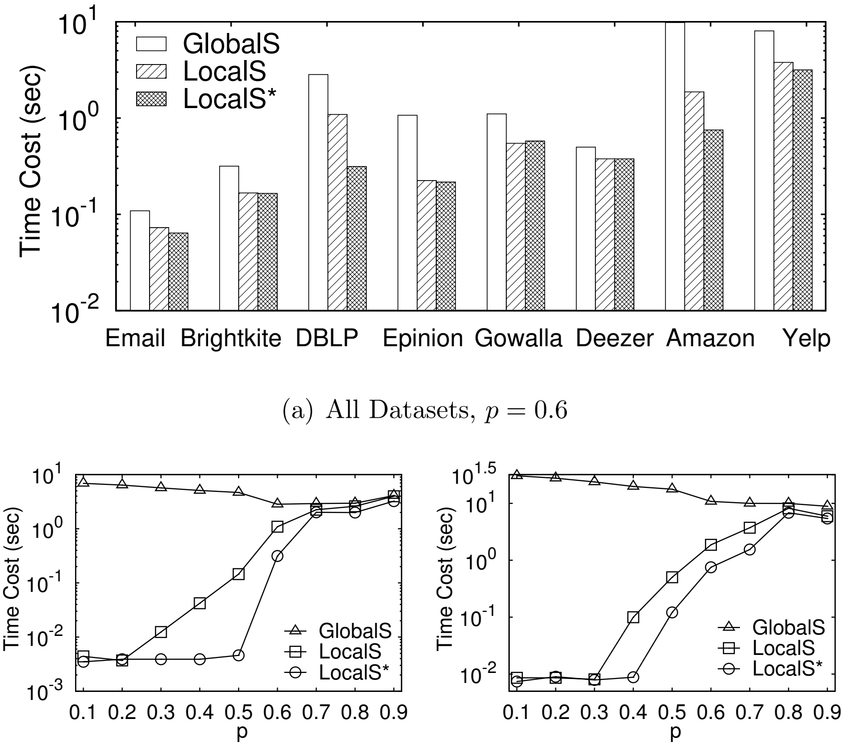
Evaluation of MinSeed Algorithms. In Table 5.4 and Table 5.5 we report the seed numbers returned by IC-Deg, IC-BF, IC-Core, IC-Truss, and IC-PC when r varies from 0.2 to 0.8. We report the seed numbers on four datasets, i.e., *Email*, *Brightkite*, *Gowalla*, and *Amazon*. According to Property 1, it is difficult for the p-cohesions to be influenced from the outside vertices. Our IC-PC gives selection priority to the vertices in minimal p-cohesions such that the resulting seed numbers can be reduced. We set ε as 0.001 and $p = 1 - r + \varepsilon$. According to statistical observation, we set $\alpha = 0.01$ and $\beta = 2$. Table 5.4 and Table 5.5

show that the IC-PC significantly selects less seeds than the other algorithms, which helps to reduce the cost of cascading the network. Some results of IC-BF are not reported because the computation cannot finish within 1 week. All methods of IC-Deg, IC-Core, IC-Truss, and IC-PC are efficient, because IC-PC just additionally conducts BUTD on a reduced graph and the BUTD is efficient. We observe that the seed number is much smaller than the size of a dataset, e.g., the seed number of IC-PC on *Brightkite* (resp. *Gowalla*) is only 2.5% (resp. 2.7%) of its vertex number when $r = 0.4$.

5.5.3 Efficiency

In this section, we report the runtime of finding a minimal p-cohesion and finding the disjoint minimal p-cohesions.

Score Function Evaluation on MPCS. Here we report the average runtime of GlobalS, LocalS, and LocalS* to compute a minimal p-cohesion containing a query vertex q over 100 runs. One query vertex is randomly selected among all the vertices. Figure 5.14(a) reports the runtime on all the datasets when $p = 0.6$. We observe that the runtime on a dataset is strongly affected by its vertex number, since every vertex belongs to at least 1 minimal p-cohesion in the result. Figures 5.14(b) and 5.14(c) report the runtime on *DBLP* and *Amazon* with p varying from 0.1 to 0.9. The runtime of GlobalS drops slightly with the increase of p, because a larger p makes vertex deletion more efficient. LocalS and LocalS* perform faster when p is small, because the computation space is reduced for a small p. Note that LocalS and LocalS* expand the query vertex to a p-cohesion and then deletes the vertices from the p-cohesion to produce a minimal p-cohesion containing q. LocalS runs slower than LocalS*, because the former does not consider the penalty effect during the expansion procedure, and usually finds a larger p-cohesion than LocalS*. When $p = 0.9$, the runtime of



(a) All Datasets, $p = 0.6$

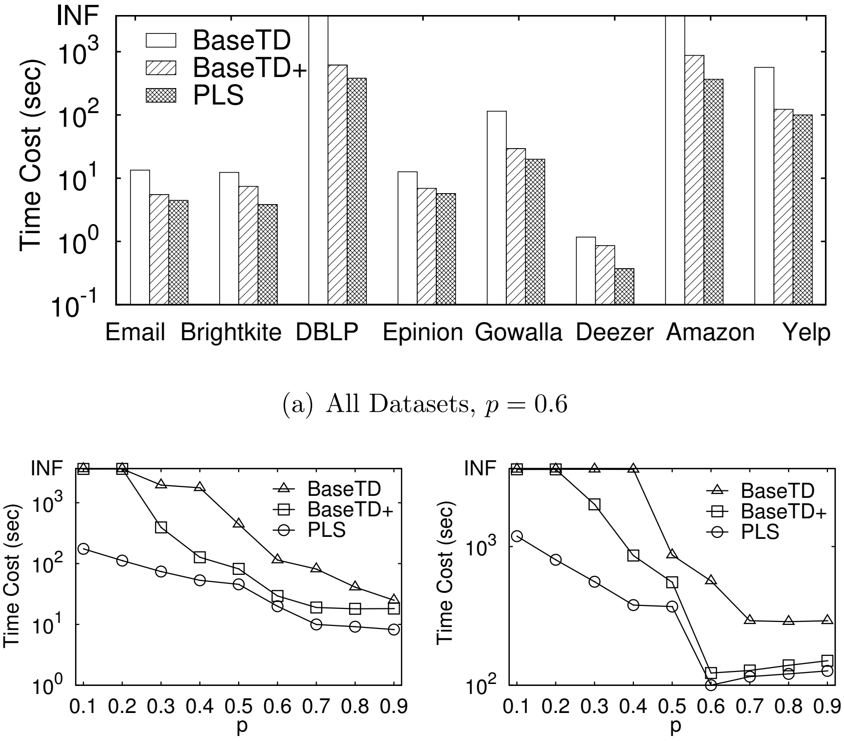
(b) DBLP

(c) Amazon

Figure 5.14: Finding a Minimal p-Cohesion

`LocalS`* is not larger than that of $p = 0.8$ on *Amazon*, because the p-cohesions have a similar size for both p values while the deletion procedure is faster for $p = 0.9$. In general, `LocalS`* significantly outperforms `GlobalS` on runtime.

Evaluating Algorithms for DPCE. Here we evaluate the performance of `BaseTD`, `BaseTD+` and `BUTD` to find disjoint minimal p-cohesions. Figure 5.15(a) reports the performance on all the datasets when $p = 0.6$. Figures 5.15(b) and 5.15(c) show that `BaseTD` and `BaseTD+` cannot finish the computation in 1 hour when p is small. Although the computation of one minimal p-cohesion is fast when p is small, the size of produced p-cohesion is quite small such that we have to compute much more minimal p-cohesions than that of large p values. This issue is relaxed when p is large enough such as $p = 0.6$. There are 4 factors which



(a) All Datasets, $p = 0.6$

(b) Gowalla

(c) Yelp

Figure 5.15: Finding Disjoint Minimal p-Cohesions

influence the trends of runtime: (a) the size of every expanded p-cohesion, (b) the size of every resulting minimal p-cohesion, (c) the number of resulting minimal p-cohesions, and (d) the deletion procedure from an expanded p-cohesion to a minimal p-cohesion. When the p value increases, usually (a) increases, (b) increases, (c) decreases and (d) speeds up, which, in total, constitutes the trends of runtime for different p. In general, BUTD is significantly faster than the other algorithms.

5.6 Chapter Summary

In this chapter, we study two representative problems on the fortress-like p-cohesion subgraphs: minimum p-cohesion search and diversified p-cohesion enumeration. We analyze the complexity of the problems and prove that finding a minimum p-cohesion is NP-hard and the minimal p-cohesion enumeration is intractable. From theory to practice, we propose efficient algorithms to find a minimal p-cohesion for a query vertex and a set of disjoint minimal p-cohesions. For a feasible solution of MinSeed, we employ the discovered minimal p-cohesions to reduce the seed number required for cascading the whole network. Comprehensive experiments show that our algorithms are efficient, the minimal p-cohesions hold the fortress property, and the algorithms help solve the MinSeed problem.

Chapter 6

EPILOGUE

In this thesis, we mainly study three representative cohesive subgraph models: k-truss, k-core, and p-cohesion in big graphs, and all of them have a large number of applications. We prove that the collapsed k-truss problem is NP-hard and inapproximate. After that, we propose an efficient algorithm collapsed k-truss, which can reduce the computation time. In addition to the k-truss model, we also study the minimum k-core search problem based on the k-core model, which has been proved to be NP-hard. Even though some existing algorithms can be used to solve this problem, they do not have any guarantee for the result subgraphs. Given this fact, we propose a progressive algorithm *PSA*, which can find the minimum k-core subgraph with a certain guarantee. Finally, we study a fortress-like cohesive subgraph model, p-cohesion. We are the first to give the complexity and propose efficient and effective algorithms to the best of our knowledge. We conduct extensive performance evaluations on several large real-world datasets for all three cohesive subgraph related problems to show our techniques and algorithms' efficiency and effectiveness.

Based on the research works in this thesis, several future research works can be conducting. Such as, the size-constraint k-core search problem can be figured out based on our progressive framework. While, our upper and lower bounds may not be suitable for the size-constraint problem, new techniques need to be considered. In addition, for the p-cohesion model, the decomposition related work can be an interested research topic.

BIBLIOGRAPHY

[1] A. Adiga and A. K. S. Vullikanti. How robust is the core of a network? In *ECML-PKDD*, pages 541–556, 2013.

[2] E. Akbas and P. Zhao. Truss-based community search: a truss-equivalence based indexing approach. *PVLDB*, 10(11):1298–1309, 2017.

[3] H. Aksu, M. Canim, Y. Chang, I. Korpeoglu, and Ö. Ulusoy. Distributed k-core view materializationand maintenance for large dynamic graphs. *TKDE*, 26(10):2439–2452, 2014.

[4] P. Alimonti and V. Kann. Hardness of approximating problems on cubic graphs. In *CIAC*, pages 288–298, 1997.

[5] M. Altaf-Ul-Amine, K. Nishikata, T. Korna, T. Miyasato, Y. Shinbo, M. Arifuzzaman, C. Wada, M. Maeda, T. Oshima, H. Mori, et al. Prediction of protein functions based on k-cores of protein-protein interaction networks and amino acid sequences. *Gen. Inf.*, 14:498–499, 2003.

[6] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. Large scale networks fingerprinting and visualization using the k-core decomposition. In *NIPS*, pages 41–50, 2005.

[7] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. K-core

decomposition of internet graphs: hierarchies, self-similarity and measurement biases. *NHM*, 3(2):371–393, 2008.

[8] O. Amini, D. Peleg, S. Pérennes, I. Sau, and S. Saurabh. On the approximability of some degree-constrained subgraph problems. *Discrete Applied Mathematics*, 160(12):1661–1679, 2012.

[9] R. Andersen and K. Chellapilla. Finding dense subgraphs with size bounds. In *WAW*, pages 25–37, 2009.

[10] S. Aral and D. Walker. Tie strength, embeddedness, and social influence: A large-scale networked experiment. *Management Science*, 60(6):1352–1370, 2014.

[11] A. Arora, S. Galhotra, and S. Ranu. Debunking the myths of influence maximization: An in-depth benchmarking study. In *SIGMOD*, pages 651–666, 2017.

[12] G. D. Bader and C. W. V. Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4:2, 2003.

[13] E. Bakshy, B. Karrer, and L. A. Adamic. Social influence and the diffusion of user-created content. In *ACM Conference on Electronic Commerce*, pages 325–334, 2009.

[14] E. Bakshy, I. Rosenn, C. Marlow, and L. Adamic. The role of social networks in information diffusion. In *WWW*, pages 519–528, 2012.

[15] S. Bandi and D. Thalmann. Path finding for human motion in virtual environments. *Comput. Geom.*, 15(1-3):103–127, 2000.

[16] N. Barbieri, F. Bonchi, E. Galimberti, and F. Gullo. Efficient and effective community search. *Data Min. Knowl. Discov.*, 29(5):1406–1433, 2015.

[17] V. Batagelj and M. Zaversnik. An o (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*, 2003.

[18] K. Bhawalkar, J. M. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma. Preventing unraveling in social networks: The anchored k-core problem. *SIAM J. Discrete Math.*, 29(3):1452–1475, 2015.

[19] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM*, 16(9):575–576, 1973.

[20] P. M. Cahusac, E. T. Rolls, Y. Miyashita, and H. Niki. Modification of the responses of hippocampal neurons in the monkey during the learning of a conditional spatial response task. *Hippocampus*, 3(1):29–42, 1993.

[21] J. Cannarella and J. A. Spechler. Epidemiological modeling of online social network dynamics. *CoRR*, abs/1401.4208, 2014.

[22] S. Carmi, S. Havlin, S. Kirkpatrick, Y. Shavitt, and E. Shir. A model of internet topology using k-shell decomposition. *PNAS*, 104(27):11150–11154, 2007.

[23] D. Centola and M. Macy. Complex contagions and the weakness of long ties. *American journal of Sociology*, 113(3):702–734, 2007.

[24] L. Chang, X. Lin, L. Qin, J. X. Yu, and W. Zhang. Index-based optimal algorithms for computing steiner components with maximum connectivity. In *SIGMOD*, pages 459–474, 2015.

[25] L. Chang and L. Qin. Cohesive subgraph computation over large sparse graphs. In *ICDE*, pages 2068–2071, 2019.

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

[27] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *APPROX*, pages 84–95, 2000.

[28] C. Chekuri, K. L. Clarkson, and S. Har-Peled. On the set multi-cover problem in geometric settings. pages 341–350, 2009.

[29] J. Chen, I. A. Kanj, and W. Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001.

[30] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In J. F. E. IV, F. Fogelman-Soulié, P. A. Flach, and M. J. Zaki, editors, *SIGKDD*, pages 199–208, 2009.

[31] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu. Efficient core decomposition in massive networks. In *ICDE*, pages 51–62, 2011.

[32] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu. Finding maximal cliques in massive networks by h*-graph. In *SIGMOD*, pages 447–458, 2010.

[33] R. Chitnis, F. V. Fomin, and P. A. Golovach. Parameterized complexity of the anchored k-core problem for directed graphs. *Inf. Comput.*, 247:11–22, 2016.

[34] R. H. Chitnis, F. V. Fomin, and P. A. Golovach. Preventing unraveling in social networks gets harder. In *AAAI*, 2013.

[35] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, page 16, 2008.

[36] R. Colbaugh and K. Glass. Emerging topic detection for business intelligence via predictive analysis of 'meme' dynamics. In *AAAI*, 2011.

[37] W. Cui, Y. Xiao, H. Wang, and W. Wang. Local search of communities in large graphs. In *SIGMOD*, pages 991–1002, 2014.

[38] M. Danisch, O. D. Balalau, and M. Sozio. Listing k-cliques in sparse real-world graphs. In *WWW*, pages 589–598, 2018.

[39] G. Dobson. Worst-case analysis of greedy heuristics for integer programming with nonnegative data. *Math. Oper. Res.*, 7(4):515–531, 1982.

[40] N. Eagle, A. S. Pentland, and D. Lazer. Inferring friendship network structure by using mobile phone data. *PNAS*, 106(36):15274–15278, 2009.

[41] D. A. Easley and J. M. Kleinberg. *Networks, Crowds, and Markets - Reasoning About a Highly Connected World.* Cambridge University Press, 2010.

[42] A. Epasto, S. Lattanzi, and M. Sozio. Efficient densest subgraph computation in evolving graphs. In *WWW*, pages 300–310, 2015.

[43] Y. Fang, R. Cheng, S. Luo, and J. Hu. Effective community search for large attributed graphs. *PVLDB*, 9(12):1233–1244, 2016.

[44] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, and X. Lin. A survey of community search over big graphs. *VLDBJ*, 29(1):353–392, 2020.

[45] Y. Fang, K. Yu, R. Cheng, L. V. S. Lakshmanan, and X. Lin. Efficient algorithms for densest subgraph discovery. *PVLDB*, 12(11):1719–1732, 2019.

[46] H. Fernau and J. A. Rodríguez-Velázquez. A survey on alliances and related parameters in graphs. *Electron. J. Graph Theory Appl.*, 2(1):70–86, 2014.

[47] G. Fricke, S. T. Hedetniemi, and D. P. Jacobs. Independence and irredundance in k-regular graphs. *Ars Comb.*, 49, 1998.

[48] D. Garcia, P. Mavrodiev, and F. Schweitzer. Social resilience in online communities: The autopsy of friendster. In *Proceedings of the first ACM conference on Online social networks*, pages 39–50, 2013.

[49] C. Giatsidis, F. Malliaros, D. M. Thilikos, and M. Vazirgiannis. Corecluster: A degeneracy based graph clustering framework. In *IAAI*, 2014.

[50] E. Gilbert and K. Karahalios. Predicting tie strength with social media. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 211–220. ACM, 2009.

[51] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica*, 39(4):321–347, 2004.

[52] M. S. Granovetter. The strength of weak ties. *American journal of sociology*, 78(6):1360–1380, 1973.

[53] J.-L. He, Y. Fu, and D.-B. Chen. A novel top-k strategy for influence maximization in complex networks with community structure. *PloS one*, 10(12):e0145283, 2015.

[54] J. Healy, J. Janssen, E. Milios, and W. Aiello. Characterization of graphs using degree cores. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 137–148, 2006.

[55] D. O. Hebb. The organization of behavior. a neuropsychological theory. 1949.

[56] D. S. Hochba. Approximation algorithms for np-hard problems. *SIGACT News*, 28(2):40–52, 1997.

[57] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322, 2014.

[58] X. Huang and L. V. S. Lakshmanan. Attribute-driven community search. *PVLDB*, 10(9):949–960, 2017.

[59] X. Huang, W. Lu, and L. V. Lakshmanan. Truss decomposition of probabilistic graphs: Semantics and algorithms. In *SIGMOD*, pages 77–90, 2016.

[60] X. Jiao, B. T. Sherman, D. W. Huang, R. M. Stephens, M. W. Baseler, H. C. Lane, and R. A. Lempicki. DAVID-WS: a stateful web service to facilitate gene/protein list analysis. *Bioinformatics*, 28(13):1805–1806, 2012.

[61] C. Jonathan. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, 2008.

[62] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *SIGKDD*, pages 137–146, 2003.

[63] M. Kitsak, L. K. Gallos, S. Havlin, F. Liljeros, L. Muchnik, H. E. Stanley, and H. A. Makse. Identification of influential spreaders in complex networks. *Nature physics*, 6(11):888–893, 2010.

[64] J. M. Kumpula, M. Kivelä, K. Kaski, and J. Saramäki. Sequential algorithm for fast clique percolation. *Physical Review E*, 78(2):026109, 2008.

[65] R. Laishram, A. E. Sariyüce, T. Eliassi-Rad, A. Pinar, and S. Soundarajan. Measuring and improving the core resilience of networks. In *WWW*, pages 609–618, 2018.

[66] J. Lehmann, M. Lalmas, E. Yom-Tov, and G. Dupret. Models of user engagement. In *UMAP*, pages 164–175, 2012.

[67] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, 2014.

[68] M. Ley. The dblp computer science bibliography: Evolution, research issues, perspectives. String processing and information retrieval. `http://dblp.uni-trier.de`, 2002.

[69] C. Li, F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. Efficient progressive minimum k-core search. *PVLDB*, 13(3):362–375, 2019.

[70] R. Li, Q. Dai, L. Qin, G. Wang, X. Xiao, J. X. Yu, and S. Qiao. Efficient signed clique search in signed networks. In *ICDE*, pages 245–256, 2018.

[71] R. Li, J. X. Yu, and R. Mao. Efficient core maintenance in large dynamic graphs. *TKDE*, 26(10):2453–2465, 2014.

[72] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou. Efficient (a,β)-core computation: an index-based approach. In *WWW*, pages 1130–1141, 2019.

[73] R. D. Luce and A. D. Perry. A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116, 1949.

[74] X. Luo, M. Andrews, Y. Song, and J. Aspara. Group-buying deal popularity. *Journal of Marketing*, 78(2):20–33, 2014.

[75] Y. Ma, Y. Yuan, F. Zhu, G. Wang, J. Xiao, and J. Wang. Who should be invited to my party: A size-constrained k-core problem in social networks. *J. Comput. Sci. Technol.*, 34(1):170–184, 2019.

[76] T. Maehara, H. Suzuki, and M. Ishihata. Exact computation of influence spread by binary decision diagrams. In *WWW*, pages 947–956, 2017.

[77] F. D. Malliaros and M. Vazirgiannis. To stay or not to stay: modeling engagement dynamics in social graphs. In *CIKM*, pages 469–478, 2013.

[78] D. W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, 1983.

[79] P. Meladianos, G. Nikolentzos, F. Rousseau, Y. Stavrakas, and M. Vazirgiannis. Degeneracy-based real-time sub-event detection in twitter stream. In *ICWSM*, pages 248–257, 2015.

[80] S. Mihara, S. Tsugawa, and H. Ohsaki. Influence maximization problem for unknown social networks. In J. Pei, F. Silvestri, and J. Tang, editors, *ASONAM*, pages 1539–1546, 2015.

[81] A. Mislove, M. Marcon, P. K. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *SIGCOMM*, pages 29–42, 2007.

[82] S. Morris. Contagion. *The Review of Economic Studies*, 67(1):57–78, 2000.

[83] M. E. Newman. Modularity and community structure in networks. *PNAS*, 103(23), 2006.

[84] M. P. O'Brien and B. D. Sullivan. Locally estimating core numbers. In *ICDM*, pages 460–469, 2014.

[85] J.-P. Onnela, J. Saramäki, J. Hyvönen, G. Szabó, D. Lazer, K. Kaski, J. Kertész, and A.-L. Barabási. Structure and tie strengths in mobile communication networks. *PNAS*, 104(18):7332–7336, 2007.

[86] E. T. Peterson and J. Carrabis. Measuring the immeasurable: Visitor engagement. *Web Analytics Demystified*, 14:16, 2008.

[87] R. Rotabi, K. Kamath, J. M. Kleinberg, and A. Sharma. Detecting strong ties using network motifs. In *WWW*, pages 983–992, 2017.

[88] M. Rubinov and O. Sporns. Complex network measures of brain connectivity: Uses and interpretations. *NeuroImage*, 52(3):1059–1069, 2010.

[89] A. E. Sariyüce, C. Seshadhri, A. Pinar, and Ü. V. Çatalyürek. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *WWW*, pages 927–937, 2015.

[90] J. Sauro and J. S. Dumas. Comparison of three one-question, post-task usability questionnaires. In *SIGCHI*, pages 1599–1608, 2009.

[91] H. Seba, S. Lagraa, and H. Kheddouci. Alliance-based clustering scheme for group key management in mobile ad hoc networks. *J. Supercomput.*, 61(3):481–501, 2012.

[92] S. B. Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.

[93] S. B. Seidman and B. L. Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology*, 6(1):139–154, 1978.

[94] K. Seki and M. Nakamura. The collapse of the friendster network started from the center of the core. In *ASONAM*, pages 477–484, 2016.

[95] K. Shin, T. Eliassi-Rad, and C. Faloutsos. Patterns and anomalies in k-cores of real-world graphs with applications. *Knowl. Inf. Syst.*, 54(3):677–710, 2018.

[96] S. Sintos and P. Tsaparas. Using strong triadic closure to characterize ties in social networks. In *SIGKDD*, pages 1466–1475, 2014.

[97] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *SIGKDD*, pages 939–948, 2010.

[98] O. Sporns. *Networks of the Brain*. MIT press, 2010.

[99] D. Szklarczyk, A. L. Gable, D. Lyon, A. Junge, S. Wyder, J. Huerta-Cepas, M. Simonovic, N. T. Doncheva, J. H. Morris, P. Bork, L. J. Jensen, and C. von Mering. STRING v11: protein-protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic Acids Research*, 47(Database-Issue):D607–D613, 2019.

[100] J. Tang, X. Tang, X. Xiao, and J. Yuan. Online processing algorithms for influence maximization. In *SIGMOD*, pages 991–1005, 2018.

[101] E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci.*, 363(1):28–42, 2006.

[102] J. Ugander, L. Backstrom, C. Marlow, and J. Kleinberg. Structural diversity in social contagion. *PNAS*, 109(16):5962–5966, 2012.

[103] D. Vogiatzis. Influence study on hyper-graphs. In *AAAI*, 2013.

[104] J. Wang and J. Cheng. Truss decomposition in massive networks. *PVLDB*, 5(9):812–823, 2012.

[105] M. Wang, C. Wang, J. X. Yu, and J. Zhang. Community detection in social networks: An in-depth benchmarking study with a procedure-oriented framework. *PVLDB*, 8(10):998–1009, 2015.

[106] X. Wang, R. Donaldson, C. Nell, P. Gorniak, M. Ester, and J. Bu. Recommending groups to users using user-group engagement and time-dependent matrix factorization. In *AAAI*, 2016.

[107] D. Wen, L. Qin, Y. Zhang, L. Chang, and X. Lin. Efficient structural graph clustering: An index-based approach. *PVLDB*, 11(3):243–255, 2017.

[108] D. Wen, L. Qin, Y. Zhang, X. Lin, and J. X. Yu. I/O efficient core graph decomposition at web scale. In *ICDE*, pages 133–144, 2016.

[109] S. Wirth, M. Yanike, L. M. Frank, A. C. Smith, E. N. Brown, and W. A. Suzuki. Single neurons in the monkey hippocampus and learning of new associations. *Science*, 300(5625):1578–1581, 2003.

[110] C. I. Wood and I. V. Hicks. The minimal k-core problem for modeling k-assemblies. *The Journal of Mathematical Neuroscience*, 5(1):14, 2015.

[111] S. Wu, A. D. Sarma, A. Fabrikant, S. Lattanzi, and A. Tomkins. Arrival and departure dynamics in social networks. In *WSDM*, pages 233–242, 2013.

[112] S. Wuchty and E. Almaas. Peeling the yeast protein network. *Proteomics*, 5(2):444–449, 2005.

[113] J. Yao, C. Lin, X. Xie, A. J. A. Wang, and C. Hung. Path planning for virtual human motion using improved a star algorithm. In *ITNG*, pages 1154–1158, 2010.

[114] Yelp. Yelp Dataset Challenge: Discover what insights lie hidden in our data. `https://www.yelp.com/dataset/challenge`, 2015.

[115] I. G. Yero and J. A. Rodríguez-Velázquez. Defensive alliances in graphs: a survey. *arXiv preprint arXiv:1308.2096*, 2013.

[116] H. P. Young. The dynamics of social innovation. *PNAS*, 108(Supplement 4):21285–21291, 2011.

[117] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang. Diversified top-k clique search. *VLDBJ*, 25(2):171–196, 2016.

[118] L. Yuan, L. Qin, W. Zhang, L. Chang, and J. Yang. Index-based densest clique percolation community search in networks. *TKDE*, 30(5):922–935, 2018.

[119] A. Zarezade, A. Khodadadi, M. Farajtabar, H. R. Rabiee, and H. Zha. Correlated cascades: Compete or cooperate. In *AAAI*, pages 238–244, 2017.

[120] F. Zhang, C. Li, Y. Zhang, L. Qin, and W. Zhang. Finding critical users in social communities: The collapsed core and truss problems. *TKDE*, pages 78–91, 2018.

[121] F. Zhang, L. Yuan, Y. Zhang, L. Qin, X. Lin, and A. Zhou. Discovering strong communities with user engagement and tie strength. In *DASFAA*, pages 425–441, 2018.

[122] F. Zhang, W. Zhang, Y. Zhang, L. Qin, and X. Lin. OLAK: an efficient algorithm to prevent unraveling in social networks. *PVLDB*, 10(6):649–660, 2017.

[123] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. Finding critical users for social network engagement: The collapsed k-core problem. In *AAAI*, pages 245–251, 2017.

[124] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. When engagement meets similarity: Efficient (k, r)-core computation on social networks. *PVLDB*, 10(10):998–1009, 2017.

[125] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. Efficiently reinforcing social networks over user engagement and tie strength. In *ICDE*, pages 557–568, 2018.

[126] H. Zhang, H. Zhao, W. Cai, J. Liu, and W. Zhou. Using the k-core decomposition to analyze the static structure of large-scale software systems. *The Journal of Supercomputing*, 53(2):352–369, 2010.

[127] P. Zhang, W. Chen, X. Sun, Y. Wang, and J. Zhang. Minimizing seed set selection with probabilistic coverage guarantee in a social network. In *SIGKDD*, pages 1306–1315, 2014.

[128] Y. Zhang, J. X. Yu, Y. Zhang, and L. Qin. A fast order-based approach for core maintenance. In *ICDE*, pages 337–348, 2017.

[129] F. Zhao and A. K. H. Tung. Large scale cohesive subgraphs discovery for social network visual analysis. In *PVLDB*, pages 85–96, 2012.

[130] Z. Zhou, F. Zhang, X. Lin, W. Zhang, and C. Chen. K-core maximization: An edge addition approach. In S. Kraus, editor, *IJCAI*, pages 4867–4873, 2019.