University of Bergen
Department of Informatics

A faster algorithm for computing c-closure

Author: Tuva Ulveseth Kvalsøren
Supervisor: Pål Grønås Drange

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

June, 2023

Abstract

Exploiting the structure of graphs is a well-known strategy for solving hard problems on complex graphs. In our studies we consider structural characteristics of social network graphs, in particular ways to exploit the high frequency of closed triangles due to the property of triadic closure.

In recent years this property has been captured in the notion of c-closure introduced by Fox et al. [SIAM Journal on Computing, 2020]. A graph is c-closed if any two nodes with at least c friends in common are also friends themselves. Multiple NP-complete problems exhibits FPT algorithms parameterized by c.

Computing the actual value for c can be done in $O(n^\omega)$ time, where ω is the matrix multiplication complexity. We provide an improved algorithm for computing the c-closure of a graph that runs in $O(n \cdot d^3)$ time for a majority of real-world networks and $O(n \cdot d^3 + n^2 \cdot d)$ in the worst case. The algorithm exploits another common property of social networks, a low degeneracy value d.

Acknowledgements

First of all I would like to thank my supervisor Pål for his guidance and expertise throughout my thesis. I would also like to thank all my friends at Jafu for a wonderful final year in Bergen, with long days and countless lunches and coffees.

Thank you to my friends and family for enriching my life with all your knowledge, experiences and love during the course of my studies.

Lastly, a special thank you to Nøstegaten and my dear roommates Kari and Andrea. I am grateful for the unwavering support and for creating an encouraging, understanding and safe environment at home.

Tuva Ulveseth Kvalsøren
Tuesday 13th June, 2023

Contents

Chapter 1

Introduction

The world of data and networks yields an intricate web of information, with every single record, node and connection offering valuable insights about the network they belong to. But with a vast amount of data that is constantly evolving and accumulating, we are faced with some critical challenges: How can we extract information and insights from networks in an efficient manner?

To achieve this we lean on a specialized field within computer science called parameterized complexity. In essence, it provides methods to solve complex problems fast by exploiting known patterns and properties of the data we are working with.

Take for instance social networks, which are the primary focus of our study. Here, we notice that two people with many mutual friends are likely to be introduced to each other. It also applies to meeting new people through common interests, work or school. New connections forming as a result of existing connections are far more likely than two randomly selected people in the network becoming friends. One can also validate the effects of this by simply looking at our own social circles. This principle is referred to as *triadic closure*.

Motivated by the concept of triadic closure, Fox et al [5] recently introduced the parameter c-closure rooted in the very intuitive tendencies of the dynamics in social networks described above. Put a bit more formally, they state that the c-closure of a graph is the largest amount of common connections two people can have, without being connected themselves, see the formal definition, Definition 3.

Due to the effects of triadic closure, social networks exhibit small c-closure values relative to in randomly generated graphs. As c-closure is a simple concept to grasp and not a hard parameter to calculate relative to some of the known hard problems in the world of algorithms, it has been utilized to design more efficient algorithms for solving hard problems on social network graphs by several researches following Fox et al.

As well as a discussion upon c as a parameter, we aim to tackle the complexity associated with computing the parameter itself. We apply the ideas behind exploiting c to design efficient algorithms for hard problems, and provide a new algorithm to compute c by exploiting another common property of social networks, a small degeneracy d. The degeneracy of a network is a measure of sparseness, and aims to quantify how densely knit groups of people are within the network. By utilizing this property, we obtain an algorithm that runs in linear time in the input size of the graph for 94% of the real-world data sets we considered and $O(n^2 \cdot d + n \cdot d^3)$ in the worst case. This is a significant improvement to the naive approach that in the worst case can take cubic time relative to the input size.

The effects of triadic closure also implies that there is an abundance of triangles within a social network, compared to a randomly generated graph. We make use of known methods to count triangles in graphs to give an altered algorithm of computing c-closure that has shown to perform very well compared to naive approaches.

In our results we have determined the c-closure of roughly 250 new data sets varying in size in addition to testing the performance of our proposed algorithms, thus gaining new insights into the properties of large social networks.

1.1 Preliminaries

In order to provide context and clarity we give some definitions and specifications regarding notions used throughout the thesis. We assume some fundamental knowledge in algorithms and computer science prior to our general discussion.

1.1.1 Graphs

The constantly growing amount of data generated within the world of social networks holds important insights into the social dynamics at play. However, exploiting these insights can become an impossible task without a method for structuring the data, often consisting of thousands or even millions of records. Representing the data available to us as *graphs* lets us transform what appears initially as unstructured pools of data to mathematical models. This enables us to make use of computational procedures to reveal the internal structures and information embedded within large data sets.

A graph $G = (V, E)$ is a data structure consisting of a set of vertices V, also called nodes, and a set of edges E. Each edge, denoted as a tuple (u, v), represents a certain relationship between two nodes u and v.

Unless stated otherwise, all graphs considered in this thesis are assumed to be *undirected* and *unweighted*. In an undirected graph, the relationship between two nodes is symmetric.

This implies that an edge (u, v) also yields an edge (v, u). We use a single edge between two nodes to signify their relationship. In an unweighted graph, every edge carries equal significance, and does not have an associated weight.

When working with graphs, we are interested in which nodes are connected to each other, and introduce the neighborhood of a node v. For a node v, the adjacent nodes make up its open neighborhood, denoted $N(v)$. We give the following formal definition: $N(v) = \{u \in G | (u, v) \in E, u \neq v\}$. That is, $N(v)$ is the set of all nodes that can be reached from v by traversing a single edge in G. In addition, we define the closed neighborhood of v, $N[v]$, as the union of the open neighborhood of v and v itself, hence $N[v] = N(v) \cup \{v\}$.

When implementing a graph model, we need a suitable data structure to keep track of which nodes are connected. The most commonly used for representing adjacency in a graph is an *adjacency matrix* or an *adjacency list*. The choice of adjacency representation depends on what is most efficient for the purpose of the graph or algorithm that utilizes it.

Substructures of graphs

A *subgraph* S of a graph G is defined as a set of nodes and edges within G such that $S \subseteq G$. Including the edges between the nodes in a subgraph yields substructures within the graph, also referred to as *induced subgraphs*. These provide important insights about a graph and its nodes in their surrounding context.

Our work primarily relies on examining and exploiting substructures frequently found in social networks. Among these are triangles. A triangle, denoted (u, v, z) is a complete subgraph of G composed of three nodes. In a complete subgraph the nodes are pairwise adjacent meaning that there is an edge between every pair of nodes in the graph. This is also referred to as a *clique*.

Another substructure essential to the analysis of density and triangle patterns in social networks are open triangles, commonly referred to as wedges throughout the thesis. A wedge, denoted (u, v, z), is an induced subgraph of G made up of three nodes but only including two edges (u, v) and (v, z). A wedge is also referred to as a path of three vertices, or an open triangle as it is equivalent to a triangle missing an edge. The relationship between wedges and triangles in social networks and its significance in the context of c-closure will be discussed in Section 2.3.

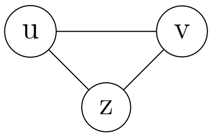

Figure 1.1: A triangle (u, v, z).

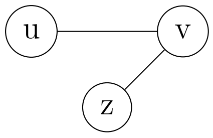

Figure 1.2: A wedge (u, v, z).

1.1.2 Computational complexity

One of the first concepts we come across when entering the field of computer science and algorithms, is classifying problems as hard or perhaps simple to compute. In order to have some agreed upon way to characterize a problem as hard, the field of computational complexity theory is introduced. The field aims to provide a hierarchy to classify problems based on scalability.

We introduce basic terminology used throughout the thesis when discussing the complexity of problems and how well both known and new algorithms perform.

A *problem instance* is a formalization of a computational problem and represents some specific input to a problem. On small problem instances most computable problems can be solved within a reasonable amount of time. Consequently, it can be difficult to distinguish between the complexity of two such problems. However, as the size of the problem instances grows, the time required to find a solution can evolve very differently. In other words, two problems that appear equally complex at a smaller scale may exhibit great differences in the time it takes to find a solution when the problem instances grow in size.

This concept can be represented visually by plotting the computational time of algorithms against thesis instance size, see Figure 1.3[1]. The instance size, commonly measured in number of nodes n or edges m, is plotted along the x-axis, while the computational time, represented as a function of the instance size $f(n)$, is plotted along the y-axis. By illustrating how the functions evolve when the instance sizes increase, the differences in *time complexity* of algorithms becomes evident. Another common measure of computational complexity is *space complexity*, which serves as a measure of the memory usage of an algorithm. Throughout this thesis space complexity is not considered to an extensive degree.

1.2 Roadmap of the thesis

Throughout the remaining parts of the thesis we introduce fundamentals within social network theory and characteristics of social networks in Section 2.1. We elaborate upon

[1]Figure 1.3 is an illustration from Stack Overflow cc-by-sa

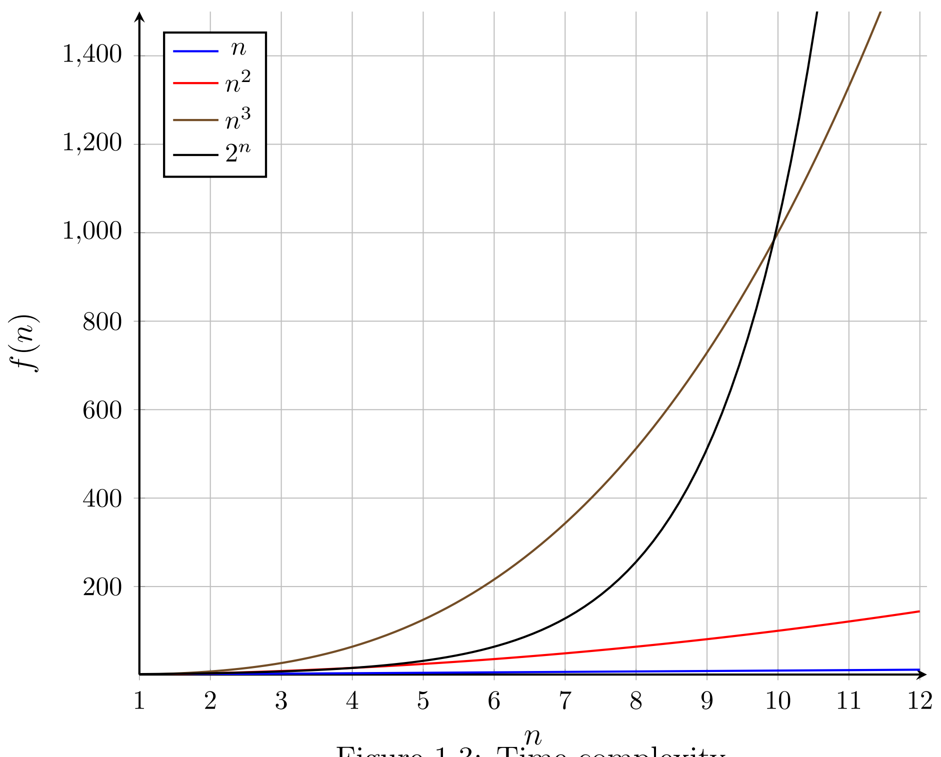

Figure 1.3: Time complexity

the topic of *triadic closure* and its impacts in the formation of triangles in social networks in Section 2.1.3 as well as the correlation between open triangles and the c-closure of a graph. In Section 2.2 we introduce an extended field of complexity theory, *Parameterized complexity*. We introduce the motivation and ideas behind exploiting known structures of graphs to obtain better algorithms, both for intractable problems in Section 2.2.1 and tractable problems in Section 2.2.2. We go in depth on the formal definition and results of exploiting c-closure in Section 2.3. Throughout the section we also encounter a discussion on the complexity of computing the c-closure itself and suggest multiple approaches to reduce this cost. In Section 2.4 we present the degeneracy value d of a graph and how it can contribute to the computation of the c-closure. The actual algorithms for both known and new methods to compute c-closure and d-degeneracy is presented in Section 3. The results of running our algorithms on hundreds of data-sets are presented in Section 3.3.

Finally, we conclude in Chapter 4.

Chapter 2

Background

2.1 Social Networks

In the field of computer science, social networks are often modeled as graphs with nodes representing individuals and edges representing a relationship between two individuals. What yields an edge can be any predefined parameter and can encompass a variety of social interactions, such as friendships, connections on social media, e-mail interactions, long-term relationships or just a brief acquaintance. The reasoning behind the evolution of edges is less relevant for evaluating and constructing the actual algorithms for social networks but an interesting topic when researching the theory behind social networks. Throughout our general discussion we will refer to two people being friends as equivalent to their corresponding nodes sharing an edge.

As mentioned in our introduction to graph theory in Section 1.1.1, our algorithms and results consider undirected and unweighted graphs, unless stated otherwise. Undirected graphs are common for social network graphs that models a mutual relationship, such as friendships. Directed social network graphs, on the other hand, are intended to capture one-way relationships between nodes. Social media platforms that utilizes follower relationships, such as Instagram and Twitter, serves as good examples of social networks that would call for a directed graph model. Weighted graphs are useful for modelling social networks that incorporate multiple types of relationships that can be measured along a weight spectrum. For instance, one might want to assign a larger weight to a long-term relationship compared to a brief acquaintance if this serves the purpose of the model. However, the introduction of weights adds additional parameters to consider and may increase the complexity of the problem without providing significant insight into the actual problem we are looking to solve. In our studies we ignore weighted relationships as it does not serve any good purpose for the issues we are considering.

Modelling large and complex networks, such as social networks as graphs with restricted parameters, allows us to utilize computer science to provide insight in the mechanisms behind social network evolution. It also facilitates solving mathematical problems on concepts that from a human perspective is anything but mathematical, such as interpersonal relationships and social constructs.

2.1.1 Central measures in social networks

When we aim to describe characteristics of social networks, certain measures and metrics are essential. They allow for a thorough understanding of and reasoning behind the structures within a social network. We provide a brief explanation of central terms in social network theory.

1. *Degree*: The degree is a measure that applies to a single node and is simply the number of connecting nodes a node has. The degree of a node u, denoted $\deg(u)$, is the size of its neighborhood, $|N(u)|$.

2. *Density*: The density is a measure that applies to the overall structure of the graph. It tells us how interconnected a network is, essentially revealing how tightly the nodes are linked to each other. There are several methods to measure graph density, but one commonly applied method involves the ratio of the actual edges in the graph, denoted as $|E(V)|$, to the maximum possible number of edges, indicated by $\binom{n}{2}$.

3. *Diameter:* The diameter of a graph is the longest shortest path between any two nodes. Put simply, its the longest one will have to travel to reach any node from any other in a graph.

4. *Centrality*: Centrality contains several different measures designed to identify the most significant nodes within a social network. These include degree centrality, closeness centrality and betweenness centrality, among others. Each of the measures embodies different interpretations of what constitutes an influential node. Degree centrality categorizes nodes with the highest degrees, i.e. the most connections, as the most important. This measure emphasizes the pure connectivity of a node within the network. On the other hand, closeness centrality focuses on the proximity of a node to the other nodes within the network. This way it emphasizes the most centrally located nodes relative to all other nodes which is a beneficial measure when desiring efficient information flow or rapid interaction within the network. Betweenness centrality captures the frequency of which a node occurs on the shortest path between other nodes. Nodes with high betweenness centrality are likely to serve as critical bridges within the network and can play a crucial part in controlling the

flow of information or interaction. Each of the measures captures different aspects of an individuals potential role within a social network.

5. *Clustering Coefficient*: The clustering coefficient is a metric related to a single node and encapsulates how tightly embedded a node is within its own neighborhood. Essentially, it quantifies how interconnected a nodes' neighbors are with each other. The clustering coefficient is calculated by simply counting the number of pairs of neighbors of a given node u that are also mutually connected. The clustering coefficient plays an important role in understanding the local structure surrounding individual nodes, particularly as it directly relates to the formation of triangles and wedges in a social network. While this concept will be further elaborated upon in Section 2.1.3, it is important to recognize its role in analyzing and understanding the underlying structures within social networks.

6. *Transitivity*: The transitivity of a network is also a measure of density more specifically the ratio between triangles and wedges in a graph. As with the clustering coefficient, the transitivity of a graph can be directly related to its c-closure.

In addition to the measures and properties introduced above there are several other terms often used in the context of social networks. However, these are among the most central and serves a purpose for understanding how we can benefit from the structures of social networks.

2.1.2 Characteristics of social networks

When studying complexity theory and algorithms on a theoretical level the primary focus is on the algorithms themselves, with their complexity expressed as a function of the number of nodes within a graph. The internal structure of a graph is not usually a central part of the discussion. Yet, when it comes to applying our theoretical knowledge to real-world data, the actual performance of an algorithm will often depend on the structure of the graph it operates on. Acquiring in-depth knowledge about the data we are working with can be an important tool to further develop algorithms for graphs of certain characteristics.

Social networks, for instance, are among the complex graph types that tend to exhibit a predictable structure and thus making them prime candidates for developing fine-tuned algorithms. Unlike a randomly generated large graph, the structure of a social network is a product of human behaviour over time. This offers us valuable information about the structure of the graph, allowing us to identify specific characteristics that can be exploited to construct algorithms that performs better given these characteristics. There has been done extensive research on common characteristics in social networks. Some of the most prominent ones are:

1. *Power law distribution:* Social networks have a tendency to conform to a power law distribution. This implies that while a vast majority of the nodes have a degree near the average, there exists a few nodes with an extraordinarily high degree. The phenomenon, also referred to as a *heavy tail distribution*, results in a distribution where the majority of the nodes occurs far away from the peak of the distribution. This arises naturally from popularity dynamics. Consider celebrities as an example. In a social network, a celebrity (represented as a node) will have an atypically high number of connections compared to the average individual. This implies that, for the most part, the average degree of the nodes in a social network does not differ too much. This is an observation that can be useful in the complexity analysis and the design of algorithms applied to social networks and more efficiently handle real-world data.

2. *Communities*: Within social network theory, we refer to communities as tightly-knit groups with a significantly higher connectivity internally than externally. Community structures tend to naturally form within social networks as a result of a networks evolution over time. Identifying these communities can provide valuable insights into understanding the social dynamics at play. Furthermore, the concept of communities serves as a crucial role in several centrality measures, including but not limited to, the closeness centrality measure previously mentioned.

3. *Small-world phenomenon:* The small world phenomenon refers to the surprisingly low diameter found in social networks. As a consequence, regardless of how the social network scales in size, it is usually possible to reach any given node within a remarkable small number of steps. The concept is also commonly called *six degrees of separation*, a term popularized by a play written by Guare in 1990[3]. The idea was that any two individuals in the world can be connected through a chain of no more than six common acquaintances. There is also done extensive research on the topic of small world phenomenon, but our interest is mainly lies in what this implies for the structure of the graph, specifically its low diameter.

4. *Triadic Closure:* Triadic closure is the tendency of triangles closing in social networks over time due to common connections. The principle is quite intuitive: it reflects the likelihood that an individual is introduced to new connections through existing ones, such as new friends through a common friend, or new interests through the interests among people in your community. The effects of triadic closure close wedges

formed within the network, forming closed triangles. In upcoming sections we will dive deeper into the implications of triadic closure, particularly how it affects the formation of triangles and wedges, and consequently c-closure of a graph.

These characteristics illustrates the special structures found in social networks and are exactly what we aim to exploit in the design of our algorithms.

2.1.3 Triadic closure: The relationship between wedges and triangles in social networks

As the effect of triadic closure, introduced in the previous Section 2.1.2, plays out in a social network, the amount of closed triangles in the graph increase. Counting and enumerating these substructures play an important part in both deciding the transitivity of a network and central nodes in a network [7].

Triangle counting and enumeration is not considered an inherently difficult task from an algorithmic perspective, following the definition of hard problems given in Section 1.1.2 regarding computational complexity. As early as in 1978, Itai and Rodeh presented an algorithm for listing triangles in terms of the number of edges, m, with a runtime of $O(m^{\frac{3}{2}})$ [8]. However, it has shown to be an expensive procedure for large graphs. Social network graphs can consist of millions of nodes with dynamically evolving relationships(edges), which makes triangle enumeration costly. Multiple of the metrics discussed in Section 2.1.1 concerning the density of a social network is directly related to the number of open and closed triangles in the graph. Two of these are the clustering coefficient of a node and transitivity the entire graph. The local clustering coefficient of a node u in a graph G tell us something about how embedded a node is in its surrounding community. Nodes with a high clustering coefficient tend to be centers of cliques, both in the social and in the algorithmic sense of the word. Formally it is defined as the fraction of neighbors of u that are neighbors themselves. Hasan et al. [7] give the following formal definition:

$$C(u) = \frac{\{|(v,w) : (v,w) \in E \wedge v, w \in N(u)\}}{\frac{1}{2}|N(u)| \cdot (|N(u)| - 1)}$$

Two adjacent neighbors of u would yield a triangle, making the clustering coefficient of a node also a measure of how many triangles it contributes to. The metric can also be used to detect global clusters in general graphs and community detection in social networks.

The second metric mentioned is the transitivity. Whilst the clustering coefficient relates to a single node, the transitivity of a graph is defined as the ratio between open and closed triangles in the entire graph. The task of computing the transitivity is thus identical to computing the number of open and closed triangles in a graph. Hasan et al.

[7] give the following definition of the transitivity ratio of a graph G denoted by $\gamma(G)$

$$\gamma(G) = \frac{|triangles|}{|triples|} = \frac{|triangles|}{|opentriangles| + |triangles|}$$

Only the number of open and closed triangles are used in the computation, so computing the transitivity does not require the substructures to be enumerated just counted.

The understanding of clustering coefficients and the transitivity of a graph is valuable when arguing the evolution of c-closure as a parameter, as these measures relate to the same substructures in a graph, triangles and wedges.

2.2 Parameterized complexity

Complexity theory in the field of computer science serves as a foundational pillar in the field of theoretical computer science, with the P versus NP question as one of its central challenges. At its core, it seeks to classify computational problems based on their difficulty by categorizing them into complexity classes such as P, problems solvable in polynomial time, and NP, problems whose solutions can be verified in polynomial time but not necessarily solved in polynomial time, see Section 1.1.2. However, there may be different factors influencing the difficulty of problems within the same complexity class. *Parameterized complexity* aims to offer an approach to solving and understanding computational problems by considering how different parameters contribute to the complexity. In classical complexity, the most common parameter to measure complexity by is the input size of the problem n and m, i.e. nodes and edges, whereas parameterized complexity tries to acknowledge that real world problems exhibits varying levels of difficulty depending on the characteristics of the problem or data. For social networks some of these characteristics are introduced in Section 2.1.2 and can serve as a topic of designing *parameterized algorithms* for these types of graphs. A parameterized algorithm is simply an algorithm that makes use of some known parameter [2]. In our work we discuss algorithms parameterized by respectively c-closure c and degeneracy d.

In Parameterized complexity an extended class hierarchy, like the hierarchy found in classical complexity theory, is introduced as the *W-hierarchy*[2]. This hierarchy aims to classify problems beyond the class of NP-complete problems. We do not go into detail regarding the hierarchy, but will provide some examples of how parameterizing problems by different parameters may yield different complexity classes in a later Section 2.2.3.

2.2.1 FPT

One of the key concepts within parameterized complexity is *Fixed Parameter Tractability*, *FPT* for short. Fixed parameter tractability is a complexity class that contains problems solvable within a function of some parameter k other than the input size n which is the traditional measure of complexity. This parameter can be any chosen parameter or combination of parameters often known to be smaller than n. The most common value to parameterize by is the solution size, as the optimal solution of NP-complete problems with a minimization objective is often small relative to the size of the problem instance.

When k is significantly smaller than n one can obtain efficient algorithms that may run in exponential time with respect to k, yet in polynomial time with respect to n. The formal definition is as follows:

Definition 1 ([2])**.** *For a parameterized problem $L \subseteq \Sigma^* \times N$ we say that it is FPT if there exists an algorithm A, a computable function $f : \mathbb{N} \to \mathbb{N}$, and a constant c such that A correctly decides whether an instance of L, $(x, k) \in L$ in time bounded by $f(k) \cdot |(x, k)|^c$.*

As different parameters may influence the complexity of a problem, the choice of parameter needs careful consideration when designing FPT algorithms. As established in "Parameterized Algorithms" [2], there are two primary criteria that should be satisfied to obtain a successful parameterization:

1. We want to select a parameter that is likely to be small for the specific problem instances in consideration. If the parameter we choose is large, the complexity might not differ much from its classical complexity measured by the size of the input.

2. Secondly, we want to design algorithms where the time consuming terms are restricted to the parameter of choice. Meanwhile, the growth induced by other parameters, such as input size n, should be kept as small as possible and within polynomial growth.

These outlined criteria are also the topic of the two primary strategies for optimizing FPT-algorithms. We aim to either minimize the *parametric dependence*, Item 1, or minimize the *polynomial factor*, Item 2.

Another central aspect of designing FPT-algorithms and parameterizing problems is *kernelization*. In the field of parameterized complexity, kernelization involves applying a polynomial-time preprocessing algorithm to an instance of a computationally hard problem. In "Parameterized Algorithms" [2] they give the following formal definition:

Definition 2. *A kernelization algorithm, or simply a kernel, for a parameterized problem Q is an algorithm $\mathcal{A}$ that, given a problem instance (I, k) of Q, works in polynomial time* *and returns an equivalent instance* (I', k')*. Moreover, we require that size* $\mathcal{A}(k) \leq g(k)$ *for some computable function* $g : \mathbb{N} \to \mathbb{N}$

The aim is to reduce the size of the problem instance, thereby making it feasible to apply an exact algorithm to solve the reduced instance. As it is proven that a decidable problem admits a kernel if and only if its fixed-parameter tractability, kernelization can serve as another way of defining FPT [2].

The motivation behind designing FPT algorithms is directly applicable to solving problems for complex social networks, when we know that these networks often exhibit low values for certain parameters as the c-closure value c and degeneracy value d discussed in later Sections 2.3 and 2.4. By also expanding the principles of designing FPT algorithms beyond intractable problems, major contributions to bridging the gap between theoretical computation and real world application can be made. This topic will be further explored in the following Section 2.2.2.

2.2.2 FPT in P

Traditionally, FPT algorithms are designed for solving intractable problems, specifically those in the class of NP-hard problems. However, when solving problems known to be in P on large networks, by designing algorithms that exploits known small parameters of the network rather than solely considering the instance size, the cost can be significantly reduced. As we recall, this strategy is a fundamental aspect of designing FPT algorithms for intractable problems, as it enables the development of more efficient real-world applications. In *FPT-in-P*, instead of obtaining polynomial dependence on the input size, we aim towards getting a linear or closer to linear dependence on the input size. The FPT-in-P algorithms differ from the FPT algorithms designed for NP-complete problem in that the dependence on the parameter, $f(k)$, may be polynomial. Giannapoulou et al. [6] provide three desirable algorithmic properties that one should aim for when designing FPT-in-P algorithms, equivalent to the listed goals for traditional FPT-algorithms mentioned in the previous Section 2.2.1:

1. The running time should have a polynomial dependency on the parameter.
2. The running time should be as close to linear as possible for instances when the parameter value is constant.
3. The parameter value, or a good approximation thereof, should be computable efficiently.

Following the example from the introduction of the article by Giannapoulou above [6], assume there exists an $O(n^z)$-time algorithm for a problem with instance size n. Our aim is to exploit a smaller parameter k and provide an algorithm $f(k) \cdot n^{z'}$ such that $z' < z$ and $f(k)$ depends solely on our parameter k. In contrast to the class of Fixed Parameter Tractable (FPT)-problems where $f(k)$ may be exponential in k, for the class of *Polynomial Fixed Parameter Tractable*, P-FPT, problems, $f(k)$ is polynomial in k, i.e k^t for some fixed integer t.

2.2.3 Not in FPT: W[1]-hard problems

What about the problems that does not exhibit an FPT-algorithm parameterized by solution size k? As a sidenote to further emphasize the motivation behind the choice of parameter when designing FPT-algorithms we introduce some of the complexity classes in the W-hierarchy, $W[1]$ and $W[2]$. Some NP-complete problems, like the independent set and dominating set problem, does not have a known FPT-algorithm parameterized by solution size k. For these two problems we say that they are respectively *$W[1]$-hard and $W[2]$-hard with respect to the solution size k*. For our purposes, it is sufficient to say that a problem being $W[1]$-hard for a parameter k is equivalent to saying that it does not yield an FPT-algorithm. This also hold for the higher levels of the hierarchy, such as $W[2]$-hard, as they are subsets of each other. However, this does not necessarily imply that these problems does not yield FPT-algorithms parameterized by other parameters.

2.3 C-closure

Motivated by triadic closure in social networks, the notion of c-closure was recently introduced by Fox et al. [5]. The c-closure of a graph is a concept that captures the likelihood of two individuals having many mutual friends being friends themselves. They state that if two people in a c-closed graph are not friends, they share less than c common friends. Equivalently, if two nodes in a c-closed graph have more than or exactly c common neighbors, they are also adjacent. More formally Fox give the following definition of c-closure:

Definition 3. *[5] A graph G is c-closed if $|N(u) \cap N(v)| < c$ for each pair of nonadjacent vertices u and v. The c-closure of G is the smallest integer c such that G is c-closed.*

We say that a graph is c-closed for its lowest number of c-closure.

As a social network matures, the effects of triadic closure tends to reduce the c-closure value. This pattern is evident in our results presented in our table of results 3.3 for a significant portion of our datasets. In their work, Fox et al.[5] utilize the c-closure of a graph to obtain algorithms parameterized by c, with their main result being a new $O(p(n, c) + 3^{\frac{c}{3}} n^2$ upper bound for generating all maximal cliques in any c-closed graph, with $p(n, c)$ denoting the complexity of listing all wedges. [5]. Several studies following the initial introduction of c-closure by Fox et al. have developed FPT-algorithms and claimed new upper bounds for known NP-complete problems for complex graphs by parameterizing by c [1, 9, 12, 13, 10, 11, 14, 15].

Observe that c is not a good measure for bipartite graphs, since by definition any two vertices with a common neighbor do not share an edge.

2.3.1 c as a parameter

When designing FPT-algorithms, the parameter chosen needs careful consideration. One can argue that c as a parameter is quite fragile as it only requires a single pair of "bad" nodes to increase the value of c significantly, as the parameter is defined for its highest c. However, c-closure effectively captures a simple effect of human behavior and is an intuitive parameter to grasp, across various fields of study. Its computation is also relatively simple, and our results in Table 3.3 reveal that c is typically small for many of the social networks under consideration. Other types of complex networks, not necessarily social, can exhibit low c-closure due to various reasons, and are also included among the data-sets we have used for our analysis.

The parameter also holds other properties that is worth to note when utilizing it for parameterization. Contrary to parameterizing by the solution size k, the c-value is derived directly from the structure of the graph itself and is not an externally chosen parameter. When a graph is modified, either due to its natural evolution or during computational processes such as kernelization algorithms, briefly introduced in Section 2.2.1, the c-closure value of the graph can change.

To illustrate this, imagine a kernelization algorithm that reduces a problem instance (I, c) to a smaller instance (I', c'). This yields an interesting discussion regarding the c-closure of the reduced problem instance. Since the removal of nodes from a graph could alter the c-closure value, this aspect of parameterizing by c-closure needs to be considered when designing FPT algorithms and kernelization algorithms that make use of c. If the value for c decreases, one might be able to give an even tighter bound. But a change in c may also call for a new computation of c for each iteration of an algorithm. This is one of the motivations behind desiring an optimized algorithm for computing c-closure in the following Section 2.3.2.

2.3.2 Computing c-closure

A majority of the results following the introduction of c-closure does not include a discussion regarding the computation of the c-closure itself. As it is a parameter computed in polynomial time it makes sense that it is not not a priority in arguing the complexity of algorithms solving NP-complete problems. However, it is evident that optimizing the time it takes to compute c can remarkably reduce the overall costs associated with the algorithms. Recall that the c-closure is a property a graph itself holds and not a chosen parameter. Because of this it may be necessary to compute the c-closure multiple times during the execution of an algorithm, following from the discussion in Section 2.3.1.

The computational complexity associated with determining the c-closure, beyond simply using it as a parameter, needs to be considered when contributing to this field of study. Understanding and optimizing the computation of c-closure could potentially further enhance the efficiency and applicability of algorithms parameterized by c.

When calculating the c-closure of a graph G the common neighborhood of every pair of non-adjacent vertices needs to be considered. This can be time consuming and we discuss the restrictions it puts on the c-closure metric later in this section.

To calculate the c-closure of a graph G, Fox et al. suggest squaring the adjacency matrix of G in $O(n^\omega)$ with $\omega < 2.373$ as the matrix multiplication exponent[5]. When squaring the adjacency matrix of a graph the entry in the (u, v) position of the squared adjacency matrix represents the number of paths on three vertices from u to v. This number is equivalent to the number of common neighbors of u and v seeing as any path between u an v on three vertices passing through a single vertex w implies that w is adjacent to both u and v. Even though this method performs well in theory, constructing these $n \times n$ adjacency matrices for large sparse graphs involves a lot of overhead and is not considered the optimal approach for such graphs. As this tend to be the case for social networks, with sparsely distributed nodes with a constant average low degree, we argue that other methods can perform just as well or better in practice.

We can also apply our knowledge from metrics related to the c-closure of a graph to develop more efficient algorithms to compute it. The low c-closure of social networks follows from the same reasoning as the abundance of closed triangles in social networks. Two non-adjacent nodes (u, v) with c common neighbors, also contributes to c wedges in the graph. If an edge forms between u and v the number of wedges decrease by c whilst the number of triangles increase by c. With this direct correlation between c-closure and the number of triangles and wedges in a graph, we can make alterations to known methods of wedge-counting to determine the c-closure of a graph. In section 3, we provide a worst case $O(n^3)$-algorithm for computing c-closure based on some of the most efficient results in triangle and wedge enumeration by Schank [16] that actually performs significantly better than $O(n^3)$ in practice.

Another approach to optimizing the computation of c-closure is tackling the constraint the complexity of computing the common neighborhood of nodes puts on the parameter.

By the definition of c-closure 3, we know that the size of the common neighborhood of any two non-adjacent pair of nodes are bounded by c. The sum of the sizes of all common neighborhoods of non-adjacent nodes is therefore $O(cn^2)$. However, an algorithm that calculates these in $O(cn^2)$ is yet to be found. Koana et al. [11] states that the discovery of such an algorithm would immediately yield better results for wedge enumeration in c-closed graphs , which is also what is done for the preproccessing stage for the maximal clique algorithm presented by Fox et al. [5].

As one of our main results, we provide a new algorithm exploiting another common measure of density in social networks, the *degeneracy* of a graph. The idea is to limit the cost of calculating common neighborhoods by limiting the size of the neighborhoods we are examining.

2.4 Degeneracy

Another metric commonly used in the analysis of social networks is the d-degeneracy of the graph. The d-degeneracy of a graph serves as a measure of the sparseness and robustness of a graph. More specifically, the degeneracy d represents the minimal degree found within the densest segment of a graph. The formal definition of degeneracy considers an ordered sequence of the graph's vertices such that each vertex has at most d adjacent vertices that occupy a higher position in the order. Koana et al. citekoana2020weaklyclosedsubgraphs, provide the following formal definition of a d-degenerate graph:

Definition 4. *A graph G is* d-degenerate *if one of the following two equivalent conditions holds:*

- *There exists a degeneracy ordering $\delta := (v_1, ..., v_n)$ of G, that is, an ordering such that $deg_{G_i}(v_i) \leq d$ where $G_i := G[\{v_i, ..., v_n\}]$*
- *Every induced subgraph G' of G has a vertex v with $deg_{G'}(v) \leq d$*

The degeneracy number d of the entire graph G is the lowest d such that G is d-degenerate.

Erdös and Hajnal[4] gave an equivalent definition of the coloring number of a graph, stating that the coloring number of a graph is the lowest number κ there exists an ordering for such that each node has κ or less neighbors previously in the ordering.

To compute the d-degeneracy ordering, sequentially remove the node with lowest degree and update the degrees of the remaining nodes. We present the pseudocode and correctness of computing the degeneracy ordering in Section 3.1.

The maximum degree of any node removed throughout the procedure determines the d-degeneracy of the graph. This is also refereed to as the maximal d-core subgraph. A

d-core subgraph is a subgraph of G that consists of the remaining nodes after exhaustively removing all nodes with degree less than d. It is worth noting that a d-core subgraph can consist of multiple disconnected cores, offering insights in the number of tightly-knit communities in a graph. The cores represent groups of nodes in the graph with high inter-connectivity.

The properties of the degeneracy ordering following its Definition 4 allows for a restricted search in the common neighborhoods when computing c-closure. We present an algorithm utilizing this property in Section 3.2.3.

Chapter 3

Algorithms

3.1 Computing degeneracy

As discussed in Section 2.4, computing the degeneracy ordering is a linear operation with respect to the number of nodes, n. During each iteration of the algorithm, the node with the current smallest degree is appended to the ordering and removed from the graph G. Using a suitable data-structure in the implementation of the algorithm, such as a minimum priority queue, results in a time complexity of $O(n \cdot \log(n))$. The initial step involves arranging the nodes in the queue according to their degrees. Seeing as each insertion operation into a priority queue typically requires $O(\log(n))$ time due to the data structure's tree-like nature, processing all nodes results in a total complexity of $O(n \cdot \log(n))$ for this preprocessing stage. The construction of the degeneracy ordering involves dequeueing the node with the smallest degree, an operation that takes $O(1)$ time. Furthermore, the degree of each node in the dequeued nodes neighborhood is decreased by one, and inserted to the queue without necessarily removing its previous version. To ensure that no node is appended to the ordering more than once, we keep track of all processed nodes. Consequently, any re-inserted node will always be processed before its preceding version with a higher degree, due to the properties of the priority queue.

For the ordering to serve its purpose in limiting the number of nodes needed to be processed in our algorithm, it is important to ensure that the left neighborhood of each node is bounded by degeneracy value, d, calculated by the algorithm. The correctness of this property follows from the construction of the algorithm. As each node is appended to the ordering, its current degree is equal to the count of its neighbors that have yet to be included in the ordering. Should this count exceed the current degeneracy value, d, it is updated. Upon the termination of the algorithm, the highest degeneracy value recorded throughout the iterations is returned. Thus, the construction of the algorithm validates that the ordering and degeneracy value corresponds.

Algorithm 1 Degeneracy Ordering

```
1: procedure DegeneracyOrdering($G$)
2:     $ordering \leftarrow empty\ list$
3:     $degeneracy \leftarrow 1$
4:     while $G \neq empty$ do
5:         $v \leftarrow$ vertex in $G$ with the minimum degree
6:         $ordering.add(v)$
7:         $degeneracy \leftarrow \max(degeneracy, degree(v))$
8:         $G.remove(v)$
9:     end while
10:    return $ordering$
11: end procedure
```

3.2 Computing c-closure

In Section 2.3.2 we present the constraints of computing the c-closure of a graph, and why reducing this cost can contribute to faster algorithms parameterized by c.

Based off of these we give three different algorithms for computing c-closure along with a discussion on its computational complexity.

3.2.1 Algorithm: Naive approach

In this section, we outline a straightforward brute force algorithm designed to compute the c-closure of a graph. This is achieved by simply counting the size of the common neighborhood for each pair of non-adjacent vertices in a graph G. While we have not incorporated this algorithm in our results for the real-world data-sets, primarily due to its extensive computation time and relative inefficiency compared to other approaches, it still serves as a useful point of reference. The brute force approach helps to illustrate the improvements achieved by our more fine-tuned algorithms, which we introduce in the upcoming Sections 3.1 and 3.2.3.

For each pair of nodes (u, v) in $E(G)$, the algorithm checks if they are not adjacent, calculates the size of their common neighborhood, and updates the c-value whenever a larger common neighborhood is found. The algorithm iterates over every pair of nodes in the graph, both adjacent and non-adjacent. Consequently, the complexity of iterating over the pairs of non-adjacent nodes are upper bound by $O(n^2)$ regardless. When a pair of non-adjacent vertices are found, the complexity of computing their common neighborhood is bounded by $O(\min(|N(u)|, |N(v)|)) = O(n)$.

The correctness of the algorithm follows from its construction, as it checks the common neighborhood of every non-adjacent pair of nodes in G.

Algorithm 2 CC-Bruteforce

```
 1: procedure CC-BRUTEFORCE($g$)
 2:     $c \leftarrow 0$
 3:     for each $u$ in $V(G)$ do
 4:         for each $v$ in $V(G)$ do
 5:             if $u \neq v$ and $(u, v) \notin E(V)$ then $c = max(c, [N(u) \cap N(v)])+$
 6:             end if
 7:         end for
 8:     end for
 9:     return $c$
10: end procedure
```

In conclusion, the overall complexity of the algorithm is upper bounded by $O(n^3)$. However, a somewhat tighter bound on the common neighborhood, based on the average degree of the graph, is discussed in the implementation of our degeneracy based algorithm in Section 3.1.

3.2.2 Algorithm: Wedge enumeration alteration

The direct correlation between the number of wedges in a graph and its c-closure lets us apply established techniques for triangle and wedge counting in the computation of c-closure.

A simple wedge-enumeration algorithm introduced by Koana and Nichterlein [11]. The algorithm considers all edges in a graph and computes the union of their neighborhoods. For each node found in the union but not in the intersection, a wedge is output. By counting all of the wedges for which two nodes are the endpoints, we can determine a graphs c-closure. Our findings is motivated in Theorem 4 as presented in [11], stating that there exists an $O(cn^2 + m^{\frac{3}{2}})$-time algorithm to compute the c-closure of a graph. The proof provided by the authors builds upon a constructive approach. Initially, all wedges are enumerated by the above procedure. Each wedge with endpoints in u and v is stored in a set denoted as $\mathcal{P}_3^{uv}$. The sets are sorted by size using radix sort. Radix sort completes the sorting operation in $O(n)$, where n represents the number of sets. The size of the largest set where u and v are not adjacent incremented by one yields the c-closure value of the graph.

The algorithm described above serves as the idea behind our altered algorithm. We want to compute the sizes of the sets of wedges for which two nodes contribute to, and this way obtain the c-closure value. We have also based our implementation on results from basic triangle enumeration presented by Schank [16].

By considering the nodes sorted by degree we can avoid checking the common neighborhoods of nodes that can not possibly contribute to a higher c. This alteration can be made as we do not need the total number of wedges in order to compute the c-value of a graph.

In our algorithm we initially sort the nodes by degree in decreasing order. This process has a worst case complexity of $O(n \log n)$.

As long as there are nodes remaining in the graph, choose the node u of maximum degree. For every node v adjacent to u, iterate over the neighborhood of v, $N(v)$. For each node w found in the $N(v)$, check if u and w are not adjacent. If they are not, the node w share at least one common friend with u and we increase the count of common neighbors between u and w by 1. If the updated value for common neighbors between two nodes found in the iteration exceeds the current largest value for c found, update the c value to the new, larger value. After considering every v and w for u, u is removed from the graph.

Before searching through the neighbors of the node u of current lowest degree picked at the start, we check if the degree $\deg(u)$ is lower than the current highest value for c. If this is the case, terminate the algorithm.

The worst case complexity of the algorithm sums up to $O(n^3)$ but because of our additional check, on average it performs considerably better than this.

When applied to our data-sets we can see that the wedge-alteration algorithm actually performs just as well if not better than the other algorithms considered for a majority of the data-sets.

3.2.3 Algorithm: Computing c-closure using degeneracy

We now provide a new algorithm based on the FPT-in-P design principles presented in section 2.2.2 for computing the c-closure. We argue that the algorithm is more efficient than the previous proposed methods, for graphs with a lower degeneracy value d than c-closure value c. Recall the best theoretical results from computing c-closure, based on fast matrix multiplication, as mentioned in Section 2.3.2. This method yields an $O(n^\omega)$-algorithm, where ω is the matrix exponent with $\omega < 2.373$. By parameterizing our approach using the degeneracy d of the graph, we are able to achieve a significantly smaller polynomial dependence on n, with $O(d^3 \cdot n^2)$-complexity in the worst case, but $O(d^3 \cdot n)$-complexity for 94% of our data sets.

Even though we do not, in the worst case, obtain a linear dependence in n by the principles of designing FPT-in-P algorithms, we still achieve $n^2 = O(n^\omega)$ for fixed d. However, for data-sets with $d < c$ the same algorithm yields an $O(d^3 \cdot n)$-algorithm, thus achieving a linear dependence in n.

Algorithm 3 CC-Wedge

```
procedure CC-WEDGE($G$)
    $c \leftarrow 0$
    $visited \leftarrow$ SET
    while $V(G)$ not empty do
        $u \leftarrow u \in u(G)$ of maximum degree
        $u\_wedges \leftarrow []$
        if $degree(u) < c$ then
            return $c$
        end if
        for each $v$ in $N(u)$ do
            for each $w$ in $N(v)$ do
                if $(u, w) \notin E(G)$ then
                    $u\_wedges[w] + +$
                    $c \leftarrow max(u\_wedges[w], c)$
                end if
            end for
        end for
    end while
    return $c$
end procedure
```

We now argue that replacing a squared dependence on n, $O(n^2)$, with a cubic dependency in d, $O(d^3)$ results in a significantly better run time for the majority of real-world data-sets. As we can see from our results in Table 3.3, even in the largest networks the degeneracy value d of the graph remains small. To further emphasize the improved bound, consider the `map-geology`-graph from Table 3.3 with $n \approx 3\,000\,000$ and $d = 13$. It is evident that the $O(d^3)$ factor is minor compared to the $O(n^2)$ dependency on n. The correctness of the $O(d^3 \cdot n)$-algorithm when $d < n$ will be given alongside the construction of the algorithm, in Section 3.2.3

The relationship between degeneracy and c-closure

While both c-closure and degeneracy serves as a measure of sparseness in a graph, it is important to note that the parameters do not necessarily correlate. In other words, a graph exhibiting low c-closure is not definitively associated with either high or low d-degeneracy values, and the same principles applies the other way around.

To illustrate the lack of correlation between c-closure and d-degeneracy, let us consider a complete graph with n nodes, where all nodes are pairwise adjacent, see Figure 3.1. In this scenario, the c-closure value c of the complete graph is 0, as there is no pair of nodes that are not directly connected. On the other hand, the d-degeneracy value d of the same graph equals the total number of nodes, $n-1$. This is because, regardless of which node v is removed first when constructing the degeneracy ordering of the graph, the node will have $n-1$ neighbors. This yields a degree of $n-1$, $\deg(v) = n-1$, resulting in a d-degeneracy value of $n-1$, $d = n-1$. Consequently, a complete graph exemplifies a scenario where a low c-closure coexists with a high d-degeneracy.

As a contrasting example, consider a complete bipartite graph with two nodes in one partition and an arbitrary number of nodes, n, in the other partition. In a complete bipartite graph there exists an edge between every node in one partition to every node in the other partition. We denote our example-graph $K_{2,n}$, see Figure 3.2. In this scenario, the two nodes in one partition have n common neighbors, resulting in a c-closure value of $n+1$. Yet, each of the nodes in the other partition has a degree of 2. Therefore, during the computation of the degeneracy ordering, by sequentially removing all nodes in the largest partition first, the resulting d-degeneracy value d will be 2 due to the iterative degree updates.

To exemplify a graph with both low c-closure and low d-degeneracy consider a chain graph of arbitrary length, see Figure 3.4. This structure, regardless of the number of nodes, will always yield a c-closure of $c = 2$ and a d-degeneracy of $d = 1$.

Conversely, a complete bipartite graph with a large number of nodes in both partitions serves as an example of high degeneracy and high c-closure, see Figure 3.3. This underscores the possibility of small values for both parameters simultaneously.

These examples demonstrate the absence of direct correlation between c-closure and d-degeneracy emphasizing their independent roles when computing and characterizing graphs later on in the thesis.

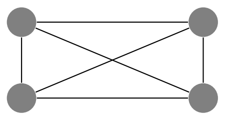
Figure 3.1: K_4

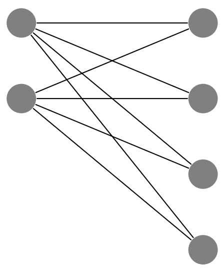
Figure 3.2: $K_{2,n}$

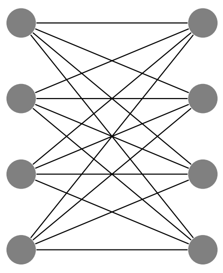
Figure 3.3: $K_{4,4}$

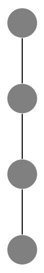
Figure 3.4: $P4$

The Algorithm

The goal of the algorithm remains somewhat the same as with the naive algorithm, we want to consider the size of the common neighborhood of two non-adjacent nodes u and v for every pair (u, v). The algorithm determines the c-closure of a graph with a $O(d^3 \cdot n^2)$-complexity in the worst case, and a reduced $O(d^3 \cdot n)$-complexity when $d < c$, by limiting the number of common neighborhoods that needs to be computed. We accomplish this by dividing the procedure into three distinct cases, each handling separate possibilities of pairings of non-adjacent vertices u and v. The construction and correctness of the cases will be argued in the remaining part of this section.

As a preprocessing stage of the algorithm we simply compute the degeneracy d and the degeneracy ordering of G. This is achieved by exhaustively removing the node $v \in G$ with the current lowest degree, thus decreasing the degree of the neighbors of v by 1 at each step, as described in Section 2.4. The order in which we remove the vertices is the degeneracy ordering. To compute the c-closure value c of a graph we can exploit the fact that for each node v in the ordering, a maximum of d neighbors of v appear before v in the ordering, by the definition of a degeneracy ordering 4. Using the degeneracy ordering we can limit the number of pairs we consider within a factor of d.

As parameters, the algorithm takes in a graph G, the degeneracy d and the degeneracy ordering, D. For each x in the ordering we want to find all pairs (u, v) where $u, v \in N(x)$ and $(u, v) \notin E(G)$. When iterating the ordering, x is the rightmost vertex in the common neighborhood of u and v, $N(u) \cap N(v)$. When trying to locate all non-adjacent pairs (u, v) adjacent to x we can picture the possible locations of the three nodes along the degeneracy ordering. As the ordering can be represented as a path of nodes, this yields three possible cases:

1. $u < v < x$: Both u and v are located in the left neighborhood of x, see Figure 3.5
2. $u < x < v$: The node x is located between u and v in the ordering, see Figure 3.6
3. $x < u < v$: The node x is located to the left of both u and v, see Figure 3.7

For all cases, u is located to the left of v. As u and v are nodes we label when we find them, the correctness of this follows from the way we iterate over the nodes.

Separating the three instances and only considering pairs of nodes with at least one common neighbor, x, lets us avoid the cost of computing the common neighborhood of two nodes with no common neighbors. Even though the size of the common neighborhood of two nodes is zero, the complexity of calculating it remains the same as it would with two nodes sharing a large common neighborhoods. As it is trivial to see that two nodes with no common neighbors cannot contribute to the c-closure, the correctness of not considering these combinations of pairs holds. We proceed to construct each case:

Case 1: $u < v < x$

In the first case the algorithm encounters, u and v are both located in the left neighborhood of x. From the definition of degeneracy we know that node x has a total of d neighbors to its left in the ordering. Thus, for each x we consider, we iterate over the possible combinations of two unique nodes, (u, v), in the left neighborhood of x, $N^{-}(x)$, which at

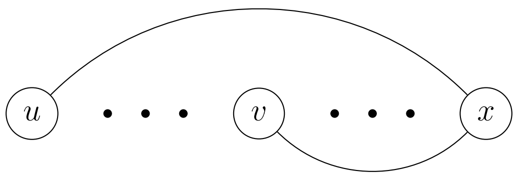

Figure 3.5: Case 1 $u < v < x$

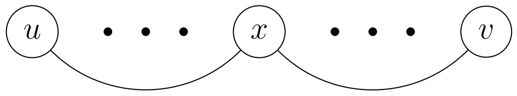

Figure 3.6: Case 2 $u < x < v$

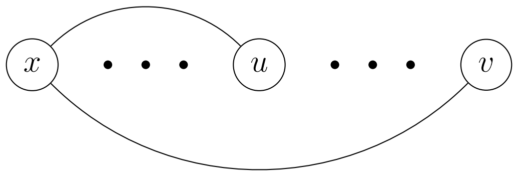

Figure 3.7: Case 3 $x < u < v$

most consists of $O(d^2)$ pairs. For each of the pairs (u, v), we check if u and v are adjacent. If they are not adjacent, we compute their common neighborhood in $O(\min(\deg(u), \deg(v)))$ time. Intuitively this would yield an $O(n)$ complexity, but because we exclusively choose the node with minimum degree for two vertices u and v and consider unique pairs of (u, v) for each iteration, the sum of nodes processed at this stage is bound by d times the average degree of the graph. The average degree of a graph is formally defined as avg $\deg(G) = \sum_{v \in V(G)} \frac{\deg_G(v)}{n}$. The sum of degrees can be substituted by 2 times the number of edges m, counting one for each endpoint u and v for a single edge $(u, v) \in E(G)$. In a degeneracy ordering we obtain the number of edges by simply summing up the left neighborhood for each node in the ordering inducing the following substitution, $\frac{1}{n} \cdot 2m = 2\frac{m}{n} = 2\frac{d \cdot n}{n} = 2d$. This replaces the original n-factor by a factor of d. The procedure is repeated for every x in the ordering, starting with x as the rightmost node in the neighborhood of u and v, yielding an $O(n \cdot d^2 \cdot d) = O(n \cdot d^3)$ complexity for case 1.

After terminating case 1, the algorithm has calculated some current value for c. We make the following claim:

Claim 1. *If the c-closure value, c, is higher than the degeneracy value, d, after case 1, return the current c-closure c and terminate the algorithm. This yields an $O(n \cdot d^3)$-complexity for* C-Closure Degeneracy *when $c > d$.*

The correctness of this claim follows from the definition of degeneracy.

After case 1, every pair of u and v in the left neighborhood of x, $N^{-}(x)$, have been considered and we are left with some value for c. We argue that if c is larger than d, we cannot find a larger c in case 2 or case 3.

In case 2, x is contained in the left neighborhood of v, $N^{-}(v)$. Recall that x is the rightmost node in the common neighborhood of u and v. If u and v has a common neighbor to the right of x, the pair has been considered in a previous iteration, as a consequence of choosing x in decreasing order. Because x is located in the right neighborhood of v, $N^{-}(v)$ with $N^{-}(v) \leq d$, and x is the rightmost common neighbor of u and v, the size of the common neighborhood of u and v is also bound by d. u and v having more than d common neighbors would contradict the property of x being the rightmost neighbor in their common neighborhood and we can safely claim that this can not be the case.

For case 3, the claim also holds as x is contained in the left neighborhood of both u and v which is more restrictive than only being contained in the left neighborhood of v, as in case 2.

It is clear that an increased c cannot be found when $c > d$ seeing as the sizes of the neighborhoods we are calculating are lower than the current c.

Assume this is not the case, and we have $d > c$ after the first case. We then proceed to argue the complexity of case 2:

Case 2: $u < x < v$

In the second case, we consider the pairs of u and v adjacent to x with vertex x positioned between vertices u and v in the ordering, such that $u < x < v$. Recall that x is the rightmost node in the common neighborhood of u and v.

Contrary to the first case, the set of neighbors for vertex x, where v may be located, is not bound by the degeneracy d of the graph and can require $O(n)$ time to locate. However, we make use of the fact that, 1: x is situated to the left of v and, 2: that x and v are adjacent. Rather than searching the unbounded right-neighborhood of x, we opt to instead search the left neighborhood of v, $N^{-}(v)$, to find x. For each node for x, we search its left neighborhood, $N^{-}(x)$ to identify u, because $u < x$. For each non-adjacent combination of v and u, calculate their common neighborhood. The complexity of calculating their common neighborhood follows the same line of argumentation as in case 1.

Consequently, the total time complexity case 2 contributes to in the algorithm is $O(n \cdot d \cdot d \cdot d) = O(n \cdot d^3)$.

We continue to argue the final part of our algorithm:

Case 3: $x < u < v$

By summarizing the runtime obtained for each case we get the following: $O(n \cdot d^3) +$

$O(n \cdot d^3) + O(n^2 \cdot d) = O(n^2 \cdot d + n \cdot d^3)$, when $c > d$. This completes our proof, and validates our claim.

Algorithm 4 C-Closure Degeneracy

```
1: procedure C_CLOSURE_DEGENERACY($G, D, d$)
2:     $c \leftarrow 0$
                                                   ▷ CASE 1: $u < v < x$
3:     for $x \in V(G)$ do
4:         for $\{u, v\} \subseteq N^-(x)$ do
5:             if $(u, v)$ not adjacent then
6:                 $c \leftarrow \max(c, |N(u) \cap N(v)|)$
7:             end if
8:         end for
9:     end for
10:    if $c \geq d$ then
11:        return $c$
12:    end if
                                                   ▷ CASE 2: $u < x < v$
13:    for $v \in V(G)$ do
14:        for $x \in N^-(v)$ do
15:            for $u \in N^-(x)$ do
16:                if $(u, v)$ not adjacent then
17:                    $c \leftarrow \max(c, |N(u) \cap N^-(v)|)$
18:                end if
19:            end for
20:        end for
21:    end for
                                                   ▷ CASE 3: $x < u < v$
22:    for $v \in V(G)$ do
23:        for $u \in V(G)$ do
24:            if $(u, v)$ not adjacent then
25:                $c \leftarrow \max(c, |N^-(u) \cap N^-(v)|)$
26:            end if
27:        end for
28:    end for
29:    return $c$
30: end procedure
```

3.3 Results

We provide new calculations for the degeneracy values d and c-closure values c for a wide range of network data sets. For every data set listed in Table 3.3, we ran the wedge enumeration algorithm denoted *Wedge*, see Algorithm 3 and two alterations of the algorithm using the degeneracy ordering denoted *Deg* and *Deg2*, see Algorithm 4.

We timed out each algorithm at 75 seconds for each data set, resulting in some undetermined values in our results. For each data set the c-closure has been calculated which can be used for future work in developing FPT-algorithms parameterized by c.

As we have mentioned in the run time analysis for the *C_closure_degeneracy*-Algorithm 4 in Section 3.2.3, for a majority of the data sets the computed degeneracy value d is significantly lower than the c-closure value c. In fact, for 94% of the data sets considered this is the case.

					Algorithm		
Dataset	n	m	d	c	Deg	Deg2	Wedge
iscas89-s27	9	8	1	1	0.19	0.08	∞
wafa-padgett	15	27	3	3	0.24	0.16	0.22
BioGrid-Human-Immunodeficiency-Virus-2	19	15	1	1	0.4	0.14	∞
wafa-hightech	21	159	12	16	1.12	0.54	0.83
wafa-ceos	26	93	5	6	0.67	0.49	0.65
BioGrid-Dictyostelium-Discoideum-Ax4	27	20	1	1	0.37	0.24	∞
seventh-graders	29	250	13	17	1.36	0.73	1.64
karate	34	78	4	6	0.77	0.31	0.36
windsurfers	43	336	11	16	4.59	2.18	4.97
BioGrid-Glycine-Max	44	39	2	2	0.47	0.28	0.3
wafa-eies	45	652	24	30	4.78	2.63	6.07
dutch-textiles	48	90	5	11	0.77	0.39	0.31
bergen	53	272	9	12	2.58	1.27	2.81
iscas89-s208.1	61	67	2	2	0.62	0.33	0.46
dolphins	62	159	4	4	2.31	1.17	1.98
train_bombing	64	243	10	8	1.76	0.98	1.65
BioGrid-Emericella-Nidulans-Fgsc-A4	64	62	2	2	1.35	0.77	0.66
pollination-tenerife	68	129	4	10	2.24	1.06	1.36
BioGrid-Cricetulus-Griseus	69	57	1	1	0.82	0.4	∞
Noordin-terror-relation	70	251	11	10	3.58	2.03	3.46
mg_watchmen	76	201	7	5	1.79	0.97	1.98
lesmiserables	77	254	9	8	1.91	0.97	1.75
mg_godfatherII	78	219	8	8	1.97	0.95	1.37
iscas89-s298	92	131	2	5	1.08	0.56	0.4
mg_forrestgump	94	271	8	3	5.69	3.24	8.2
win95pts	99	112	2	3	1.22	0.76	0.68
iscas89-s344	100	122	2	4	1.19	0.66	0.55
iscas89-s641	100	144	3	6	1.56	0.89	0.47
movies	101	192	3	5	2.19	1.01	1.36
iscas89-s349	102	127	2	4	1.11	0.58	0.51
polbooks	105	441	6	15	3.33	1.52	1.8
mg_casino	109	326	9	7	2.15	1.18	3.22
word_adjacencies	112	425	6	13	7.01	3.18	4.36
hypertext_2009	113	2196	28	53	62.32	24.62	26.8
iscas89-s386	114	200	3	12	1.5	0.78	0.44
StackOverflow-tags	115	245	6	6	1.84	1.06	1.32
football	115	613	8	9	5.6	3.09	5.43
iscas89-s382	116	168	2	4	8.56	11.43	4.64
iscas89-s400	121	182	2	4	1.64	0.9	0.94
BioGrid-Human-Herpesvirus-5	121	107	1	1	1.3	0.76	∞
Noordin-terror-loc	127	190	3	6	2.41	0.94	0.9

iscas89-s420.1	129	145	2	2	1.57	0.89	1.1
Noordin-terror-orgas	129	181	3	10	1.44	0.74	0.55
iscas89-s444	134	206	2	4	1.55	0.88	0.82
BioGrid-Hepatitus-C-Virus	136	134	1	1	1.11	0.68	∞
iscas89-s713	137	180	3	6	1.72	0.94	0.57
capitalist	139	1071	19	34	9.97	4.48	11.62
foodweb-otago	141	832	14	35	21.51	8.12	2.64
american_revolution	141	160	3	9	1.37	0.79	0.5
BioGrid-Canis-Familiaris	143	125	2	2	1.59	0.93	0.69
iscas89-s526n	159	268	3	8	1.95	1.03	0.67
iscas89-s526	160	270	3	8	2.7	1.35	0.98
iscas89-s510	172	251	2	2	2.3	1.9	2.26
BioGrid-Human-Papillomavirus∞6	173	186	2	17	2.02	1.07	0.72
BioGrid-Human-Herpesvirus∞	178	208	3	10	1.96	1.17	0.95
CoW-interstate	182	319	4	12	2.51	1.27	1.1
jazz	198	2742	29	41	32.54	13.71	26.02
mousebrain	213	16089	111	178	1642.88	762.78	215.11
residence_hall	217	1839	11	17	23.6	8.85	18.95
airlines	235	1297	13	42	11.61	5.1	6.95
sp_data_school_day_2	238	5539	33	53	300.87	113.66	162.43
iscas89-s820	239	480	3	15	4.44	2.24	0.96
rhesusbrain	242	3054	19	37	60.64	22.88	37.81
iscas89-s832	245	498	3	18	4.68	1.85	1.04
BioGrid-Danio-Rerio	261	266	3	3	2.54	1.46	1.61
iscas89-s838.1	265	301	2	2	3.14	1.66	2.14
haggle	274	2124	39	40	12.21	8.04	35.53
celegans	297	2148	10	40	39.44	11.75	7.68
BioGrid-Human-Herpesvirus-4	323	326	2	5	3.61	2.52	1.59
hex	331	930	3	2	44.95	36.81	∞
iscas89-s953	332	454	2	3	5.14	2.39	3.15
autobahn	374	478	2	2	4.76	2.72	3.21
photoviz_dynamic	376	610	4	12	6.06	3.62	1.94
iscas89-s1196	377	537	2	2	7.07	3.69	4.66
ia-infect-dublin	410	2765	17	22	31.79	13.79	20.57
BioGrid-Gallus-Gallus	413	436	4	21	9.14	4.86	2.46
iscas89-s1238	416	625	2	2	9.37	5.16	8.32
ecoli-transcript	423	578	3	10	7.39	5.43	3.46
iscas89-s1423	423	554	2	4	5.79	3.09	2.43
muenchen-bahn	447	578	2	1	13.01	11.67	∞
BioGrid-Bos-Taurus	454	424	3	6	3.99	5.05	2.31
iscas89-s1488	463	779	3	10	10.2	4.13	1.73
iscas89-s1494	473	796	3	10	6.68	3.36	1.84
foodweb-caribbean	492	3313	13	154	155.73	56	4.76
pigs	492	592	2	6	27.91	12.69	6.26
ratbrain	503	23030	67	126	283.06	191.58	490.32
BioGrid-Human-Herpesvirus-8	716	691	3	5	7.44	13.14	2.61
codeminer	724	1015	4	6	9.68	5.36	4.35
pollination-daphni	797	2933	9	44	49.52	19.11	9.48
cpan-authors	839	2112	9	51	24.62	9.55	9.94
columbia-mobility	863	4147	9	11	59.05	17.8	54.19
columbia-social	863	7724	18	53	97.48	41.52	74.4
unicode_languages	868	1255	4	18	13.78	6.7	6.23
soc-wiki-Vote	889	2914	9	17	31.51	12.22	20.23
link-pedigree	898	1125	2	13	10.64	5.78	2.6
Opsahl-forum	899	7036	14	37	247.82	77.38	97.07
pollination-uk	984	16712	35	201	4405.58	1564.44	99.42

EU-email-core	986	16064	34	161	929.57	304.67	102.07
roget-thesaurus	1010	3648	6	7	211.81	66.02	139.66
bn-mouse_retina_1	1076	90811	121	402	36565.4	16115.6	4621.39
BioGrid-Candida-Albicans-Sc5314	1121	1609	9	63	15.24	7.66	4.77
ia-email-univ	1133	5451	11	18	82.51	33.15	62.46
BioGrid-Human-Immunodeficiency-Virus∞	1138	1319	3	42	16.43	8.88	5.17
euroroad	1174	1417	2	3	14.44	8.45	6.81
BioGrid-Far-Western	1199	1089	3	10	52.91	35.14	15.38
polblogs	1224	16715	36	129	1346.81	460.73	170.67
BioGrid-Escherichia-Coli-K12-Mg1655	1273	1889	5	12	22.26	12.35	8.99
web-google	1299	2773	17	17	24.93	14.15	8.52
munin	1324	1397	3	28	29.66	15.86	6.94
iscas89-s5378	1411	1639	3	4	20.08	10.73	6
diseasome	1419	2738	11	8	21.21	12.36	15.42
BioGrid-Dosage-Growth-Defect	1447	2193	5	69	26.74	11.54	4.92
netscience	1461	2742	19	5	27.1	17.65	13.54
chicago	1467	1298	1	1	14.52	9.25	∞
pollination-carlinville	1500	15255	18	66	952.73	287.03	90.5
bitcoin-otc-negative	1606	3259	16	38	60.83	25.98	15.8
BioGrid-Fret	1700	2395	19	37	22.54	12.86	8.72
BioGrid-Dosage-Lethality	1776	2289	4	32	121.63	56.08	39.81
bn-fly-drosophila_medulla_1	1781	8911	18	41	186.84	67.01	151.23
BioGrid-Affinity-Capture-Luminescence	1840	2312	6	37	29.09	15.99	9.43
DNC-emails	1866	4384	17	73	65	28.97	33.12
wikipedia-norm	1881	15372	22	137	1473.37	431.54	316.97
exnet-water	1893	2416	2	2	23.79	12.75	14.6
Opsahl-socnet	1899	13838	20	111	914.45	275.78	103.55
Y2H_union	1966	2705	4	29	40.51	21.68	9.62
iscas89-s9234	1985	2370	4	9	21.92	12.55	5.18
NZ_legal	2141	15739	25	128	765.81	237.13	150.1
BioGrid-Co-Crystal-Structure	2291	2021	5	5	27.38	17.37	10.17
Yeast	2361	7182	10	21	73.51	33.06	35.78
soc-hamsterster	2426	16630	24	76	404.47	132.93	135.83
iscas89-s13207	2492	3406	4	30	31.79	17.65	6.39
moreno_health	2539	10455	7	16	92.34	44.43	26.38
minnesota	2642	3303	2	2	30.05	16.44	23.29
ODLIS	2900	16377	12	85	544.97	138.17	123.45
openflights	2939	15677	28	110	514.41	257.52	192.62
iscas89-s15850	3247	4004	4	15	38.89	21.93	8.59
BioGrid-Dosage-Rescue	3380	6444	7	34	57.77	30.46	12.62
BioGrid-Co-Localization	3543	4452	6	24	42.52	23.59	12.1
twittercrawl	3656	154824	142	506	∞	31298.5	579.41
BioGrid-Escherichia-Coli-K12-W3110	4063	181620	156	441	∞	∞	6967.28
boards_gender_1m	4134	19993	25	19	183.48	100.88	128.57
boards_gender_2m	4220	5598	4	20	54.26	32.88	20.74
web-EPA	4271	8909	6	63	457.42	232.95	102.18
BioGrid-Co-Purification	4326	5970	12	70	90.07	39.27	15.59
ingredients	4372	431654	297	835	∞	∞	52127.6
advogato	5155	39285	25	215	2603.04	672.53	144.92
soc-advogato	5167	39432	25	217	2703.6	726.77	140.9
ca-GrQc	5242	14496	43	42	138.33	76.39	52.01
bitcoin-otc-positive	5573	18591	20	91	543.61	181.51	188.11
JUNG-javax	6120	50290	65	618	11918.3	3046.9	164.81
web-california	6175	15969	11	68	298.26	176.93	39.81
reactome	6327	147547	176	445	15803.3	6422.56	181.19
BioGrid-Caenorhabditis-Elegans	6394	23646	64	81	1088.83	495	509.43

JDK_dependency	6434	53658	65	706	9180.59	2321.43	125.04
as20000102	6474	12572	12	42	477.83	159.92	319.75
zewail	6651	54182	18	98	2098.01	780.9	370.15
ia-reality	6809	7680	5	18	90.87	49.09	48.24
wiki-vote	7115	100762	53	440	26818.3	7723.15	503.66
eva-corporate	7253	6711	3	6	64.73	42.88	26.84
chess	7301	55899	29	63	2811.21	907.18	1252.1
lederberg	8324	41532	15	174	1473.78	489.98	141.31
BioGrid-Biochemical-Activity	8620	17746	11	181	477.31	164.14	72.51
iscas89-s38584	9193	12573	4	12	154.05	89.05	33.45
BioGrid-Drosophila-Melanogaster	9330	60556	83	114	2976.28	881.29	428.86
iscas89-s38417	9500	10635	4	11	583.84	312.07	132.94
movielens_1m	9746	1000209	221	2355	∞	∞	7612.18
BioGrid-Arabidopsis-Thaliana-Columbia	10417	47916	26	384	3727.6	1183.21	64.81
p2p-Gnutella04	10876	39994	7	28	1410.29	238.19	79.77
BioGrid-Co-Fractionation	11017	56354	83	83	1589.05	667.48	601.44
AS-oregon∞	11174	23409	17	74	540.25	214.36	467.52
AS-oregon-2	11461	32730	31	117	881.34	283.61	507.99
ca-HepPh	12006	118489	238	89	4223.35	2179.1	5809.21
ukroad	12378	15641	3	4	152.03	98.52	32.34
iscas89-s35932	12515	15961	2	1	8683.81	19835.2	∞
foldoc	13356	91471	12	62	1930.61	719.78	373.99
BioGrid-Affinity-Capture-Rna	13765	42815	54	1629	6815.16	2201.38	152.56
escorts	16730	39044	11	68	924.48	379.98	174.65
marvel	19428	96662	18	744	16837	5580.36	272.25
BioGrid-Affinity-Capture-Western	21028	64046	17	92	1705.95	602.9	433.16
as-22july06	22963	48436	25	217	1893.41	608.05	916.96
edinburgh_associative_thesaurus	23132	297094	34	204	63635.8	14940.1	5234.23
ca-CondMat	23133	93439	25	26	1821.29	937.45	1263.38
cora_citation	23166	89157	13	69	1466.61	633.56	193.42
google+	23628	39194	12	489	1547.25	446.16	189.74
soc-gplus	23628	39194	12	489	1511.75	470.67	173.32
BioGrid-Homo-Sapiens	24093	369767	71	801	∞	66570.3	5797.07
cit-HepTh	27769	352285	37	1669	43545.1	11724.6	257.65
digg	30398	86312	10	19	1889.52	821.53	1753.49
linux	30834	213217	23	4835	∞	50150.8	549.76
BioGrid-Chemicals	33266	28093	1	1	302.02	214.42	∞
cit-HepPh	34546	420877	30	299	35749.2	11218.1	1077.14
email-Enron	36692	183831	43	186	19597.6	5297.92	4772.45
BioGrid-Affinity-Capture-Ms	40495	321887	58	842	∞	22773	1847.17
slashdot_threads	51083	117378	15	64	4827.19	1590.45	2527.13
deezer	54573	498202	21	82	20741.8	893079	3278.69
loc-brightkite_edges	58228	214078	52	183	8417.44	3745.35	1572.27
facebook-links	63731	817090	52	232	∞	42368.4	13507.9
BioGrid-All	75550	1316843	164	1791	∞	∞	4066.71
soc-Epinions1	75879	405740	67	550	∞	33899.6	5919.28
soc-Slashdot0811	77360	469180	54	617	∞	25663.8	2536.62
NYClimateMarch2014	102378	327080	34	1036	∞	14548.7	3135.96
livemocha	104103	2193083	92	∞	∞	∞	∞
tv_tropes	152093	3232134	115	2654	∞	∞	41177.8
gowalla	196591	950327	51	1131	∞	31112.2	6484.08
bahamas	219856	246291	6	57	10359.6	3342.71	860.32
location	225486	293697	5	853	66911.6	6611.02	835.75
offshore	278877	505965	13	5697	∞	17383.8	1023.31
dogster_friendships	426820	8546581	249	32803	∞	∞	1.09764e+06
Cannes2013	438089	835892	27	3167	∞	21512.5	2582.39

actor_movies	511463	1470404	14	216	46932.2	21746.8	1589.21
paradise	542102	794545	23	4241	51289.8	15224.5	1600.58
panama	556686	702437	62	1947	18667.8	10530.8	1519.89
countries	592414	624402	6	11047	∞	25819.4	1733.09
teams	935591	1366466	9	350	∞	47165.9	4751.34
mag_geology_coauthor	2852295	4448428	13	192	∞	73773.8	7574.13

Chapter 4

Conclusion

We give a new algorithm that is faster, both in theory and in practice, for computing the c-closure of a graph. The algorithm is a FPT-in-P algorithm parameterized by the degeneracy d. For 94% of the real-world graphs we considered, the algorithm runs in $O(n \cdot d^3)$ time, whilst for the remaining it runs in $O(n^2 \cdot d + n \cdot d^3)$ time. The latter only happens when the degeneracy value of the graph is larger than the c-closure value of the graph, which has shown to be rare for social networks. In the case when $d = O(\sqrt[3]{n})$ our algorithm is faster than the current best algorithm to compute c, regardless of the value of c.

In addition to an improved algorithm, we provide the c-closure value of a large amount of social and biological networks.

Bibliography

[1] Balaram Behera, Edin Husić, Shweta Jain, Tim Roughgarden, and C. Seshadhri. Fpt algorithms for finding near-cliques in c-closed graphs. 2021.

[2] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. ISBN 978-3-319-21274-6. doi: 10.1007/978-3-319-21275-3.
URL: https://doi.org/10.1007/978-3-319-21275-3.

[3] Vicki Dowling. *John Guare's Six degrees of separation : a text response guide / Vicki Dowling*. Warringal Publications Fitzroy, Vic, 2001. ISBN 1876557427.

[4] P Erdős and A Hajnal. On chromatic number of graphs and set-systems. *Acta Mathematica Academiae Scientiarum Hungarica*, 17(1):61–99, March 1966.

[5] Jacob Fox, Tim Roughgarden, C. Seshadhri, Fan Wei, and Nicole Wein. Finding cliques in social networks: A new distribution-free model. *SIAM Journal on Computing*, 49(2):448–464, 2020. doi: 10.1137/18M1210459.
URL: https://doi.org/10.1137/18M1210459.

[6] Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *Theoretical Computer Science*, 689:67–95, 2017. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2017.05.017.
URL: https://www.sciencedirect.com/science/article/pii/S030439751730422X.

[7] Mohammad Hasan and Vachik Dave. Triangle counting in large networks: a review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8, 10 2017. doi: 10.1002/widm.1226.

[8] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978. doi: 10.1137/0207033.
URL: https://doi.org/10.1137/0207033.

[9] Lawqueen Kanesh, Jayakrishnan Madathil, Sanjukta Roy, Abhishek Sahu, and Saket Saurabh. Further Exploiting c-Closure for FPT Algorithms and Kernels for Domination Problems. 219:39:1–39:20, 2022. ISSN 1868-8969. doi: 10.4230/LIPIcs.STACS.2022.39.
URL: https://drops.dagstuhl.de/opus/volltexte/2022/15849.

[10] Tomohiro Koana and André Nichterlein. Detecting and enumerating small induced subgraphs in c-closed graphs. *Discrete Applied Mathematics*, 302:198–207, 2021. ISSN 0166-218X. doi: https://doi.org/10.1016/j.dam.2021.06.019.
URL: https://www.sciencedirect.com/science/article/pii/S0166218X21002572.

[11] Tomohiro Koana and André Nichterlein. Detecting and enumerating small induced subgraphs in c-closed graphs. *Discrete Applied Mathematics*, 302:198–207, 2021. ISSN 0166-218X. doi: https://doi.org/10.1016/j.dam.2021.06.019.
URL: https://www.sciencedirect.com/science/article/pii/S0166218X21002572.

[12] Tomohiro Koana, Christian Komusiewicz, and Frank Sommer. Exploiting c-closure in kernelization algorithms for graph problems. *SIAM Journal on Discrete Mathematics*, 36(4):2798–2821, 2022. doi: 10.1137/21M1449476.
URL: https://doi.org/10.1137/21M1449476.

[13] Tomohiro Koana, Christian Komusiewicz, and Frank Sommer. Computing dense and sparse subgraphs of weakly closed graphs. *Algorithmica*, Jan 2023. ISSN 1432-0541. doi: 10.1007/s00453-022-01090-z.
URL: https://doi.org/10.1007/s00453-022-01090-z.

[14] Tomohiro Koana, Christian Komusiewicz, and Frank Sommer. Essentially tight kernels for (weakly) closed graphs. *Algorithmica*, 85:1–30, 01 2023. doi: 10.1007/s00453-022-01088-7.

[15] Daniel Lokshtanov and Vaishali Surianarayanan. Dominating set in weakly closed graphs is fixed parameter tractable. 213:29:1–29:17, 2021. ISSN 1868-8969. doi: 10.4230/LIPIcs.FSTTCS.2021.29.
URL: https://drops.dagstuhl.de/opus/volltexte/2021/15540.

[16] Thomas Schank. Algorithmic aspects of triangle-based network analysis. 2007.