Counting small patterns in networks

A dissertation presented
by

Tomaž Hočevar

to
The Faculty of Computer and Information Science
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy
in the subject of
Computer and Information Science

Ljubljana, 2018

APPROVAL

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgement has been made in the text.

— Tomaž Hočevar —

January 2018

The submission has been approved by

dr. Borut Robič

Professor of Computer and Information Science

EXAMINER

dr. Gašper Fijavž

Professor of Mathematics

EXAMINER

dr. Yuval Shavitt

Professor of Electrical Engineering

EXTERNAL EXAMINER

Tel Aviv University

dr. Janez Demšar

Professor of Computer and Information Science

ADVISOR

PREVIOUS PUBLICATION

I hereby declare that the research reported herein was previously published/submitted for publication in peer reviewed journals or publicly presented at the following occasions:

[1] T. Hočevar and J. Demšar. A combinatorial approach to graphlet counting. *Bioinformatics*, 30(4):559–565, 2014. doi: 10.1093/bioinformatics/btt717

A pre-copyedited, author-produced version of an article accepted for publication in Bioinformatics following peer review is included. The version of record *Tomaž Hočevar, Janez Demšar; A combinatorial approach to graphlet counting. Bioinformatics 2014; 30 (4): 559-565* is available online at https://academic.oup.com/bioinformatics/article/30/4/559/205331/A-combinatorial-approach-to-graphlet-counting.

[2] T. Hočevar and J. Demšar. Computation of Graphlet Orbits for Nodes and Edges in Sparse Graphs. *Journal of Statistical Software*, 71(10):1–24, 2016. doi: 10.18637/jss.v071.i10

[3] T. Hočevar and J. Demšar. Combinatorial algorithm for counting small induced graphs and orbits. *PLOS ONE*, 12(2):1–17, 2017. doi: 10.1371/journal.pone.0171428

I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as graduate student at the University of Ljubljana.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*
Tomaž Hočevar
Štetje majhnih vzorcev v omrežjih

POVZETEK

Omrežja pogosto uporabljamo za vizualizacijo in analizo relacij med pari entitet, ki jih predstavimo z množico vozlišč, povezave med njimi pa predstavljajo relacije. Ena izmed relacij v bioinformatiki, ki jo pogosto modeliramo z omrežji, so interakcije med pari proteinov. Nedavne študije v zvezi z lokalno strukturo takih omrežij so uporabljale majhne povezane vzorce s 4 ali 5 vozlišči, ki jim rečemo tudi grafki. Vozlišča grafkov se običajno delijo v orbite glede na njihovo "vlogo" oz. simetrije. Kolikokrat neko vozlišče v omrežju nastopa v vsaki izmed orbit, predstavlja neke vrste podpis lokalne strukture v okolici vozlišča. Z zanašanjem na predpostavko, da je lokalna struktura vozlišča povezana z njegovo funkcijo v omrežju, je raziskovalcem uspelo z uporabo grafkov napovedati nove funkcije proteinov.

Glavna ovira pristopov na osnovi grafkov je običajno v času, ki ga zahteva štetje grafkov. Ta omejitev je vedno bolj izrazita zaradi vedno večje količine razpoložljivih podatkov. Disertacija se posveča izboljšavi obstoječih metod za štetje grafkov. Te namreč delujejo na osnovi enostavnega izčrpnega naštevanja vseh grafkov v omrežju.

V disertaciji predstavljen algoritem Orca prešteje grafke, ne da bi jih naštel, kot to počnejo ostale metode. Izkorišča povezave med frekvencami orbit za pripravo sistema enačb, ki ga sestavi izredno učinkovito. Orca za štetje grafkov velikosti k našteje zgolj grafke velikosti $k-1$. Tako doseže pohitritev, ki je sorazmerna največji stopnji vozlišča v omrežju. V praksi to pomeni, da prešteje grafke v večjih omrežjih proteinskih interakcij 50 do 100-krat hitreje.

Algoritem Orca je bil v osnovi razvit za štetje grafkov in orbit vozlišč velikosti 4 in 5. Pristop smo uspešno prilagodili tudi štetju orbit povezav z enakimi prihranki glede časa izvajanja. Rešitev je možno posplošiti za štetje poljubno velikih grafkov. V ta namen smo identificirali potrebne pogoje in dokazali, da jih je mogoče izpolniti tudi v primeru štetja večjih grafkov.

Disertacija se posveča tudi problemu generiranja naključnih omrežij s predpisano porazdelitvijo grafkov. Ta problem predstavlja motivacijo za prilagoditev algoritma Orca za uporabo v dinamičnih oz. spreminjajočih omrežjih, kjer lahko nastajajo nove povezave ali pa obstoječe propadajo. Spremembe so lahko posledica postopka za generiranje naključnega omrežja ali pa so del procesa, ki ga omrežje modelira. Generirana omrežja se zelo približajo želeni porazdelitvi grafkov. Poleg števila grafkov pa so si podobna tudi po drugih merah lokalne strukture omrežij.

Razviti algoritem je pomembno orodje za analizo omrežij z grafki in predstavlja pomemben korak k analizi večjih in gostejših omrežij. Kot najhitrejša metoda štetja grafkov je tudi osnova nadaljnjega raziskovanja učinkovitih metod štetja vzorcev v omrežjih.

Doktorska disertacija temelji na treh objavljenih znanstvenih člankih, ki skupaj s poglavjem, ki vsebuje še neobjavljeno delo, tvorijo jedro disertacije.

Ključne besede: grafki, orbite, omrežje, graf, podgraf, vzorec, štetje

University *of Ljubljana*
Faculty *of Computer and Information Science*
Tomaž Hočevar
Counting small patterns in networks

ABSTRACT

Networks are an often employed tool that can help us visualize and analyze binary relationships by representing the entities as a set of nodes and the relations between them as edges in the network. One type of relations in the field of bioinformatics that is often modeled by networks are interactions between pairs of proteins. Recent studies have focused on analyzing the local structure of such networks by observing small connected patterns consisting of 4 or 5 nodes, which are also known as graphlets. The nodes of graphlets are further divided into orbits by their "roles" or symmetries. The number of times a node from the network participates in each orbit forms a signature of the node's local network topology. Working under the assumption that the node's local topology is correlated with its function in the network, researchers have successfully used graphlets to predict new protein functions.

The bottleneck of graphlet-based approaches is usually in the time required to count them. This restriction is becoming even more pronounced with a growing amount of available data. This dissertation focuses on improving existing graphlet counting techniques that are based on simple exhaustive enumeration.

We present the algorithm Orca that counts graphlets and their orbits instead of enumerating them. It exploits relations between orbit counts to construct a system of equations that can be set up efficiently. Orca achieves this by enumerating $(k-1)$-node graphlets to count k-node graphlets, effectively obtaining a speed-up by a factor proportional to the maximum degree of a node in the network. In practical terms, it counts graphlets in larger protein-protein interaction networks about 50-100 times faster.

Orca was designed for counting graphlets with 4 and 5 nodes. However, we adapt the approach to counting edge-orbits in addition to the original node-orbits with the same gains in run time. We also show that this approach can be generalized to graphlets

of arbitrary size by identifying the necessary conditions and proving that these conditions can be fulfilled even for larger graphlets.

Finally, we consider the problem of generating random graphs with prescribed graphlet distributions. This motivated the adaptation of Orca for dynamic or changing networks, where edges can be added or removed. These changes can be a consequence of the procedure for generating a random graph or can be inherent in the network and the process it models. The generated graphs closely match the desired graphlet counts and as a consequence approximate other structural measures as well.

The developed algorithm is a valuable tool for graphlet-based network analysis and a significant stepping stone towards analyzing larger and denser networks. As the fastest graphlet counting method it also presents a basis for further development of efficient pattern counting methods in graphs.

This doctoral dissertation is based on three published papers that together with a chapter containing some unpublished work form the core of the dissertation.

Key words: graphlets, orbits, network, graph, subgraph, pattern, counting

ACKNOWLEDGEMENTS

I would like to thank my parents who introduced me to programming and computer science and my advisor for guiding me through the unpredictable waters of research and publishing. I am also especially grateful to all my colleagues who were involved in algorithmic discussions—related or unrelated to the topic of this dissertation but contributing to the final result nevertheless.

— Tomaž Hočevar, Ljubljana, January 2018.

CONTENTS

I

Introduction

1.1 Problem definition

Networks naturally arise in many different disciplines and are modeled as graphs. Small patterns in these graphs can uncover the basic building blocks of networks and help us understand the structure and properties of these networks. However, their discovery and counting remain computationally intensive tasks. We can estimate the pattern frequencies on random samples, yet in some applications we would prefer to know their exact frequencies. This motivates the research of efficient methods for exact pattern counting in sparse graphs.

We will refer to these small connected patterns that occur as induced subgraphs in the network as *graphlets* [1]. For an even finer description, the nodes of graphlets can be further divided into their automorphism *orbits* [2] or roles of individual nodes. Figure 3.1 illustrates all 4-node and 5-node graphlets along with their orbits. The graphlet counting problem consists of computing the orbit distribution for every node in the graph—how many times does a node in the graph participate in each of the orbits? This gives us a topological signature of a node's local neighbourhood.

Problem definition Given a simple graph G, we would like to compute every node's frequency distribution of orbits from graphlets with at most k nodes that appear as induced subgraphs in G. Figure 4.2 illustrates the problem on a simple toy network.

Graphlet and orbit distributions can be viewed as generalizations of a node degree, which corresponds to the first non-trivial graphlet consisting of two connected nodes. We can extend the notion of orbits to edges and observe edge-orbits [3] as defined by edge-automorphisms of the patterns. The number of common neighbours of two adjacent nodes is equivalent to the number of triangles spanning over a given edge. This suggests that an edge-orbit distribution provides a generalization of the number of common neighbours, which is an important feature in network analysis.

The graphlet counting problem differs from *motif* detection in networks [4, 5]. However, the term motif is sometimes also used in the same way as graphlet or pattern and is not limited to the statistically too frequent patterns. We are interested in frequencies of all patterns because the most infrequent ones might also be the most important in a given area of application. For a similar reason we also focus on exact computation although there have been several attempts at approximating graphlet distributions [6–8].

1.2 Areas of application

Network analysis with graphlet distributions has been so far used mainly in bioinformatics but is in no way limited to this area [9]. Protein-protein interaction (PPI) networks have been the main subject of graphlet analysis. They have been used to find random network models [1] that fit real PPI networks, cluster proteins and predict new protein properties [10] and predict new genes related to cancer [11] or aging [12].

Graphlets can also assist in other analytic methods, such as global network alignment. GRAAL [13] is an algorithm for aligning arbitrary networks based solely on their topology, which employs a local topology similarity measure based on graphlet degree vectors. The technique was used to show the large amount of shared network topology between yeast and human PPI networks, which can be used to predict biological functions of aligned proteins or reconstruct phylogenetic trees. H-GRAAL [14] aligns networks by reducing the problem to a weighted bipartite matching that can be solved with Hungarian algorithm [15]. Finally, MI-GRAAL [16] integrates multiple sources of node similarity information, including the graphlet degree vectors.

1.3 Motivation

There is a number of graphlet counting tools, which are used in bioinformatics. FANMOD [17] is a network motif detection tool based on sampling random subgraphs and comparing their counts with those from random network models. Whelan [18] developed GraphletCounter, which works as a Cytoscape [19] plugin and merges graphlet analysis with visual inspection of the network. GraphCrunch [20, 21] also provides methods for further analysis and comparison of computed graphlet distributions. Rapid Graphlet Enumerator (RAGE) [22] is one example of a faster graphlet counting algorithm. Instead of counting the induced subgraphs directly, it reconstructs them from counts of non-induced subgraphs. Unlike FANMOD and GraphCrunch, RAGE works only for up to four-node graphlets.

All previously mentioned methods rely on an exhaustive enumeration of graphlets. Despite limiting the analysis to patterns on at most five nodes, this already presents a significant problem on larger PPI networks. For example, an estimated run time of GraphCrunch for counting graphlets of size five in human PPI network from the BioGRID database [23] is several months. One possibility is to exploit hardware optimizations [24] which reduce the time by a constant factor, but we will need new

algorithmic approaches with such fast growth of available data and size of networks.

1.4 Theoretical background

A classic result by Itai and Rodeh [25] refers to counting triangles in a graph. Raising the graph's adjacency matrix A to the third power gives the number of paths of length 3 between pairs of nodes. Elements $A^3_{x,x}$ give the number of paths of length 3 that start and finish in the node x, which equals twice the number of triangles that include x. The total number of triangles is then $\frac{1}{6}\sum_{x\in G} A^3_{x,x}$. Note that the same triangle is counted twice for each of its three nodes. The time complexity of this procedure equals that of multiplying two matrices. A natural extension of this result deals with counting larger cliques of size k in a graph. More precisely, can this be accomplished faster than with an exhaustive enumeration which would require $O(n^k)$ time in dense graphs? Nešetřil and Poljak [26] showed that clique detection problem can indeed be solved faster by reducing the original problem to detection of triangles in a graph with $O(n^{k/3})$ nodes. Since we can detect triangles faster than in $O(n^3)$ with fast matrix multiplication algorithms [27, 28], we can also detect cliques of size k faster than $O(n^k)$.

A different approach exploits the relations between the numbers of occurrences of induced subgraphs in a graph. Kloks et al. [29] presented a system of equations that allows computing the number of occurrences of all six possible induced four-node subgraphs if we know the count of any of them. The time complexity of setting up the system equals the time complexity of multiplying two square matrices of size n. Kowaluk et al. [30] generalized the result by Kloks to counting subgraph patterns of arbitrary size. Their solution depends on the size of the independent set in the pattern graph and fast matrix multiplication techniques.

It is interesting to note that the problem of counting all induced patterns is equivalent to counting all non-induced patterns. These counts are connected through a simple system of equations. However, counting a single pattern is easier in some cases of non-induced patterns. All induced patterns seem to be equally difficult to count [30], while non-induced patterns with large independent sets seem to present an easier problem. For example, counting a star graph with k nodes can be achieved simply by summing $\binom{\deg(x)}{k-1}$ over all nodes x.

How distinctive are graphlets as a signature for topological structure of networks? A closely related problem is the Reconstruction conjecture [31], which states that any

graph on at least three nodes is uniquely determined by the multiset of its vertex-deleted subgraphs. The conjecture has been verified for graphs on up to 11 nodes by McKay [32]. However, the frequency of small connected patterns (graphlets) does not uniquely define the graph itself. Fig. 1.1 shows the smallest (in terms of the number of nodes and edges) example of two such graphs. The graphs are not isomorphic: the first one is planar while the second one contains $K_{3,3}$ (a complete bipartite graph with three nodes on each side), which is a known non-planar graph. However, they contain the same number of all 5-node graphlets. For example, they both contain 12 edges (graphlet 0), no triangles (graphlet 2), 2 star graphs S_4 (graphlet 11), etc.

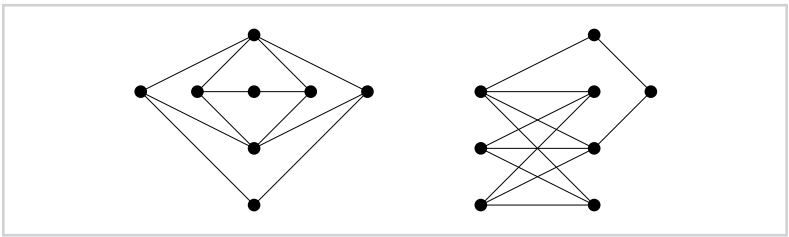


Figure 1.1

An example of two non-isomorphic graphs with equal 5-node graphlet counts.

1.5 Methodology

Existing methods consist of a simple exhaustive enumeration or rely on fast matrix multiplication techniques. The former are becoming too slow on larger networks and the latter are unsuitable on sparse networks that typically arise in practical applications. Our research was focused on developing a new method that outperforms exhaustive enumeration methods and is suitable for sparse networks.

We focus on developing a counting algorithm instead of trying to optimize enumeration approaches. No matter how optimized such enumeration approach is to avoid listing the same item several times, it still has to list each of them at least once. A counting approach can improve upon such methods by observing some patterns and by considering entire batches of items in a single operation, i.e. adding more than 1 to the accumulated result in one operation.

A trivial example is that of counting even numbers in the range from 1 to n. Iterating over all numbers in the range and checking whether a number is odd or even is clearly a waste of time. The pattern is obvious: every other number is even, therefore there

are $\lfloor n/2 \rfloor$ of them. Things get slightly more complicated if we're interested in numbers divisible by 2 or 3. There are $\lfloor n/2 \rfloor$ multiples of 2 and $\lfloor n/3 \rfloor$ multiples of 3. But we over-counted those that are divisible by 2 and 3, i.e. multiples of 6. Subtracting $\lfloor n/6 \rfloor$ gives the correct answer. A generalization of this is the inclusion-exclusion principle.

Let's move on to counting problems in graphs. Suppose we want to count the number of triangles. One approach is to consider every edge in the graph and try to count in how many triangles it occurs. We will obviously over-count the triangles. However, the good news is that we will over-count them exactly three times. Every triangle will be considered once for each of its three edge. How can we efficiently count triangles spanning over a given edge in the graph without actually considering all triangles? A short answer is: not in a simple way. This is exactly the number of common neighbours of edge's endpoints. By adding the degrees of the endpoints we would consider all common neighbours but also the non-common ones. We would obtain the number of triangles and the number of paths of length 2 that contain this edge.

Constructing an expression to count exactly that one pattern, which we set out to count, turns out to be rather difficult. Instead we change our approach to constructing an entire system of such equations. As long as the number of these equations is large enough and they are independent, we can solve the system to obtain the numbers we want. Moreover, the system of equations must be efficient to set up.

1.6 Scientific contributions

This section summarizes the principal scientific contributions. With each contribution I list the dissertation sections where the topic is discussed and the references to the publications of which I was the first author. The listed references were published in internationally renowned scientific journals and were thus internationally reviewed and discussed. All article were published in journals from the top half of the Science Citation Index (SCI) by their Impact Factor in at least one field of research.

The following scientific contributions are presented in this dissertation:

- *An orbit counting algorithm for four-node and five-node graphlets*

  I developed an Orbit counting algorithm (Orca) for counting graphlets and orbits. The algorithm is capable of counting node- and edge-orbits of four- and five-node graphlets. Its substantial advantage over existing graphlet counting

methods comes from a combinatorial approach to counting instead of enumerating.

The contribution is covered by chapters 3 and 4 that contain reformated versions of journal papers [33] and [34].

- *A general algorithm for counting node and edge orbits*

  I generalized the approach for generating equations required by the Orca algorithm and proved that such equations can be derived for arbitrarily large graphlets and are therefore not limited to sizes four and five.

  The paper [35] addressing this topic is included in chapter 5.

- *A dynamic orbit counting algorithm for changing networks*

  I adapted Orca for the dynamic setting with interleaved additions and removals of edges and graphlet count queries on individual nodes. This dynamic algorithm can be used to generate random networks with desired graphlet counts.

  Chapter 6 contains our so far unpublished research results.

2

Overview

This chapter gives an overview of the published journal papers that comprise this dissertation. The papers have been reformatted to fit the dissertation template and contain some minor corrections that were suggested by the committee members.

Recent development of high-throughput experimental techniques for detecting protein interactions [36, 37] led to a rapid increase in the amount of available data and establishment of large protein interaction databases [23, 38]. Consequently, there is also an increasing need for more efficient algorithms for processing this data, which is usually presented in the form of networks. One of the concepts used in connection with PPI networks are graphlets. Counts of these small patterns have been used as a signature of local network topology to predict protein functions [10] and as a statistic for many other applications. The first paper presents our algorithm Orca that addresses this problem.

Graphlet analysis has been recently extended from the concept of node-orbits to edge-orbits in an attempt to overcome the problem of node membership in several functional groups simultaneously. By clustering edges instead of nodes (according to their graphlet signatures) the authors of [3] were able to identify new pathogen-interacting proteins and hence new drug target candidates. The second paper addresses the problem of counting edge-orbits by employing a similar approach as the one used in our initial algorithm Orca. We also present a graphlet counting library for programming language R. In contrast with the first paper, this one focuses on the implementation details that lead to observed speed-ups.

After improving node- and edge-orbit counting algorithms for 4- and 5-node graphlets we turn to the problem of computing orbits counts of larger graphlets. More specifically, can the approach of our Orca algorithm be generalized to graphlets of arbitrary size? This problem is of more theoretical interest because the number of graphlets grows rapidly with their size (Table 2.1). 853-dimensional node description vectors would be quite sparse and not too informative for analyzing networks. Besides, 7-node graphlets would cover most of the network anyway, contradicting the idea of graphlet frequencies as a measure of local topology.

Some of our yet unpublished work on adapting the algorithm to a dynamic setting of the problem and its use for generating graphs with desired graphlet frequencies is presented in Chapter 6. Some networks exhibit dynamic properties where new connections are being added and old ones removed. Because each connection change affects only its local neighbourhood, we can efficiently update algorithm's internal structures.

Table 2.1

Number of k-node graphlets. See also http://oeis.org/A001349.

k	2	3	4	5	6	7	8	9	10
graphlets	1	2	6	21	112	853	11117	261080	11716571

We developed a dynamic version of Orca for counting 4-node graphlets. The algorithm supports additions and removals of edges that can be interleaved with graphlet counting queries for a specific node. The addition and removal operations efficiently update the internal data structures. Queries can then use these values to set up the system of equations and finally solve it to obtain the actual orbit counts.

The dynamic algorithm was developed with a specific application in mind—how to generate a random graph that closely approximates given graphlet frequencies. Unfortunately, the developed dynamic version of Orca was not fast enough for our approach to generating random graphs and was optimized even further for maintaining the graphlet counts in networks after every modification (addition or removal of an edge).

2.1 A combinatorial approach to graphlet counting

In this paper we present our graphlet counting algorithm Orca. We focus on the practical use of the algorithm in bioinformatics for counting graphlets in PPI networks.

The algorithm exploits relations between orbit counts for efficient computation. Systems of equations by Kloks [29] and Kowaluk [30] rely on fast matrix multiplication techniques and are therefore not feasible on sparse networks. Our results show that we can systematically design a different system of equations for four-node and five-node graphlets with the purpose of efficient graphlet counting in sparse graphs. Our system of equations also relates orbit counts as opposed to only graphlet counts. The proposed system of equations is obtained through observing possible extensions of smaller patterns with another node. To make use of these relations we have to select and efficiently enumerate a single orbit while the other orbit counts can be computed from the previously derived relations. For this, we focus on cliques because there are very few of them in sparse networks and we can efficiently enumerate them with some known enumeration algorithms [39, 40].

We evaluated our algorithm by comparing it with existing methods on PPI net-

works. The algorithm exhibits a speed-up proportional to the maximum degree of a node in the graph, which coincides with the theoretical estimate of its time complexity. It is 50-100 times faster on the larger PPI networks and as much as 1800 times faster on the largest human PPI network that we used in our experiments.

A reformated version of the paper [33] is inserted as Chapter 3.

2.2 *Computation of graphlet orbits for nodes and edges in sparse graphs*

This paper presents a package `orca` for programming language `R` that is capable of efficiently counting node-orbits and edge-orbits of 4-node and 5-node graphlets. Note that we could reduce the problem of counting edge-orbits to counting node-orbits in line graphs. However, the obtained line graphs would be much larger in terms of the number of nodes in the network as well as in the size of the patterns.

We derive edge-orbit relations using a similar technique as for deriving node-orbit relations. For every edge-orbit of a graphlet we are able to select a special node such that we can list all patterns without this special node and for each of them count the number of missing special nodes that exist in the host graph. In contrast to counting node-orbits, this special node should not coincide with the endpoints of the edge.

We provide a system of equations that can be employed in a graphlet counting algorithm and show how the obtained system translates to the implementation of the algorithm. The expected speed of counting edge-orbits is the same as that for counting node-orbits. The paper concludes with a demonstration of the use of the developed package. The demo finds related concepts or entities in the Wikipedia for Schools network.

A reformated version of the paper [34] is inserted as Chapter 4.

2.3 *Combinatorial algorithm for counting small induced graphs and orbits*

After a successful development of Orca for 4-node and 5-node graphlets we generalized our algorithm to counting orbits of graphlets of arbitrary size. Melckenbeeck et al. [41] attempted to do the same but overlooked a crucial requirement that the system should be efficient to build from a given network in order to offer the promised speed-up. Their systems of equations, while correct, does not lead to an efficient algorithm.

We identified the necessary conditions for systematically constructing an efficient system of equations relating the orbit counts. Next, we proved that we can indeed construct equations satisfying these conditions for graphlets of arbitrary size. The paper also contains an analysis of algorithm's expected time complexity on random Erdos-Renyi graphs.

A reformated version of the paper [35] is inserted as Chapter 5.

3

A combinatorial approach to graphlet counting

Bioinformatics

A combinatorial approach to graphlet counting

Tomaž Hočevar and Janez Demšar

Faculty of Computer and Information Science, University of Ljubljana,
SI-1000 Ljubljana, Slovenia

3.1 Abstract

Motivation: Small induced subgraphs called graphlets are emerging as a possible tool for exploration of global and local structure of networks, and for analysis of roles of individual nodes. One of the obstacles to their wider use is the computational complexity of algorithms for their discovery and counting.
Results: We propose a new, combinatorial method for counting graphlets and orbit signatures of network nodes. The algorithm builds a system of equations that connect counts of orbits from graphlets with up to 5 nodes, which allows to compute all orbit counts by enumerating just a single one. This reduces its practical time complexity in sparse graphs by an order of magnitude as compared to the existing, pure enumeration-based algorithms.
Availability: Source code is available freely at
http://www.biolab.si/supp/orca/orca.html
Contact: tomaz.hocevar@fri.uni-lj.si

3.2 Introduction

Following the advent of high-throughput methods more than a decade ago, analysis of complex network data has assumed the central role among computational methods

in bioinformatics. The huge size of such networks on one hand, and the computational intractability of the related methods on the other, have spawned a number of innovative analytic approaches.

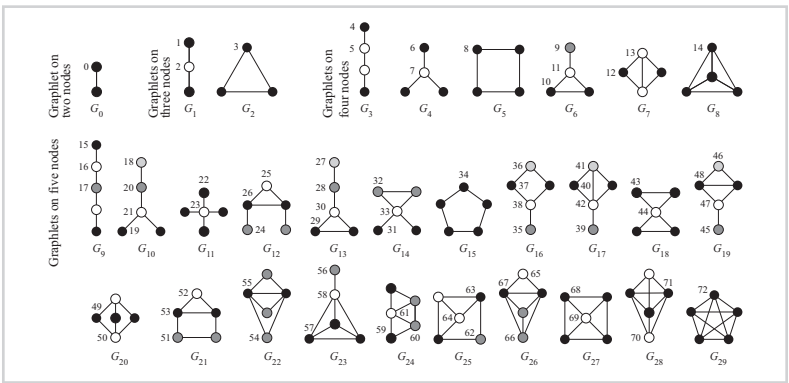


Figure 3.1

Graphlets with 2–5 nodes and automorphism orbits. Notation follows [2]. Colors are chosen arbitrarily; nodes of the same color belong to the same orbit within that graphlet, *e.g.* both black nodes in G_{14} belong to orbit 31.

Przulj et al. [1] described an approach focused on small induced subgraphs called graphlets. Due to combinatorial explosion, such analysis is usually limited to the 30 graphlets with 2–5 nodes (Fig. 3.1). The number of appearances of graphlets in the network provides a description of the network's structural properties. On a local level, counting how many times a particular node participates in each kind of graphlet induced in the network gives a topological signature of the node's neighbourhood represented as a 30-dimensional vector.

For a finer description, the nodes of every graphlet are partitioned into orbits [2], which are equivalence classes under the action of its automorphism group. Two nodes belong to the same orbit if they map to each other in some isomorphic projection of the graphlet onto itself. Nodes of graphlets on 2–5 points are grouped into 73 orbits shown by numbers and node colors in Figure 3.1. For instance, the five nodes from G_{14} belong to three different orbits, marked with different colors and numbers; the black (as well as the gray) nodes have symmetric positions in the graphlet and thus belong to the same orbit (31 for the black, 32 for the gray), and the white node belongs to the orbit 33. By counting the number of times a node of a graph appears in each orbit, the node can be described by a 73-dimensional vector of orbit counts, which reflects its position with respect to the local structure and gives insight into its role in

the network.

Existing methods for counting the graphlets and orbits are based on direct enumeration: in order to count them, they need to find all their embeddings in the network. We propose a new method, Orbit Counting Algorithm (Orca), which reduces the time complexity by an order of magnitude by computing the orbit counts using the relations between them and directly enumerating only smaller graphlets.

3.2.1 Motivation

Graphlets are used for different kinds of analyses in bioinformatics. Milenković and Przulj [10] designed a method for comparing node neighbourhoods based on graphlets and demonstrated that clusters of nodes in protein-protein interaction (PPI) networks, obtained with their graphlet-based distance measure, share common protein properties. They showed how to use this approach to predict functions of proteins and their memberships in protein complexes, subcellular compartments and tissue expressions. Milenkovic et al. [11] studied the relation between cancer genes and their network topology. They examined several clustering methods based on a graphlet similarity measure and found a difference between the PPI network structure around the cancer and non-cancer genes. Around 80% of the predicted cancer gene candidates have indeed been validated in the literature. Similarly, cost functions for network alignment that are based on graphlet degree vectors show superior results in comparison with other state-of-the-art methods. In particular, Milenković et al. [12] showed how alignment between the PPI networks of *S. cerevisiae*, *D. melanogaster* and *C. elegans* with the human PPI network can be used for identification of genes related to aging, which are difficult to observe directly for humans due to our long lifespans. Milenković et al. [42] also applied graphlets to estimate node's topological centrality. Their graphlet degree centrality measure is based on graphlet degree vectors and captures density and complexity of a node's extended neighbourhood. They showed that the genes participating in key biological processes also reside in complex and dense parts of networks.

Hayes et al. [43] argue that in order to understand the biological networks, we need to find the mathematical models describing their structure, even though this may not be of direct predictive use. Przulj et al. [1] used graphlet distributions to show that geometric graphs match the structure of protein-protein interaction (PPI) networks better than Erdős-Rényi and scale-free graph models. Using a number of large PPI networks, Hayes et al. [43] further showed that while the network structure may be

unstable in regions with low edge density, high density regions are suitable for network comparison using graphlet degree distributions.

Graphlets can also assist in other analytic methods, such as global network alignment. GRAAL [13] is an algorithm for aligning arbitrary networks based solely on their topology, which employs a local topology similarity measure based on graphlet degree vectors. The technique was used to show the large amount of shared network topology between yeast and human PPI networks, which can be used to predict biological functions of aligned proteins or reconstruct phylogenetic trees. H-GRAAL [14] aligns networks by reducing the problem to a weighted bipartite matching that can be solved with Hungarian algorithm. Finally, MI-GRAAL [16] integrates multiple sources of node similarity information, including the graphlet degree vectors.

Solava et al. [3] extended the use of graphlets by defining the orbits for graphlet edges and demonstrated their use with a new clustering method that is not limited to locally similar edges and allows some overlap between clusters. As a practical result, they predicted new pathogen-interacting proteins from clusters in the human PPI network that represent drug target candidates.

Graphlet analysis is therefore a useful tool for bioinformatics and with the increase of available data there is also a growing need for fast graphlet counting tools.

3.2.2 *Related work*

We will denote the explored graph as $G = (V, E)$. Let $n = |V|$ and $e = |E|$ be the number of nodes and edges, and let d denote the maximal node degree. Let $N(u)$ denote the set of nodes adjacent to node u. In numbering the graphlets and orbits, we follow Przulj [2]; we refer to the j-th graphlet and i-th orbit by G_j and O_i, respectively.

Counting subgraphs is a computationally intensive task. Common approaches to speed it up include sampling [6–8], exploiting pattern symmetries [44] or using reconfigurable hardware accelerators based on FPGA chips [24].

The method described in this paper is related to the approach developed by Kloks et al. [29], who constructed a system of equations that allows computing the number of occurrences of all six induced four-node subgraphs by knowing the count of any of them. The time complexity of setting up the system equals the time complexity of multiplying two square matrices of size n. We extend this approach to counting how many times each node participates in each orbit. Our method also works on five-node graphlets and scales better on sparse graphs. Kowaluk et al. [45] generalized the result

by Kloks *et al.* to count subgraph patterns of arbitrary size.

There are several programs for graphlet counting and motif detection that are used in bioinformatics. FANMOD [17] is a network motif detection tool based on sampling random subgraphs and comparing their counts with those from random network models. Besides implementing a novel sampling algorithm [8] it also provides a full enumeration procedure for graphlets on 2–8 nodes. Whelan and Sönmez [18] developed GraphletCounter, which works as a Cytoscape plugin and merges graphlet analysis with visual inspection of the network.

GraphCrunch [20] is a tool for large network analysis. It includes a function for computing orbit signatures of every graph node for graphlets of up to five nodes using an enumeration procedure with correction for over-counting some of the graphlets. A well-organized enumeration method imposes constraints that eliminate the need for isomorphism testing except for distinguishing between a few different graphlets; this is further accelerated by comparing the number of edges and individual node degrees. GraphCrunch has been extended with a new method for topological network alignment and with comparison of the networks with some additional mathematical models [21]. The graphlet counting procedure in the new version remained essentially the same.

Rapid Graphlet Enumerator (RAGE) [22] takes a different approach to counting four-node graphlets. Instead of counting the induced subgraphs directly, it reconstructs them from counts of non-induced subgraphs. For computing the latter, it uses specifically crafted methods for each of the 6 possible subgraphs (G_3 to G_8 in Fig. 3.1). The time complexity of counting non-induced cycles and complete graphs is $O(e \cdot d + e^2)$, while counting other subgraphs requires $O(e \cdot d)$. Another bound, which is also more suitable for comparison with our method, is $O(e \cdot d^2) = O(n \cdot d^3)$. Unlike FANMOD and GraphCrunch, RAGE works only for up to four-node graphlets.

3.3 Methods

Let x represent a certain node of interest in graph G. Our task is to compute the number of times, o_i, that x appears in each orbit O_i across all graphlets induced in G. We will present an approach based on a system of linear equations that relate the orbit counts o_i. The rank of the system is smaller than the number of orbits by one, so we can compute all values of o_i from directly enumerating only a single one. The algorithm allows to compute the orbits for all points x in a graph in time that is smaller

than the existing, direct enumeration approaches by an order of magnitude.

We will first show how to construct a system of equations for four-node graphlets. As for the single orbit that must be enumerated, we chose O_{14}, which represents nodes of the complete graph, K_4 (or G_8); we show an efficient way to enumerate it. The approach used for four-node graphlets is less suitable for larger graphlets, so we present a different technique for five-node graphlets.

3.3.1 Orbits in four-node graphlets

Right-hand sides of equations we are about to construct contain terms that are computed from the graph G. Let $c(u,v) = |N(u) \cap N(v)|$ denote the number of common neighbours of nodes u and v. Let $p(u,v)$ denote the number of paths on three nodes that start at node u, continue with v and end with some node t, which is not connected to u. We can compute $p(u,v)$ as $p(u,v) = deg(v) - 1 - c(u,v)$.

If some node x participates in a k-node graphlet G_i, it also participates in some $(k-1)$-node graphlet G_j. This can be seen by removing one of the graphlet's nodes that are the farthest away from x. The subgraph induced by the remaining nodes is connected (otherwise a whole component of the resulting graph would be farther from x), so it is isomorphic to some $(k-1)$-node graphlet G_j.

We will use this observation in reverse: every four-node graphlet can be constructed by adding a node to some three-node graphlet(s). To find the relations between counts of orbits in four-node graphlets for a certain node x, we enumerate all three-node graphlets touching the node and count their possible extensions with the fourth node.

An example is shown in Figure 3.2. Nodes x, y and z induce graphlet G_1, a path on three nodes; we will observe its extensions to four-node graphlets with the fourth node, w, connected to y and z (dashed lines). The number of such nodes w is $c(y,z)$. In our example, there are $c(y,z) = 3$ such nodes, which we marked by w_1, w_2 and w_3 (Fig. 3.2(a)). The edge (x,w) might exist in the graph G (as in the case of w_3, the dotted line) or not (as for w_1 and w_2). With no edge, nodes x, y, z and w form a paw (G_6) with x in orbit O_9 (Fig. 3.2(b)). With an edge between x and w, they form a diamond (G_7) with x in orbit O_{12} (Fig. 3.2(c)). Since all $c(y,z)$ nodes in $N(y) \cap N(z)$ must participate either in G_6 or G_7, which puts x in O_9 or O_{12}, this gives $o_9 + o_{12} = c(y,z)$ for the particular triplet x, y and z.

We sum this over all possible three-node paths starting at x. Summation must account for symmetries: each graphlet G_6 appearing in the graph is counted twice with

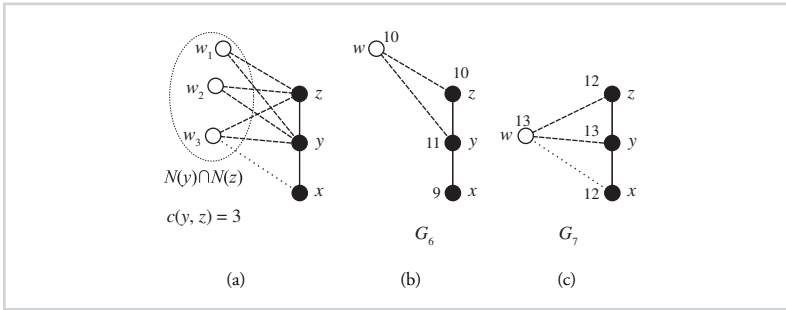


Figure 3.2

Relation between orbits O_9 and O_{12}. Solid lines are edges in the three-node graphlet being extended. Dashed lines exist by definition: w (or w_i) are the common neighbours of y and z. Dotted lines are optional edges that make the resulting four-node graphlet on x, y, z and w_i isomorphic to G_6 or G_7.

roles of z and w reversed, and G_7 is counted twice with reversed roles of y and w. Accounting for this, we get

$$2o_9 + 2o_{12} = \sum_{\substack{y,z:\, x,z \in N(y) \\ G[\{x,y,z\}] \cong G_1}} c(y,z),$$

where $\cong$ denotes graph isomorphism (*e.g.*, $G[\{x,y,z\}]$, a subgraph on nodes x, y, z, is isomorphic to G_1, a path with three nodes).

For a different example, we will relate orbits O_6 and O_9. We will extend a path on nodes x, y and z with another path that starts with nodes x and y; we denoted the number of such paths by $p(x,y)$ (Fig. 3.3(a)). Depending on whether the new node is adjacent to z, the extended graphlet is either a claw (G_4, Fig. 3.3(b)) or a paw (G_6, Fig. 3.3(c)). After accounting for symmetries and subtracting 1 since $p(x,y)$ also covers the case when $w = z$, we get

$$2o_6 + 2o_9 = \sum_{\substack{y,z:\, x,z \in N(y) \\ G[\{x,y,z\}] \cong G_1}} \big(p\,(x,y) - 1\big)\,.$$

There are only two three-node graphlets and relatively few possible extensions. Investigating all possibilities in a similar manner yields 10 linearly independent equations with 11 variables that correspond to counts of 11 orbits in four-node graphlets (see the Supplementary).

Right-hand sides depend on the graph G and need to be computed for each point x. To accelerate their computation, we precompute values of $c(u,v)$ and $p(u,v)$. In all

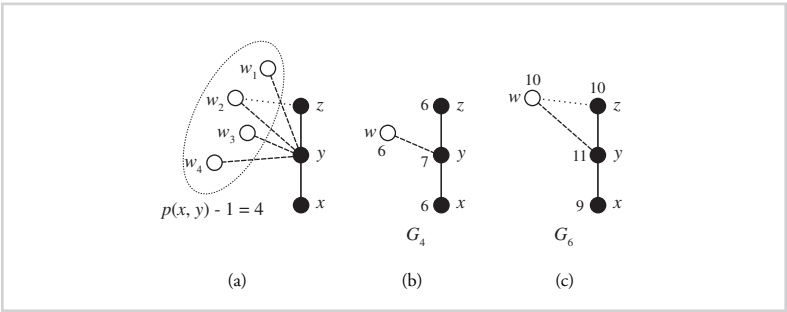


Figure 3.3

Relation between orbits O_6 and O_9. Edges are marked like in Figure 3.2.

equations, except for the last one, $c(u,v)$ is computed on pairs of nodes (u,v) that are connected; in $p(u,v)$, they are connected by the definition of p. It therefore suffices to precompute $c(u,v)$ and $p(u,v)$ only for all pairs of adjacent nodes u and v, which requires $O(e)$ space. The last equation, in which the new node closes a cycle, is treated separately. Nodes x and z are not adjacent but we can pre-compute the number of paths of length 2 that start at node x and end at node y. This requires $O(n)$ space for each point; since we compute orbits for one point at a time, this memory can be recycled. Altogether, all lookups in the sums on the right-hand sides can be done in constant time by sacrificing the memory of size $O(e+n)$ for precomputed values $c(x,y)$ and $p(x,y)$.

The total time complexity for computing all orbits for all nodes is $O(e \cdot d + T_4)$, where $O(T_4)$ is the time needed to enumerate complete graphlets on four nodes. Below, we describe an algorithm that does this in $O(n \cdot d^3)$, yet the actual importance of this term depends on the structure and density of the graph.

3.3.2 Counting complete graphlets

For every node we still have to determine the count of one of the 11 orbits. Since graphs are usually sparse, a good candidate is the rare orbit 14, which represents the nodes of the complete graphlet on four nodes G_8. Because of very few occurrences of this graphlet and its symmetricity, we can efficiently restrict the enumeration.

A straightforward way to count the complete graphlets of size 4 that touch a given node x_1 is to start with that node and in every step add a neighbour x_i of the last added node x_{i-1}, while checking that the new node is also connected to all nodes before x_i,

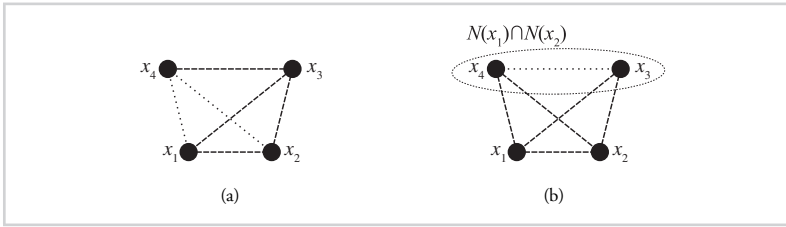


Figure 3.4

Enumerating G_8 by adding one neighbour at a time or by checking pairs of neighbours. Dashed edges are added by iterating through neighbours, and dotted edges are checked in the last step.

$x_{j<i-1}$. In this way, when we add x_4 as a neighbour of x_3 we have to check whether it is connected to x_1 and x_2 (dotted lines in 3.4(a)), which is unlikely, especially in sparse graphs.

A better strategy is to find the common neighbours of x_1 and x_2, $N(x_1) \cap N(x_2)$, which can be done in $O(d)$. We then choose pairs (x_3, x_4) from this set and check whether they are connected (Fig. 3.4(b)). Candidates generated in this way have to satisfy only one additional condition, as opposed to two in the straightforward approach.

To avoid counting the same graphlet multiple times, we request that $x_2 < x_3 < x_4$ under some fixed arbitrary ordering of nodes. Although the theoretical time complexity for finding all G_8 that touch x using this algorithm is the same for both approaches, $O(d^3)$, the latter is much faster on sparse graphs.

This method can be generalized for efficient counting of larger complete graphlets in sparse graphs. In every step, we maintain a list of candidate nodes C_i for x_i that are adjacent to all previously added nodes. We select one of these candidates and form a new candidate set C_{i+1} consisting only of nodes in C_i that are adjacent to the selected node, $C_{i+1} = C_i \cap N(x_i)$ and $C_1 = V$. The time complexity of finding all complete k-node graphlets that touch x using this algorithm is $O(d^{k-1})$. Below, we use such procedure to enumerate complete subgraphs on five nodes.

3.3.3 Orbits on five-node graphlets

For counting four-node graphlets, we constructed a list of equations by adding nodes to three-node graphlets and observing the resulting four-node graphlets. Extending the four-node graphlets to five-node graphlets would yield a huge number of equations that are not linearly independent. We will use a different approach: for each orbit, we choose some node y from the corresponding graphlet and observe the graphlets and

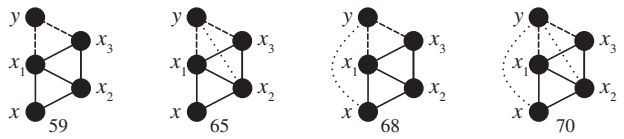


Figure 3.5

Computing orbit count o_{59}; figures show graphlets for different edges between y and other nodes, and the orbits of x.

orbits in which the node of interest, x, appears if we add edges between y and other nodes in the graphlet.

Let x be the node of interest, let y be the node whose edges we observe and let x_1, x_2 and x_3 be the other three nodes in that graphlet.

Figure 3.5 illustrates counting of appearances of x in O_{59}, which belongs to G_{24} (Fig. 3.5(a)). We will focus on the node marked by y, which is connected to the nodes marked by x_1 and x_3. Note that removing y reduces G_{24} into a diamond, G_7, with x in orbit O_{12}.

Now assume that we are computing orbits for a certain node x and discover some induced subgraph $H \cong G_7$ with x in O_{12}. We assign labels x_1, x_2 and x_3 to the remaining nodes as shown in the figure. Altogether, the graph G contains $c(x_1, x_3)$ common neighbours of x_1 and x_3 (similar to nodes marked with w in Fig.3.3(a)). While all these nodes are – by definition of $c(x_1, x_3)$ – connected to x_1 and x_3, some are also connected to x_2 or x, or both. Figure 3.5 shows all four possibilities, which give graphlets G_{24}, G_{26}, G_{27} and G_{28} with x in orbits 59, 65, 68 and 70, respectively. Therefore, $o'_{59} + o'_{65} + o'_{68} + o'_{70} = c(x_1, x_3) - 1$, where o'_i denote orbits of x with respect to H.

For the relation between o_{59}, o_{65}, o_{68} and o_{70} for the entire graph, we sum this over all possible induced G_7 with x in O_{12}. After considering the symmetries that cause counting the same graphlet multiple times with different assignments of y, x_1, x_2 and x_3, we get

$$o_{59} + 4o_{65} + 2o_{68} + 6o_{70} = \sum_{\substack{x_1,x_2,x_3: \\ x_1<x_2 \wedge x_3 \notin N(x), \\ G[\{x,x_1,x_2,x_3\}] \cong G_7}} c(x_1, x_3) + c(x_2, x_3) - 2.$$

Condition $x_1 < x_2$ (under some arbitrary ordering of nodes) is needed to consider each graphlet G_7 just once. The other two conditions put x in O_{12}. The second term in the sum, $c(x_2, x_3)$, accounts for the case in which the roles of x_1 and x_2 are exchanged.

Using a similar construction for other orbits, except for O_{72}, gives 57 linear equations for 58 orbits (see the Supplementary). Like for four-node graphlets, we directly enumerate the orbit O_{72}, which belongs to the complete graphlet. Equations are linearly independent due to the way in which they were constructed: each equation is set up with one orbit in mind (*e.g.* O_{59} in the above example) and the other orbits that appear in the equation belong to graphlets with a larger number of edges (the additional edges between y and the other nodes, like the dotted edges in Fig. 3.5(b-d)). Additional nice consequence besides independence is that the system is easy to solve since orbit counts can be computed from those belonging to graphlets with more edges towards those with less.

When constructing the equations, we choose y that allows for efficient computation of the right-hand sides: we will ensure that the right-hand sides contain only the node degrees and the numbers of common neighbours of pairs and of *connected* triplets $(c(u,v), c(u,v,t))$. This will allow us to precompute and store the values of $c(u,v)$ and $c(u,v,t)$ for all pairs and connected triplets in G before computing the orbit counts for individual nodes.

First, we choose the node y so that the remaining nodes constitute a four-node graphlet, *i.e.* removing y does not break the graphlet into disconnected components, which would require enumeration of disconnected subgraphs. Second, the node y has to have at most three edges to avoid the need to compute the number of common neighbours of four points, $c(u,v,w,t)$. Besides, when y has three neighbours, they need to induce a connected subgraph.

A node y that fulfils these criteria exists for all orbits except O_{72}. Precomputing the values $c(u,v,t)$ for all connected triplets takes $O(e \cdot d^2)$ time, and storing them in a hash table takes $O(e \cdot d)$ space. Computation of the right hand-sides also requires enumerating all the four-nodes graphlets, which again has a complexity of $O(e \cdot d^2)$.

The total time required to compute all orbit counts for all $x \in V$ is then $O(e \cdot d^2 + T_5)$ with $O(e \cdot d)$ space, where $O(T_5)$ is the time required to enumerate all complete 5-node graphlets (G_{29}). The algorithm thus has the same upper bound complexity as the existing algorithms, $O(n \cdot d^4)$. Experiments however show that the bound is not tight: the contribution of the $O(T_5)$ is negligible over the range of sensible graph densities,

Table 3.1

Statistics of benchmark real-world networks.

network	nodes	edges	max. degree
S. cerevisiae	5097	22282	289
E. coli	2984	11626	178
D. melanogaster	7618	22864	178
Human	18170	137775	9716
Internet Autonomous Systems	25368	75004	3781

and the actual running times are smaller by an order of magnitude.

We could use the same technique to construct systems of equations for larger graphlets. Note however that we reduced the running times by imposing some conditions to the selection of the node y. We have not researched whether such nodes also exist for larger graphlets; although theoretically interesting, this may be of little practical use in the context of bioinformatics.

3.4 Results and Discussion

We compared the speed of Orca with RAGE, GraphCrunch and FANMOD. We ran all experiments on a modest desktop computer (Intel Core 2, 2.67 GHz). We have not experimented with parallel execution; all four algorithms allow for trivial distribution of work on multiple cores, so the benefits of parallelization should be the same for all.

We compared the performance of methods on the three largest species-specific PPI networks from the July 2013 update of the Database of Interacting Proteins [38] and the human PPI network from the BioGRID [23] 3.2.104 release. The sizes of individual datasets are presented in Table 3.1.

All algorithms except the significantly slower FANMOD counted orbits for four-node graphlets in the smaller graphs in a few seconds (Table 3.2). Five-node graphlets present a more difficult task: running GraphCrunch on the S. cerevisiae PPI network took more than 9 minutes (as compared to 4.4 seconds for four-node graphlets). FANMOD was almost 10 times slower while Orca finished the same task 80 times faster, in 6.6 seconds. RAGE is limited to four-node graphlets. We got similar results for the other two networks.

In the larger Human network, Orca counted the four-node graphlets 100 and 1800

Table 3.2

Comparison of algorithms on real-world networks. We aborted the algorithms that took more than a day.

	four-node graphlets			
network	FANMOD	GraphCrunch	RAGE	Orca
S. cerevisiae	62 s	4.4 s	1.7 s	< 0.1 s
E. coli	34 s	1.8 s	1.0 s	< 0.1 s
D. melanogaster	21 s	3.1 s	1.6 s	< 0.1 s
Human	/	183 min	11.8 min	6.1 s
Internet Autonomous Systems	574 min	37 min	3.0 min	2.5 s

	five-node graphlets		
network	FANMOD	GraphCrunch	Orca
S. cerevisiae	87 min	9.5 min	6.6 s
E. coli	38 min	4.1 min	4.8 s
D. melanogaster	18 min	2.8 min	2.3 s
Human	/	/	269 min
Internet Autonomous Systems	/	/	49 min

times faster than Rage and GraphCrunch, respectively; we aborted FANMOD after 24 hours. Orca was also the only algorithm capable of counting five-node graphlets in a human PPI network in less than a day.

For comparison with RAGE, we included a test network of Internet Autonomous Systems[1] that was used as the benchmark for RAGE [22]. FANMOD required over 9 hours, GraphCrunch finished in 37 minutes, RAGE in 3 minutes and Orca in 2.5 seconds. Orca finished the computation for five-node graphlets in 49 minutes, while the other two algorithms were stopped after 24 hours.

The time that Orca needs for counting orbits in five-node graphlets are comparable to those that GraphCrunch needs for four-node graphlets. This is consistent with the way the two algorithms are constructed: GraphCrunch enumerates four-node graphlets to count them, while Orca enumerates them to count five-node graphlets. As expected, the time needed for enumeration of complete five-node graphlets is negligible at these network densities.

[1] http://www.netdimes.org/PublicData/csv/ASEdges4_2012.csv.gz

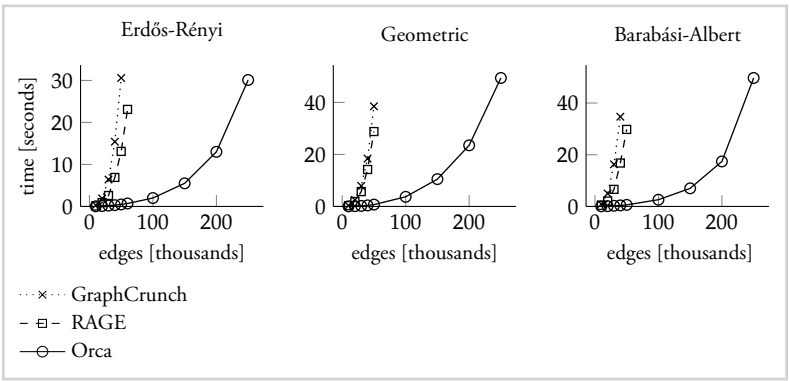


Figure 3.6

Comparison of times needed for counting orbits in four-node graphlets in random networks. Graphs are cut off at one minute; results of experiments in which the methods were allowed to run for up to one hour are available in the supplementary.

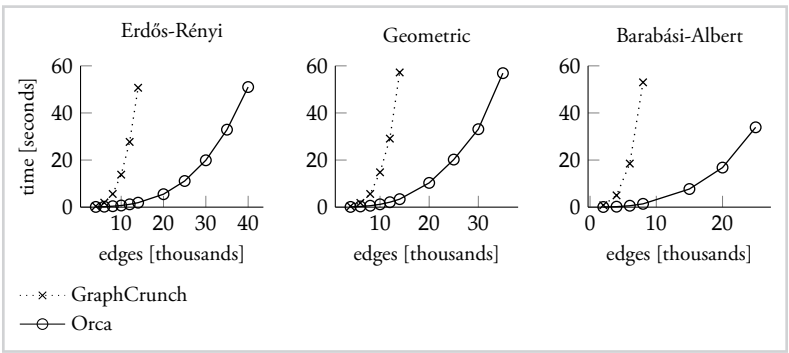


Figure 3.7

Comparison of times needed for counting orbits in five-node graphlets in random networks

For more insight into time complexities of the compared algorithms, we tested them on synthetic data using three different random network models – Erdős-Rényi, geometric and Barabási-Albert random graphs. Erdős-Rényi graphs are constructed by randomly connecting e pairs of nodes. We generated geometric graphs by randomly placing nodes in a 3-dimensional unit cube and connecting the e closest pairs; geometric graphs show largest resemblance to protein interaction networks [1]. Barabási-Albert preferential attachment model generates scale-free networks that exhibit hubs and individual highly connected nodes.

We explored the performance of GraphCrunch, RAGE and Orca at different network densities. FANMOD was not included as it consistently finished previous tests

far behind GraphCrunch. All graphs had 1000 nodes; for each method, we increased the graph density until the method needed more than a minute to complete the test. The corresponding graphs were relatively dense, containing up to 40% of all possible edges for test with four-node graphlets and around 10% for five-node graphlets.

RAGE counted the four-node graphlets slightly faster than GraphCrunch but they were both significantly outperformed by Orca (Fig. 3.6 and Tables 3.3–3.8 in the supplementary). We observed similar results when counting five-node graphlets (Fig. 3.7). Orca achieved the highest gain in comparison with other methods on Barabási-Albert models, in which hubs present a large obstacle for GraphCrunch and RAGE. This makes Orca more suitable for real-world networks, which often display the small-world property and contain hubs.

3.5 Conclusion

Graphlet-based network analysis is useful for various tasks in bioinformatics, such as alignment of PPI networks and prediction of protein functions based on topological similarities. Past studies used these approaches to, for instance, identify genes related to cancer [11] and to aging [12].

We presented a new algorithm for counting graphlet orbits that is based on derived relations between orbit counts. To count the orbits for k-node graphlets, it enumerates $(k-1)$-node graphlets and a single k-node graphlet. Empirical results confirm that this decreases the time complexity by an order of magnitude in comparison with other known methods. In practical terms, the algorithm counts orbits in large PPI networks 50-100 times faster than other state-of-the-art algorithms.

3.6 Supplementary

3.6.1 Results on random networks

Counting 4-node graphlets

Table 3.3

Results of counting 4-node graphlets in random ER networks [s]. Missing values indicate running times over 1 hour.

	edges [thousands]									
	10	20	30	40	50	60	100	150	200	250
GraphCrunch	0.3	1.9	6.4	15.4	30.6	56.0	283	957	2193	/
RAGE	0.1	0.9	2.6	6.9	13.1	23.1	337	1797	/	/
Orca	0.03	0.1	0.2	0.3	0.5	0.7	2.0	5.5	13.0	30.1

Table 3.4

Results of counting 4-node graphlets in random GEO networks [s]. Missing values indicate running times over 1 hour.

	edges [thousands]								
	10	20	30	40	50	100	150	200	250
GraphCrunch	0.3	2.5	7.9	18.4	38.5	319	1003	2272	/
RAGE	0.3	1.9	5.7	14.2	28.8	604	2877	/	/
Orca	0.03	0.1	0.2	0.4	0.7	3.7	10.5	23.5	49.4

Table 3.5

Results of counting 4-node graphlets in random BA networks [s]. Missing values indicate running times over 1 hour.

	edges [thousands]								
	10	20	30	40	50	100	150	200	250
GraphCrunch	0.9	5.0	16.3	34.7	66.0	425	1220	2578	/
RAGE	0.4	2.1	6.6	16.8	29.8	683	2884	/	/
Orca	0.04	0.1	0.25	0.4	0.6	2.6	7.0	17.4	49.7

Counting 5-node graphlets

Table 3.6

Results of counting 5-node graphlets in random ER networks [s].

	edges [thousands]										
	4	6	8	10	12	14	20	25	30	35	40
GraphCrunch	0.4	1.9	5.6	13.8	27.7	50.7	220	530	1098	2025	3553
Orca	0.1	0.2	0.4	0.7	1.2	1.9	5.5	11.1	19.9	32.9	51.0

Table 3.7

Results of counting 5-node graphlets in random GEO networks [s].

	edges [thousands]									
	4	6	8	10	12	14	20	25	30	35
GraphCrunch	0.4	1.8	5.6	14.8	29.1	57.2	260	583	1210	2190
Orca	0.1	0.3	0.6	1.2	2.1	3.4	10.3	20.2	33.1	56.9

Table 3.8

Results of counting 5-node graphlets in random BA networks [s].

	edges [thousands]						
	2	4	6	8	15	20	25
GraphCrunch	0.8	5.0	18.5	53.0	375	912	2058
Orca	0.07	0.2	0.6	1.4	7.7	16.8	33.9

3.6.2 Log-scale graphs

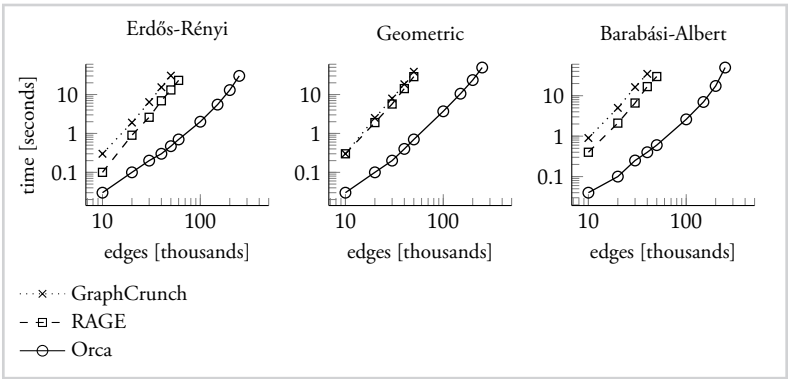


Figure 3.8

Log-scale comparison of times needed for counting orbits of four-node graphlets in random networks.

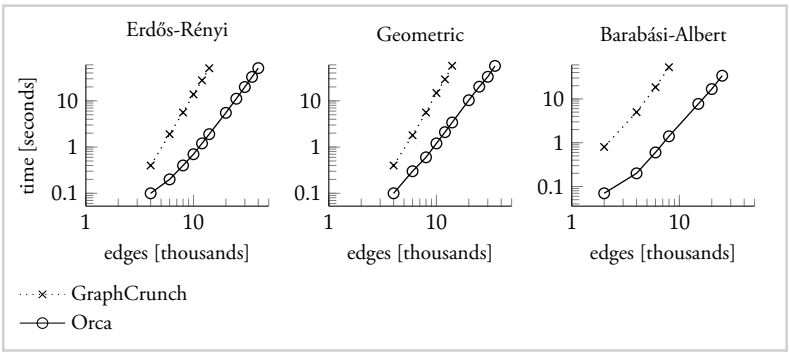


Figure 3.9

Log-scale comparison of times needed for counting orbits of five-node graphlets in random networks.

4

Computation of graphlet orbits for nodes and edges in sparse graphs

Journal of Statistical Software

Computation of Graphlet Orbits for Nodes and Edges in Sparse Graphs

Tomaž Hočevar, Janez Demšar

University of Ljubljana

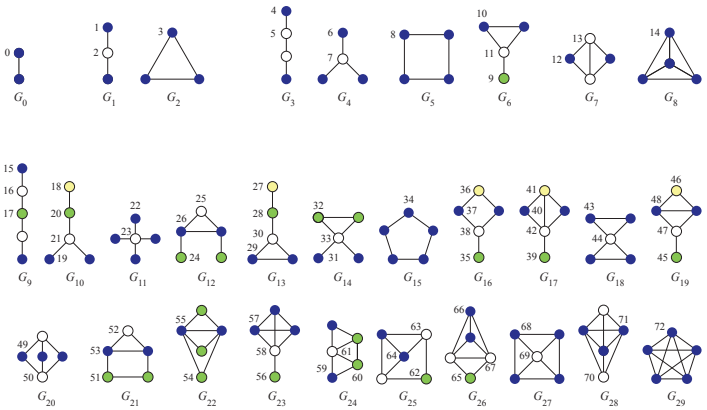
4.1 Abstract

Graphlet analysis is a useful tool for describing local network topology around individual nodes or edges. A node or an edge can be described by a vector containing the counts of different kinds of graphlets (small induced subgraphs) in which it appears, or the "roles" (orbits) it has within these graphlets. We implemented an R package with functions for fast computation of such counts on sparse graphs. Instead of enumerating all induced graphlets, our algorithm is based on the derived relations between the counts, which decreases the time complexity by an order of magnitude in comparison with past approaches.

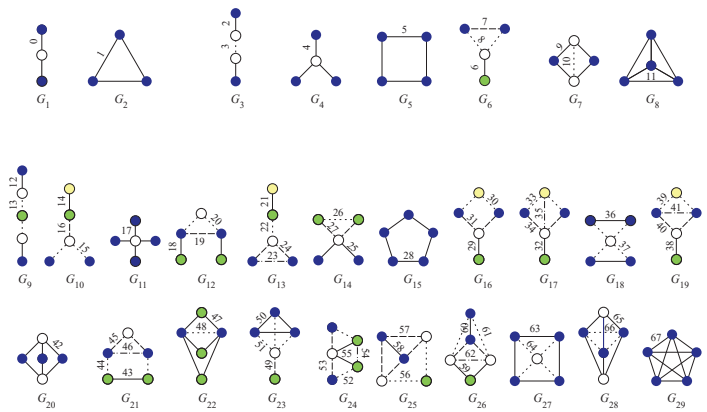
Keywords: network analysis, graphlets, data mining, bioinformatics

4.2 Introduction

Analysis of networks plays a prominent role in many areas of science and business, from genetic and protein networks in bioinformatics to social networks in mining user data. Describing the roles of individual nodes and edges, clustering them, and predicting their future development requires observing their locally defined properties. One of the methods – used particularly in bioinformatics – is based on counting graphlets and graphlet orbits.



(a) Node orbits.



(b) Edge orbits.

Figure 4.1

Graphlets with 2–5 nodes with enumeration of orbits. Node colors and line shapes, which are chosen arbitrarily, correspond to orbits within each graphlet. Node orbits are enumerated as in [2]; edge orbit numbers are enumerated by increasing orbits of the corresponding node pairs.

Graphlets are small connected simple graphs [1]. There are 9 different graphlets with two to four nodes and 30 graphlets with up to five nodes. In graphlet-based network analysis, we examine induced graphlets within the network: for each node, we count

the number of times the node is contained in an induced graphlet of each kind, which gives a 9- or 30-dimensional vector description of the local topology surrounding the observed node. One of the vector components represents, for instance, the number of times the node is included in an induced star on five nodes.

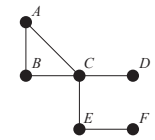
Furthermore, we can group nodes of each graphlet into orbits [2] with respect to the graphlet automorphisms (Figure 4.1(a)). Orbits define the "roles" of the nodes within the graphlet. For instance, in a star on five nodes (G_{11}), one node represents the center and the remaining four nodes are the leaves; the nodes of the star thus form two different orbits (numbered 23 and 22, respectively). Instead of counting only the number of appearances of induced stars that touch an observed node in the network, we can count how many times the node represents the center of such star (*i.e.*, the node is connected to four nodes that are not connected to each other) and how many times it has the role of a leaf (it is connected to a node that is connected to another three nodes that are disconnected from each other and from the observed node). This gives a finer description of the node's vicinity with a 15-dimensional vector for four-node graphlets and a 73-dimensional vector for five node graphlets.

Similar can be done for edges [3]: there are 68 edge orbits for graphlets with 3–5 nodes, which allow for a characterization of an edge with a 68-dimensional vector (Figure 4.1(b)).

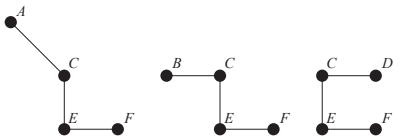
Figure 4.2 gives an illustration for a small network. Figure 4.2(a) shows the network, and figures 4.2(b), 4.2(c) and 4.2(d) show all four-node subgraphs that include node C; node C appears in orbits 5, 7 and 11. Table 4.2(e) shows all orbit counts for node C, including those belonging to two- and tree-node subgraphs. Table 4.2(f) shows the orbit counts for all nodes and orbits. The vector (row) corresponding to node C is quite different from others (*e.g.*, counts for orbits 2, 7, 11), which indicates its special place in the graph. Signatures of A and B, on the other hand, are the same since the two nodes map to each other in an automorphism of the graph.[1]

The straightforward computation of orbit counts by enumeration takes $O(nd^{k-1})$ time, where n is the number of nodes (typically thousands or tens of thousands), d is the maximal node degree (usually up to one hundred), and k is the graphlet size (4 or 5). We have recently presented a combinatorial approach for counting orbits of nodes [33] in time that is, for practical purposes, proportional to nd^{k-2}. Using this

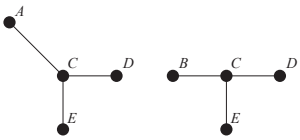
[1] In general, two nodes will have the same signature for k node graphlets if their local neighborhood of up to $k-1$ edges is the same.



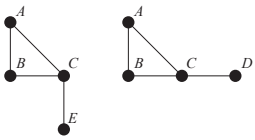
(a) Example network.



(b) Subgraphs with node C in orbit 5.



(c) Subgraphs with node C in orbit 7.



(d) Subgraphs with node C in orbit 11.

Orbit	Count	
0	4	The count of orbit 0 equals the degree of the node
1	1	The terminal node of a path on three nodes ($\{C, E, F\}$)
2	5	The middle node of a path on three nodes ($\{A, C, D\}$, $\{A, C, E\}$, $\{B, C, D\}$, $\{B, C, E\}$, $\{D, C, E\}$)
3	1	Triangle ($\{A, B, C\}$)
5	3	The inner node of a path on four nodes (Fig. b)
7	2	The central node of a 3-edge star (Fig. c)
11	2	The central node of the $L_{3,1}$ lollipop graph (Fig. d)

(e) Non-zero orbit counts for node C.

Orbit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	2	2	0	1	1	0	1	0	0	0	2	0	0	0	0
B	2	2	0	1	1	0	1	0	0	0	2	0	0	0	0
C	4	1	5	1	0	3	0	2	0	0	0	2	0	0	0
D	1	3	0	0	1	0	2	0	0	1	0	0	0	0	0
E	2	3	1	0	0	3	2	0	0	1	0	0	0	0	0
F	1	1	0	0	3	0	0	0	0	0	0	0	0	0	0

(f) Orbit counts for all nodes.

Figure 4.2

Illustration of orbit counts for a simple network.

technique, the common-size networks from proteomics can be analyzed in a reasonable time of a few hours on a common desktop computer. In this paper, we provide the first complete description of the algorithm, including its novel extension to counting edge orbits (Section 2), and then document the corresponding R package together with two usage examples (Section 3).

The notation used throughout the paper is summarized in Table 4.1.

Table 4.1

Notation used in the paper.

$G = (V, E)$	the observed graph with nodes V and edges E
n	number of nodes in the graph
e	number of edges
d	maximal node degree
k	graphlet size
O_i	node orbit i
$o_i(x)$ (or o_i)	the number of times that node x appears in orbit O_i; we use o_i to reduce the clutter where possible
E_i	edge orbit i
$e_i(p)$ (or e_i)	the number of times that edge p appears in orbit E_i
$N(x_1, x_2, \ldots x_i)$	the set of common neighbours of nodes x_1, x_2, …, x_i
$c(x_1, x_2, \ldots x_i)$	the number of common neighbours of nodes x_1, x_2, …, x_i
$G[\{x_1, x_2, \ldots x_i\}]$	a subgraph of G on nodes x_1, x_2, …, x_i
$\cong$	symbol $\cong$ denotes graph isomorphism

4.3 Combinatorial approach to orbit counting

Let $G = (V, E)$ be a simple graph with n vertices (V) and e edges (E). We assume that the graph is sparse ($e = O(n)$). We will denote graphlets as G_i and node orbits as O_i. We follow the enumeration by Przulj [2] (see Figure 4.1(a)), in which the orbit numbers are assigned somewhat arbitrarily but with the constraint that the indices of orbits belonging to the graphlets with fewer edges are smaller than those belonging to the graphlets with more edges. We will use E_i to denote edge orbits, which we enumerate as shown in Figure 4.1(b). Here we decided to ignore the pre-existing

enumeration by Solava et al. [3] and define a more consistent one in which the edge orbits are ordered by the orbits of the corresponding nodes.

The task is to count the number of times a node x appears in each orbit O_i, or the number of times an edge p appears in orbit E_i. We will denote the two numbers by $o_i(x)$ and $e_i(p)$; where possible, we will omit x or p and write only o_i and e_i. The algorithm computes the counts for all graph nodes (or edges). Computation for just a few nodes can be done faster using a brute force approach (exhaustive enumeration).

Past approaches – such as that in *GraphCrunch* [20] and *RAGE* (Rapid graphlet enumerator) [22] – are based on exhaustive enumeration of induced subgraphs.[2] Their theoretical and empirical complexity of enumerating graphlets of size k is $O(nd^{k-1})$, where d is the maximal node degree in the graph. Our approach builds on the work of Kloks et al. [29], who constructed a system of equations for counting induced subgraphs with four-nodes, and Kowaluk et al. [45], who generalized it for larger subgraphs. We use a similar principle to count orbits; besides, our approach scales better for sparse graphs. In comparison with enumeration-based algorithms, the combinatorial approach decreases the practical time complexity by the factor of d by directly enumerating only the graphlets of size $k-1$ and using them to compute the counts for graphlets of size k.

4.3.1 Node orbits

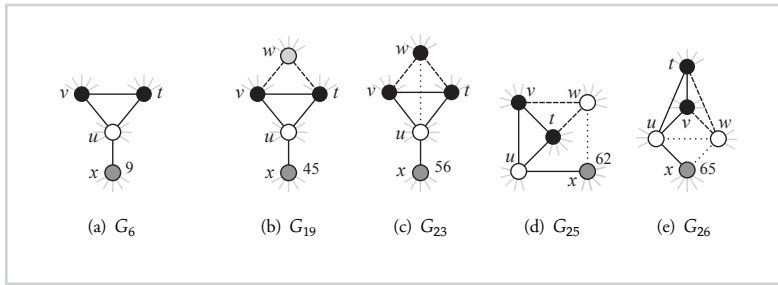
We shall demonstrate the basic idea with an example.

Let x be a node in the graph G. $o_{45}(x)$ represents the number of times x appears in orbit O_{45}, that is, the number of ways in which G_{19} can be embedded in G so that x is in orbit O_{45}. To reduce the clutter, we shall omit x and denote this by o_{45}. Counts o_{56}, o_{62} and o_{65} are defined similarly. We will show that the following relation holds for any x:

$$o_{45} + 3o_{56} + 2o_{62} + 2o_{65} = \sum_{\substack{u,v,t:\ G[\{x,u,v,t\}]\cong G_9 \\ v<t\ \wedge\ v,t\notin N(x)}} (c(v,t) - 1)\,, \tag{4.1}$$

where u, v and t are triplets of nodes that fulfil certain conditions (details are explained below) and $c(v,t)$ is the number of common neighbours of v and t. The

[2] Recent versions of *GraphCrunch* also already include a part of our approach described here.



(a) G_6 (b) G_{19} (c) G_{23} (d) G_{25} (e) G_{26}

Figure 4.3

Derivation of the relation between o_{45}, o_{56}, o_{62} and o_{65}. Solid lines belong to graphlet G_6, dashed lines represent the required edges (as described in Section 4.3.3) and dotted lines represent additional edges whose presence or absence determines the orbit of the node x. Gray lines represent edges to other nodes of G.

left-hand side of the equation is a linear combination of orbit counts that we wish to compute and the right-hand side is a statistics that is easy to obtain.

Equation 4.1 can be constructed as follows.[3] Let the subgraph on some nodes x, u, v and t be isomorphic to G_6 with x in O_9 (Figure 4.3(a)). Now we observe the possible extensions of $G[\{x,u,v,t\}]$ with a node $w \in V$ that is attached to v and t. For each such $w \in N(v,t)$, the subgraph on x,u,v,t,w is isomorphic

- to G_{19}, if w is not connected to x and u (Fig. 4.3(b)), or
- to G_{23}, if w is connected to u, but not to x (Fig. 4.3(c)), or
- to G_{25}, if w is connected to x, but not to u (Fig. 4.3(d)), or
- to G_{26}, if w is connected to x and u (Fig. 4.3(e)).

This puts x in orbits O_{45}, O_{56}, O_{62} or O_{65}, respectively. Therefore,

$$o'_{45} + o'_{56} + o'_{62} + o'_{65} = |N(v,t)| - 1 = c(v,t) - 1, \tag{4.2}$$

where o'_i represent orbit counts considering only these particular nodes (and annotations) u, v, t, and all common neighbours of v and t, $N(v,t)$. The term -1 is needed since one of the members of $N(v,t)$ is also u.

Equation 4.1 relates the total orbit counts o_{45}, o_{56}, o_{62} and o_{65} for a fixed node x. We construct it by summing the right-hand side of (4.2), $c(v,t) - 1$, over all triplets

[3] This description is intended to present the reasoning behind the relations, while the actual construction was slightly different in order to obtain a useful system of equations. Details are given in Section 4.3.3.

$\{u, v, t\} \subseteq V$ such that $G[\{x, u, v, t\}] \cong G_9$ and $v, t \notin N(x)$ (to put x in O_9 within this subgraph) and with $v < t$ under some arbitrary ordering of nodes, that is,

$$\sum_{\substack{u,v,t:\ G[\{x,u,v,t\}]\cong G_9 \\ v<t\ \wedge\ v,t\notin N(x)}} (c(v,t) - 1)\,. \tag{4.3}$$

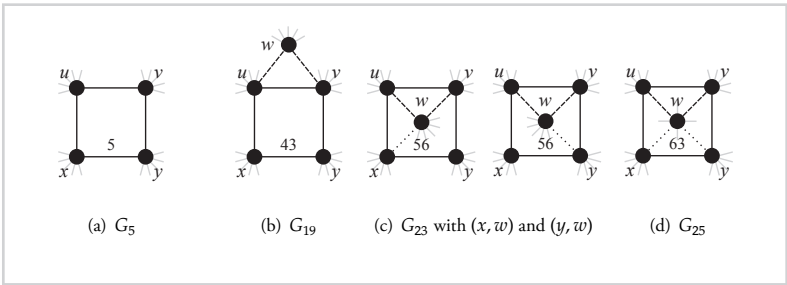
Despite the condition $v < t$, some subgraphs are counted multiple times.

- Each subgraph with x in O_{56} ($G[\{x, u, v, t, w\}] \cong G_{23}$, Figure 4.3(c)) is counted thrice: the nodes x and u are fixed while v, t and w are exchanging their roles in three possible permutations (the condition $v < t$ prohibits the other three out of the six possible permutations).
- If x belongs to O_{62} ($G[\{x, u, v, t, w\}] \cong G_{25}$, Figure 4.3(d)), the subgraph on quintuplet $\{x, u, v, t, w\}$ is counted twice, with exchanged roles of u and w; the nodes v and t are fixed due to $v < t$.
- The configuration in which x is in O_{65} ($G[\{x, u, v, t, w\}] \cong G_{26}$, Figure 4.3(e)) is similar to that of O_{62}.
- The configuration for x in O_{45} ($G[\{x, u, v, t, w\}] \cong G_{19}$, Figure 4.3(b)) is unique: for quintuplet $\{x, u, v, t, w\}$ in $x \in O_{45}$, the conditions $v < t$ and $v, t \notin N(x)$ allow for only one possible annotation of the nodes.

After accounting for these multiple counts of the same orbit when summing (4.2) over all applicable triplets u, v, t (as in (4.3)), we get Equation 4.1. To evaluate such equations we need to precompute values $c(v, t)$ and sum them over four-node induced subgraphs of the network. Both of these steps require an enumeration of all four-node induced subgraphs, which is the bottleneck of the method. Because every enumerated four-node subgraph will contribute to the sum on the right side of one or more equations, we can optimize the code and avoid explicitly checking the summation conditions in the equations because they are already included in the enumeration process (see the code snippet in Section 4.3.4 for illustration).

Certain relations involve more complicated symmetries, for instance

$$o_{37} + 2o_{68} + 2o_{64} + 2o_{63} + 4o_{62} + o_{53} + o_{51} + 4o_{49} = \sum_{\substack{u,v,t:\ G[\{x,u,v,t\}]\cong G_5 \\ u<v\ \wedge\ u,v\in N(x)}} (c(u) + c(v) - 4)\,. \tag{4.4}$$



(a) G_5 (b) G_{19} (c) G_{23} with (x, w) and (y, w) (d) G_{25}

Figure 4.4

Derivation of the relation between e_{43}, e_{56} and e_{63}. Solid lines belong to graphlet G_5, dashed lines represent the required edges (since w is defined to span over the edge (u, v)) and dotted lines represent additional edges whose presence or absence determines the orbit of the edge (x, y). Gray lines represent edges to other nodes of G.

The sum runs over all induced subgraphs in G that put node x in orbit O_8 in G_5. Nodes u and v are its neighbours and t is the node opposite of x in G_5. We obtain the same graphlet if we attach a new node w either to u or to v, and there are $c(u) - 2$ and $c(v) - 2$ such possibilities. There are also three optional edges (to x, y and u or v), which decide the orbit of x in the extended graphlet; the resulting orbit can be O_{37}, O_{68}, O_{64}, O_{63}, O_{62}, O_{53}, O_{51} or O_{49}.

4.3.2 Edge orbits

Relations for edge orbits are derived in the same way. For instance, let (x, y) represent an edge of a square (graphlet G_5). Let us label the remaining two nodes with $u \in N(x) \backslash \{y\}$ and $v \in N(y) \backslash \{x\}$ (Figure 4.4(a)). Extending this pattern with a node w that spans over the edge (u, v) leads to three possible graphlets and hence three different edge orbits of the edge (x, y) (Figure 4.4 (b-d)).

Orbit E_{56} arises when w is adjacent to either x or y, while the other two orbits, E_{43} and E_{63} can only arise in one way. Hence the relation between the orbits is

$$e_{43} + 2e_{56} + e_{63} = \sum_{\substack{u,v:\, G[\{x,y,u,v\}] \cong G_5 \\ (x,y) \in E \,\wedge\, u \in N(x) \,\wedge\, v \in N(y)}} c(u, v). \tag{4.5}$$

4.3.3 System of equations

We constructed the equations similar to those above to relate each orbit with orbits from graphlets with a larger number of edges. Complete lists of equations are provided in the appendices.

For instance, Equation 4.1 was constructed specifically to relate o_{45} with higher orbits. The actual construction of equation goes in the opposite direction from that presented in the introductory example. We started with the graphlet G_{19}, to which the orbit O_{45} belongs and picked one of the nodes (labelled w in the above case). We assumed that w is adjacent to v and t, and examined the graphlets in which w may be also adjacent to u and/or x. This ensures that the equation that is set up with O_{45} in mind relates O_{45} with orbits with higher indices since these graphlets have more edges than G_{19}. As a consequence, the resulting system of equations is triangular, and thus independent and also easy to solve by going backwards from the higher orbits (starting with O_{14} or O_{72}, which belong to complete graphs) towards lower orbits.

We impose several constraints on selection of w. Node w can not coincide with x, or with x or y when computing orbits of edge (x, y). We further require that removal of w does not break the remaining nodes into disconnected subgraphs. Node w must have at most $k-2$ neighbours; when it does have $k-2$ neighbours, they must be connected. This allows for more time- and space-efficient computations of orbits, as described in Section 4.3.4. Existence of such nodes for each orbit of four- and five-node graphlets can be proven by exhaustive search, with exception of G_5, which is handled as a special case.

All equations have the following general form:

$$a_1 o_{i_1} + a_2 o_{i_2} + \ldots + a_t o_{i_t} = \sum_{\substack{S : G[S] \cong G_j \\ x \in S \ \wedge \ \mathrm{cond}(S)}} (c(S_1) + c(S_2) + \ldots + c(S_u) + C)\,, \quad (4.6)$$

where $\mathrm{cond}(S)$ is a set of conditions that constrain the embedding of $G[S]$ into G and assign labels to nodes. For instance, in Equation 4.1, condition $v, t \notin N(x)$ assigns the labels v and t to the nodes in orbit O_{10} and $v < t$ ensures that the same quadruplet of nodes is not counted twice.

The sum runs over subgraphs $G[S]$ isomorphic to some graphlet G_j on $k-1$ nodes, that is, over some three-node graphlet when computing the orbits in four-node graphlets, or over some four-node graphlet when computing orbits in five-node graphlets. The subgraph must include x, and the conditions in the sum put x into some fixed orbit. Additional conditions may impose ordering on the remaining nodes of the graphlet to decrease the number of symmetries.

The terms in the sum are the number of common neighbours of some subsets of

nodes in the subgraph ($S_k \subset S$). The number of such terms is between 1 and 3. The size of S_k is also between 1–3, that is, the terms refer to node degrees and to the number of common neighbours of pairs and triplets of nodes. The criteria for the choice of node w, which are described above, ensure that these terms can be efficiently computed using some precomputed data as described in the following section.

The left-hand side is a fixed linear combination of orbits to which the node x evolves after extending G_j with another node connected to one of subsets S_k. The coefficients reflect the symmetries in the graphlets with regard to node assignments.

We prepared a system of 10 equations that relate the 11 node orbits of four-node graphlets, and a system of 57 equations that relate the 58 node orbits of the five-node graphlets. Likewise, we have constructed 9 equations that relate the 10 edge orbits for four-node graphlets and 55 equations for 56 edge orbits on five-node graphlets. By selecting different nodes w, we have empirically verified that it is impossible to construct a full-rank system using our approach and the constraints we put on w.

Due to the rank's deficiency, one of the orbits must be enumerated directly. The most suitable candidates are the orbits belonging to complete graphlets (O_{14} and O_{72} for nodes, and E_{12} and E_{68} for edges). First, this allows for a straightforward computation of the orbits since the system is triangular so that lower orbits are computed from the higher. Second, since we assume that the graphs are sparse, we can efficiently compute these orbits by using an enumeration method similar to the Bron-Kerbosch maximal clique enumeration algorithm [39].

4.3.4 *Algorithm*

The algorithm consists of precomputation of some data, followed by computation of orbit counts for each node or edge.

1. Precomputation:
   - Count the complete graphlets touched by each node or edge.
   - Count the common neighbours of every pair and of every three vertices that form a connected graph.
2. For each node or edge:
   - Compute the right-hand sides by enumeration of $k - 1$ node graphs using the precomputed data above.

- Solve the system of linear equations.

Our implementation of the algorithm represents the graph with adjacency and incidence lists, which are appropriate for sparse graphs. If the graph has less than 30000 nodes, we also construct an adjacency matrix. The matrix, which uses 1 bit per edge and takes at most around 100 MB, allows us to check for existence of edges between any given pair of nodes in constant time. Without it, the time complexity of the lookup for an edge between two nodes is proportional to the logarithm of the number of neighbours of the node with smaller degree.

In the following, we will describe each step in more detail.

1. Precomputation:

   - For each node, count the number of complete graphlets in which the node or edge participates. We build cliques of size k from cliques of size $k-1$ by maintaining a set of candidate nodes that are adjacent to all nodes in the smaller clique. This procedure is similar to the Bron-Kerbosch algorithm with the difference that we are not interested in maximal cliques but in all cliques of a given size.

     Although the theoretical upper bound of the time complexity of this step is $O(ed^{k-2})$, where d is the maximal node degree, the actual contribution of this step to the total running time is negligible since complete subgraphs (cliques) in sparse networks are rare.

   - Compute and store the number of common neighbours for each pair of adjacent vertices. This takes $O(ed)$ time and $O(e)$ space. For computing the orbits in five-node graphlet, we also compute the number of paths of length 2 between each pair of nodes for which such a path exists, and the number of common neighbours for all triplets of connected nodes. This takes $O(ed^2)$ time and $O(ed)$ space.

2. For each node or edge:

   - Compute the right-hand sides of the system of the linear equations. Its general form is shown in Equation 4.6. For four-node graphlets, the sums run over three-node paths or triangles in which the node appears. For five-node graphlets, they run over four-node graphlets that the node touches.

Right-hand sides of equations that sum over the same graphlet can be computed simultaneously. The following code chunk illustrates the computation of the right-hand sides of equations for orbits 13, 16 and 20 for an edge (x, y),

$$\begin{aligned} e_{13} + 2e_{22} + 2e_{28} + e_{31} + \\ e_{40} + 2e_{44} + 2e_{54} = \sum_{\substack{a,b:\ G[\{x,y,a,b\}]\cong G_3 \\ a\in N(x)\ \wedge\ b\in N(y)}} (c(a) + c(b) - 2), \end{aligned}$$

$$\begin{aligned} 2e_{16} + 2e_{20} + 2e_{22} + e_{31} + \\ 2e_{40} + e_{44} + 2e_{54} = \sum_{\substack{a,b:\ G[\{x,y,a,b\}]\cong G_3 \\ a\in N(x)\ \wedge\ b\in N(y)}} (c(x) + c(y) - 4), \end{aligned}$$

$$e_{20} + e_{40} + e_{54} = \sum_{\substack{a,b:\ G[\{x,y,a,b\}]\cong G_3 \\ a\in N(x)\ \wedge\ b\in N(y)}} c(x, y).$$

The code for computation of the right-hand sides is as follows.

```
for (int nx = 0; nx < deg[x]; nx++) {
    int const &a = adj[x][nx];
    if (a == y || adjacent(y, a))
        continue;
    for (int ny = 0; ny < deg[y]; ny++) {
        int const &b = adj[y][ny];
        if (b == x || adjacent(x,b) || adjacent(a,b))
            continue;
        EORBIT(3)++;
        f_13 += (deg[a] - 1) + (deg[b] - 1);
        f_16 += (deg[x] - 2) + (deg[y] - 2);
        f_20 += tri[xy];
    }
}
```

Here, `deg[x]` and `deg[y]` are degrees of nodes x and y, and `adj[x]` and `adj[y]` are arrays with indices of their neighbours. Function `adjacent(u,`

`t)` checks whether nodes u and t are adjacent (with a time complexity $O(1)$ or $O(\log d)$, depending on whether we construct an adjacency matrix or not) and `tri[xy]` is the number of triangles spanning over the edge between x and y. Variables `f_13`, `f_16` and `f_20` contain the right-hand sides of equations for O_{13}, O_{16} and O_{20}.

The `if`-clauses check that the edge belongs to E_3 and impose additional constraints as needed. The computation is sped up by using the precomputed data from the first two steps. In the above case, the right-hand sides of equations for orbits 13, 16 and 20 are $(c(a) - 1) + (c(b) - 1)$, $(c(x) - 2) + (c(y) - 2)$ and $c(x, y)$, respectively. The former two are trivial to compute from the graph, and the latter is precomputed in the second step above.

Note that the orbits for $k - 1$-node graphlets (as the orbit 3, above) are computed directly.

The time complexity of this step is $O(ed^{k-3})$.

- Solve the system of equations to obtain orbit counts. Since the system is triangular and the coefficients are fixed, this does not require decomposing or inverting a matrix; the orbits are computed in order, from the higher towards the lower indices, starting with the orbit belonging to the complete graphlet, as for instance, in the following code snippet from the computation of edge orbits.

```
EORBIT(67) = C5[e];
EORBIT(66) = (f_66 - 6 * EORBIT(67)) / 2;
EORBIT(65) = (f_65 - 6 * EORBIT(67));
EORBIT(64) = (f_64 - 2 * EORBIT(66));
EORBIT(63) = (f_63 - 2 * EORBIT(65)) / 2;
EORBIT(62) = (f_62 - 2 * EORBIT(66) - 3 * EORBIT(67));
EORBIT(61) = (f_61 - 2 * EORBIT(65) - 4 * EORBIT(66)
                   - 12 * EORBIT(67));
EORBIT(60) = (f_60 - 1 * EORBIT(65) - 3 * EORBIT(67));
```

The system of equations is also rather sparse, with each equation having at most (but usually much less than) eight variables. These nice properties

– sparse triangular shape – do not make the algorithm faster since the coefficients are fixed. Even a more general matrix could be inverted in advance and hard-coded into the program. The advantage of triangularity, besides the interpretability of the program, is the numerical accuracy since the entire computation stays in the realm of whole numbers.[4]

The system is solved once for each node (or edge), so the time complexity is $O(n)$ ($O(e)$ for edge-orbits).

The total time complexity for all four steps is $O(ed^{k-2} + ed^{k-3} + ed^{k-3} + n)$ for nodes and $O(ed^{k-2} + ed^{k-3} + ed^{k-3} + e)$ for edges. The theoretical complexity is thus $O(ed^{k-2})$, which is the same as for direct enumeration algorithms. Since large networks are typically sparse, the actual contribution of the first term, which comes from enumerating the cliques with k nodes, is negligible in practice. Empirical measurements indeed show that the time complexity is proportional to ed^{k-3}, that is, $O(ed)$ for four-node graphlets and $O(ed^2)$ for five-node graphlets.

4.4 *The* orca *package*

Package *orca* (Orbit counter) is written mostly in C++, with coercion and wrapper functions in R. The package requires R version 2.15 or higher. Due to using the C++ standard 11, which is not available on all platforms, CRAN only hosts the package for R 3.1. Packages for R 2.15 and binaries for OS X and MS Windows are available on the supplement page (http://www.biolab.si/supp/Rorca/).

4.4.1 *Functions*

The package provides four functions: `count4` and `count5` count the node orbits of graphlets on up to four and up to five nodes, and `ecount4` and `ecount5` count the edge orbits. All functions accept a single argument, a graph stored in

- a graph object from the *graph* package;
- an $e \times 2$ edge matrix in which each row contains a pair of nodes given by one-based integer indices; or

[4] The implementation of the library for R uses 64-bit integers internally, but returns a matrix of double precision numbers since some orbit counts for larger graphs do not fit into 32-bit integers used in R.

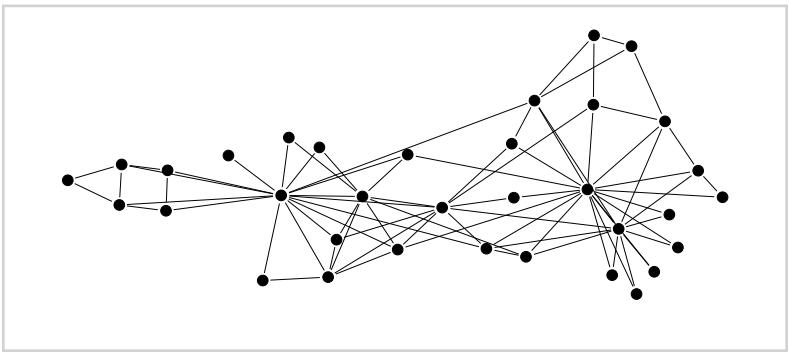


Figure 4.5
Karate club network.

- a data frame in the same format.

Functions return a numeric matrix with rows corresponding to graph nodes or edges, and the columns corresponding to orbits, with column 1 corresponding to orbit 0, column 2 to orbit 1 and so forth.[5]

We will show the package usage on the Karate club network [46], which is included in the package. The network is visualized in Figure 4.5.

```
R> library("orca")
R> data("karate")
R> dim(karate)

[1] 78  2

R> max(karate)

[1] 34
```

The network has 78 edges (the number of rows of the matrix) and 34 nodes (the maximal node index in the matrix).

[5] In the paper we adhere to the traditional numbering of orbits, which starts with 0, to avoid confusion. In practice, the numbering seldom matters since we typically observe the differences between orbit signatures of nodes and edges, in which we consider the whole vectors and not individual orbits.

The result of `count4`, which counts node orbits for graphlets with up to four nodes, has 34 rows (the number of nodes) and 15 columns (the number of orbits).

```
R> orbits <- count4(karate)
R> dim(orbits)

[1] 34 15
```

The first four orbits correspond to three-node graphlets. Here are the orbit counts of four-node graphlets for the first four nodes.

```
R> orbits[1:4, 5:15]

      O4   O5   O6   O7   O8   O9  O10  O11  O12  O13  O14
[1,]  81  197   13  352   10    6   34  171    2   30    7
[2,]  73   56   33   32    6    8   80   27    2   18    7
[3,]  72  179   84   54   20   17   75   51    6    8    7
[4,]  49   11   56    1    0    5   81    5    4    7    7
```

Note that for such small networks a visualization reveals more than orbit counts. Orbit counts become useful on large networks, which are difficult to plot out.

4.4.2 Usage example on the Schools Wikipedia network

Thiel and Berthold [47] argue that in exploring networks we are not necessarily interested in nodes that are closely positioned to the query node (spatial similarity), but also in nodes that have a similar neighborhood structure (structural similarity). They proposed activation spreading signature as a topological description of the local neighbourhood of graph vertices and demonstrate its use on the Schools Wikipedia network. We conducted a similar experiment by using orbit counts instead of activation spreading for the signature.

We downloaded the 2013 edition of Schools Wikipedia[6] and extracted the network of internal links.[7] We computed the orbits for four-node graphlets and found the

[6] http://schools-wikipedia.org/

[7] The network is available for download at http://www.biolab.si/supp/Rorca/.

nearest neighbours (in terms of Euclidean distances between orbit counts) for a few nodes.

Computation of orbits for 4-node graphlets takes 6.1 seconds on a desktop computer. In comparison, *GraphCrunch* as currently the fastest pure enumeration approach[8] takes 13.8 minutes. Computation of 5-node orbits takes 115 minutes; *GraphCrunch* needs 249 hours. This represents a speed-up by a factor of about 130.

```
R> library("orca")
R> library("FNN")

R> nodes <- scan("schools-wiki-nodes.txt", what = "", sep = "\n")
R> edges <- read.table("schools-wiki-edges.txt")

R> orbits <- count4(edges)
R> nn <- get.knn(orbits, k = 10)
R> neighbours <- nn$nn.index
R> distances <- nn$nn.dist

R> check <- c("Canada", "Germany", "Isaac Newton",
+    "Albert Einstein", "Mahatma Gandhi", "Mahabharata")
R> node_indices <- match(check, nodes)
R> for (i in 1:length(check)) {
+    cat("\n\n", check[i], ": ", sep = "")
+    s <- mapply(function(x, y) sprintf("%s (%i)", x, y),
+      nodes[neighbours[node_indices[i], ]],
+      round(distances[node_indices[i], ] / 1000))
+    cat(s, sep = ", ")
+  }
```

After the nodes are described by orbit counts, we find the ten most similar nodes (as defined by Euclidean distance, for the sake of simplicity) to several selected nodes. Results, together with distances divided by 1000, are provided below.

The results for *Canada* and *Germany* are impressive. The two nodes have similar orbit counts – and thus similar role in the local network topology – as nodes *Japan*, *Italy*, *Russia* and other nodes representing countries, cities and regions. This would indicate that it is possible to recognize the nodes corresponding to countries based on the local network structure represented by orbit counts.

[8] Newer versions of *GraphCrunch* also already include some parts of the algorithm described here.

The node orbits – and thus the structure of the network around them – for *Isaac Newton* and *Albert Einstein* are also similar to those of other nodes related to physics. The inclusion of *World War II* in the nodes similar to *Germany*, and the *Church of England* with *Isaac Newton* may, however, be just an instance of the Texas sharpshooter phenomenon. Results for *Mahatma Gandhi* and *Mahabharata* are considerably less satisfactory as these two nodes are connected to unrelated nodes.

```
Canada: Japan (3548), Italy (4224), Russia (6962), Africa (15546),
Spain (15963), London (18186), Australia (18356), Latin (19360),
China (20146), 19th century (26147)

Germany: India (3384), World War II (7343), China (16753),
Australia (18652), London (19340), Italy (32870), Europe (35056),
Canada (36875), Japan (40001), Russia (43656)

Isaac Newton: Temperature (252), Church of England (297), Jupiter (420),
University of Cambridge (432), Planet (441), Science (518),
Albert Einstein (519), Evolution (545), Elephant (548), Insect (597)

Albert Einstein: Science (165), Climate change (249),
Charles Darwin (267), Jupiter (332), Celsius (366),
United Kingdom of Great Britain and Ireland (415),
Church of England (435), United States Congress (452), Black Sea (471),
Civilization (501)

Mahatma Gandhi: Oil refinery (97), Friedrich Engels (98),
Oil shale (100), Impressionism (103), Rugby league (103),
Tropic of Cancer (111), Reggae (111),
Non-governmental organization (125), John Maynard Keynes (128),
John Stuart Mill (135)

Mahabharata: Feather (38), Shiva (38), Guitar (62), John Vanbrugh (66),
Fever (66), Introduction to evolution (68), Henry IV of England (69),
Microscope (69), René Descartes (71), 1754 (72)
```

Exploring why the topology around the nodes is similar in one case and not in another is beyond the scope of this paper.[9] While this example provides an alternative take at the problem explored by Thiel and Berthold [47], graphlet analysis is most often

[9] We speculate that nodes that represent entities of the same kind, such as countries, are connected, thus

used in bioinformatics, where orbit counts are assumed to reflect the roles of genes or proteins in the observed networks. An interested reader may find further examples in the cited works of Pržulj and Milenković.

4.5 Conclusion

We presented a new package *orca* for computing the graphlet orbit counts for nodes and edges. This paper provides the first complete description of the underlying algorithm, which runs much faster than the previous approaches; a more detailed comparison is available in [33]. The novel contribution of the paper is also the generalization of the method to counting the orbits for edges. The package is available on the CRAN repository under the GPL-3 license.

4.6 Acknowledgments

This work has been funded by the Slovenian research agency grants J2-5480 and P2-0209.

the local topology is similar because the nodes are actually close to each other. Yet this does not explain why the method fails for Mahatma Gandhi and Mahabharata for which it found some very similar but unrelated neighbours. A better explanation might require investigating how this network has been constructed.

5

Combinatorial algorithm for counting small induced graphs and orbits

PLOS ONE

Combinatorial Algorithm for Counting Small Induced Graphs and Orbits

Tomaž Hočevar, Janez Demšar

Faculty of Computer and Information Science, University of Ljubljana, Ljubljana, Slovenia

5.1 Abstract

Graphlet analysis is an approach to network analysis that is particularly popular in bioinformatics. We show how to set up a system of linear equations that relate the orbit counts and can be used in an algorithm that is significantly faster than the existing approaches based on direct enumeration of graphlets. The approach presented in this paper presents a generalization of the currently fastest method for counting 5-node graphlets in bioinformatics. The algorithm requires existence of a vertex with certain properties; we show that such vertex exists for graphlets of arbitrary size, except for complete graphs and a cycle with four nodes, which are treated separately. Empirical analysis of running time agrees with the theoretical results.

5.2 Introduction

Analysis of networks plays a prominent role in various fields, from learning patterns [48] and predicting new links in social networks [49, 50], inferring gene functions from protein-protein interaction networks [10] in bioinformatics, to predicting various properties of chemical compounds (mutagenicity, boiling point, anti-cancer activity) [51] from their molecular structure in chemoinformatics. Many methods rely on the

concept of node similarity, which is typically defined in a local sense, *e.g.* two nodes are similar if they share a large number of neighbours. Such definitions are insufficient for detecting the role of the node. A typical social structure includes hubs, followers, adversaries and intermediaries between groups. While local similarity definitions treat the hub and its adjacent nodes as similar, a role-based similarity would consider the hubs as similar disregarding their distance in the graph.

A popular approach in bioinformatics extracts the node's local topology by counting the small connected induced subgraphs (called *graphlets*) [1], which the node touches, and, when a more detailed picture is required, the node's position (*orbit*) [2] in those graphs. See the following paragraphs for a more formal definition. Fig 5.1 illustrates all four-node graphlets and orbits of their nodes. Most applications of graphlet and orbit counts are based on the assumption that the node's local network topology is somehow related to the functionality or some other property of the observed node in the network. Therefore, we can assume that nodes with similar signatures will have similar observed properties. This is the foundation for methods such as clustering of nodes, inference of certain node's properties etc. The nodes often correspond to proteins in the protein-protein interaction networks. However, the networks can model an arbitrary process. With the development of new technologies, these networks are becoming larger, which motivates the development of efficient subgraph counting algorithms.

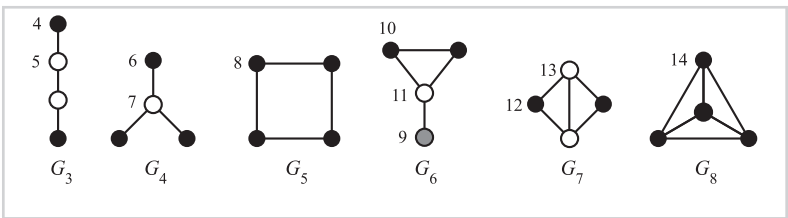


Figure 5.1

Four-node graphlets ($\mathcal{G}_4$). Vertices marked by the same color belong to the same orbit within a graphlet.

Let $\mathcal{G}_k$ be a set of all non-isomorphic connected simple graphs (graphlets) on k nodes, and let $G \in \mathcal{G}_k$. The orbit of a vertex $v \in G$ is a set of all vertices $a(v)$, $a \in Aut(G)$. Let $\mathcal{O}_k$ be a set of orbits for all $v \in G$ and for all $G \in \mathcal{G}_k$. Pržulj [2] numbered the 30 graphs in $\mathcal{G}_2$, $\mathcal{G}_3$, $\mathcal{G}_4$ and $\mathcal{G}_5$ and the corresponding 73 orbits; we will use her enumeration in the examples in this paper.

Let $H = (V, E)$ be the host graph (network) and let $x \in H$. Vertex x participates in

a number of subgraphs $G \in \mathscr{G}_k$ induced in H, in which it appears in different orbits $O_i \in \mathscr{O}_k$. Let o_i be the number of times x appears in orbit O_i in induced subgraphs from $\mathscr{G}_k$.

An example is shown in Fig 5.2. The orbit count o_{17} of vertex x is 9 since x appears in nine paths G_9 as the central vertex (note that the paths must be induced). Other orbit counts for G_9, o_{15} and o_{16}, are 0 and 4, respectively: x does not appear as the end vertex (O_{15}) of G_9 in H, but it appears four times in the role of the node between the center and the end (O_{16}). For a few more examples, $o_{44} = 1$, $o_{47} = 4$, and $o_{59} = 2$; all other orbit counts of 4-node graphlets are 0.

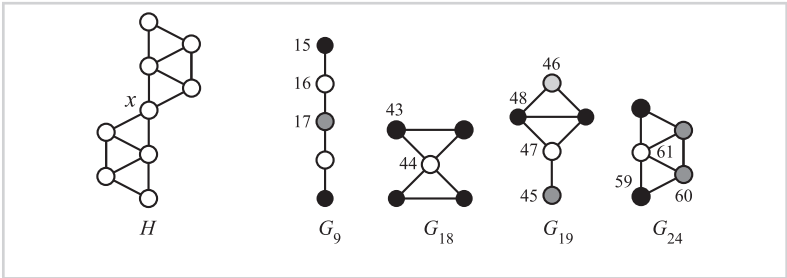


Figure 5.2

A host graph H and graphs G_9, G_{18}, G_{19} and G_{24} from $\mathscr{G}_5$. Graphs and orbits are numbered as in [2].

The *orbit count distribution* is a $|\mathscr{O}_k|$-dimensional vector of o_i for all $O_i \in \mathscr{O}_k$. The orbit count distribution represents a signature of the node: it contains a description of the node's neighborhood and the node's position ("role") within it. As such, this distribution is a useful feature vector for various network analysis tasks.

We will describe an algorithm for computation of orbit count distributions for all vertices $x \in V$ for subgraphs of arbitrary size k. The efficient implementation of the algorithm requires setting up a system of equations that relate subgraph nodes with specific properties; we will prove that such nodes exist for all $k \geq 3$ except for complete graphs and the cycle on 4 nodes, which can be treated specifically. We will show – both theoretically as well as empirically – that the algorithm's time complexity on sparse graphs is lower by an order of magnitude in comparison with enumeration-based approaches.

5.2.1 Preliminaries

Referring to graphlets, orbits, neighbours, etc., requires some notation which we summarize in Table 5.1 and use throughout this paper.

Table 5.1

Notation.

$H = (V, E)$	host graph within which we count the graphlets and orbits
n	number of nodes of H; $n = \|V\|$
e	number of H's edges; $e = \|E\|$
$d(v)$	degree of node v
d	maximal node degree in H; $d = \max_{v \in V} d(v)$
$N(v)$	set of neighbours of vertex $v \in V$
$N(v_1, v_2, \dots, v_j)$	set of common neighbours of $v_1, v_2, \dots, v_j$; $N(v_1, v_2, \dots, v_j) = N(v_1) \cap N(v_2) \cap \dots \cap N(v_j)$
$N(\mathcal{S})$	common neighbours of nodes in the set $\mathcal{S} \subset V$; $N(\mathcal{S}) = \cap_{v \in \mathcal{S}} N(v)$
$c(v)$, $c(v_1, v_2, \dots, v_j)$, $c(\mathcal{S})$	number of common neighbours of vertex v, of vertices $v_1, v_2, \dots, v_j$, and of vertices from set $\mathcal{S}$, respectively; that is, $c(v) = \|N(v)\|$, $c(v_1, v_2, \dots, v_j) = \|N(v_1, v_2, \dots, v_j)\|$, $c(\mathcal{S}) = \|N(\mathcal{S})\|$
$\mathcal{G}_k$	set of all graphlets with k nodes
G_a	graphlet a, according to some enumeration
O_i	orbit i, according to some enumeration
$o_i(v), o_i$	the number of times the node v appears in an induced subgraph in orbit i; since v will be obvious, we will use the shorter notation o_i
$m(i)$	index of the graphlet containing the orbit O_i, *e.g.* $m(16) = 9$

Let $K = (V_K, E_K)$ be a subgraph of $J = (V_J, E_J)$, and let $v \in V_K$. We will denote J's vertex that corresponds to v by v^J. If there are multiple isomorphic embeddings of K in J, v^J refers to one of them. Similarly, if $S \subseteq V_K$, then the corresponding vertices in J are denoted by S^J.

5.2.2 Related work

The most basic case of counting induced patterns in graphs is that of counting triangles. Itai and Rodeh [25] showed that this can be done faster than by exhaustive enumera-

tion in $O(n^3)$ time. Raising the graph's adjacency matrix A to the third power gives the number of walks of length 3 between pairs of nodes. Element $A^3_{x,x}$ represents the number of walks of length 3 that start and finish in the node x, which corresponds to the number of triangles that include x. The total number of triangles is then $\frac{1}{6}\sum_{x\in G} A^3_{x,x}$. Note that the same triangle is counted twice for each of its three nodes. The time complexity of this procedure equals that of multiplying two matrices, which is faster than exhaustive enumeration of triangles in dense graphs. A natural extension of this result is to larger cliques. Nešetřil and Poljak [26] studied the problem of detecting a clique of size k in a graph with n nodes. They showed that this problem can be solved faster than with the straight-forward $O(n^k)$ solution. Their approach reduces the original problem to detection of triangles in a graph with $O(n^{k/3})$ nodes. Since we can detect triangles faster than in $O(n^3)$ with fast matrix multiplication algorithms, we can also detect cliques of size k faster than $O(n^k)$.

Counting all non-induced subgraphs is as hard as counting all induced subgraphs because they are connected through a system of linear equations. Despite this it is sometimes beneficial to compute induced counts from non-induced ones. Rapid Graphlet Enumerator (RAGE) [22] takes this approach for counting four-node graphlets. Instead of counting induced subgraphs directly, it reconstructs them from counts of non-induced subgraphs. For computing the latter, it uses specifically crafted methods for each of the 6 possible subgraphs (P_4, claw, C_4, paw, diamond, and K_4). The time complexity of counting non-induced cycles and complete graphs is $O(e \cdot d + e^2)$, while counting other subgraphs runs in $O(e \cdot d)$. However, the run-time of counting cycles and cliques in real-world networks is usually much lower.

Some approaches exploit the relations between the numbers of occurrences of induced subgraphs in a graph. Kloks *et al.* [29] showed how to construct a system of equations that allows computing the number of occurrences of all six possible induced four-node subgraphs if we know the count of any of them. The time complexity of setting up the system equals the time complexity of multiplying two square matrices of size n. Kowaluk *et al.* [30] generalized the result by Kloks to counting subgraph patterns of arbitrary size. Their solution depends on the size of the independent set in the pattern graph and relies on fast matrix multiplication techniques. They also provide an analysis of their approach on sparse graphs, where they avoid matrix multiplications and derive the time bounds in terms of the number of edges in the graph.

Floderus *et al.* [52] researched whether some induced subgraphs are easier to count

than others as is the case with non-induced subgraphs. For example, we can count non-induced stars with k nodes, $\sum_{x \in V} \binom{c(x)}{k-1}$, in linear time. They conjectured that all induced subgraphs are equally hard to count. They showed that the time complexity in terms of the size of G for counting any pattern graph H on k nodes in graph G is at least as high as counting independent sets on k nodes in terms of the size of G.

Vassilevska and Williams [53] studied the problem of finding and counting individual non-induced subgraphs. Their results depend on the size s of the independent set in the pattern graph and rely on efficient computations of matrix permanents and not on fast matrix multiplication techniques like some other approaches. If we restrict the problem to counting small patterns and therefore treat k and s as small constants, their approach counts a non-induced pattern in $O(n^{k-s+2})$ time. This is an improvement over a simple enumeration when $s \geq 3$. Kowaluk *et al.* [30] also improved on the result of Vassilevska and Williams when $s = 2$. Alon et al. [54] developed algorithms for counting non-induced cycles with 3 to 7 nodes in $O(n^{\omega})$, where ω represents the infimum of exponents over all matrix multiplication algorithms.

Alon *et al.* [55] introduced the color-coding technique for finding simple paths and cycles in graphs. Their technique is applicable not just to paths and cycles but also to other patterns with bounded treewidth. The authors of [56] used such color-coding approach to approximate a 'treelet' distribution (frequency of non-induced trees) for trees with up to 10 nodes.

Recently, Melckenbeeck et al. [41] published a paper that describes how to generate systems of equations similar to those used in the ORCA algorithm [33] for arbitrarily large graphlets. However, the resulting equations do not satisfy the requirements needed for an efficient counting algorithm. Consider for example the equation $o_{50} + o_{55} = \sum_{P_7(x,a,b,c)} (c(a,b,c) - 1)$. There can be as many as $O(ed^2)$ sets of nodes $\{a, b, c\}$ with a nonzero number of common neighbours, which makes the computation of common neighbours the limiting factor in terms of space and time requirements. The method we present in this paper and the related proofs show how to avoid this issue and construct an efficient algorithm for arbitraty graphlet sizes.

5.2.3 *Outline of the proposed algorithm*

We will derive a system of linear equations that relate the orbit counts of a fixed node for graphlets with k vertices. The coefficients on the left-hand sides reflect the symmetries in the graphlets and do not depend on the host graph, so they are derived in

advance. The right-hand sides are computed as sums over graphlets with $k-1$ vertices induced in the host graph H, and the sums include terms that represent the number of common neighbours of certain vertices in the embeddings of graphlets in H.

The resulting system of equations will be triangular and have a rank of $|\mathcal{O}_k|-1$. We can efficiently enumerate the complete graphlet, after which the system of equations for the remaining orbit counts can be solved using integer arithmetic, thus avoiding any numerical errors.

5.2.4 Original contributions

We already presented the original idea of the algorithm in a recent article in Bioinformatics [33], in which we focused on its use in genetics and avoided formal descriptions and analysis. In this paper we

1. present the algorithm more formally;
2. describe a general method for derivation of the system of equations relating the orbit counts;
3. generalize it to induced subgraphs of arbitrary size; in particular, we prove that the system of equations with the properties required for the efficient implementation of the algorithm exists for any $k \geq 4$;
4. provide worst time-complexity analysis and the analysis of the expected time complexity on random graphs;
5. empirically explore the efficiency of the orbit counting algorithm and compare it with the theoretical results.

The remainder of the paper is composed of two parts. In the next section we show a technique for building the system of equations with desired properties, and in the following section we present an algorithm based on them and analyze its time- and space-complexity.

5.3 Relations between orbit counts

We will show how to construct linear relations between a chosen orbit count o_i and some orbits belonging to graphlets with a larger number of edges. We will illustrate

the procedure on figures showing the derivation of the following Equation 5.1 that relates the count for orbit 59 and counts for orbits 65, 68 and 70.

$$o_{59} + 4o_{65} + 2o_{68} + 6o_{70} = \sum_{\substack{x_1,x_2,x_3: \\ x_1<x_2\wedge x_3\notin N(x), \\ H[\{x,x_1,x_2,x_3\}]\cong G_7}} [(c(x_1,x_3)-1)+(c(x_2,x_3)-1)] \quad (5.1)$$

5.3.1 *Derivation of general relations between orbit counts*

Let orbit O_i appear in a connected simple k-node graphlet $G_a = (V_G, E_G)$ ($a = m(i)$). We denote the G_a's node that is in orbit O_i by x; if there are multiple such nodes, we pick one. Next, we choose a node $y \neq x$, such that $G' = G_a \setminus \{y\}$ is still a connected graph; we will impose additional constraints on y later to ensure an efficient implementation of the algorithm. According to our notation, $x^{G'}$ is the node in G' that corresponds to x in G_a; let O_m be its orbit. We label the remaining $k-2$ nodes with $x_1, x_2, ..., x_{k-2}$ (Fig. 5.3).

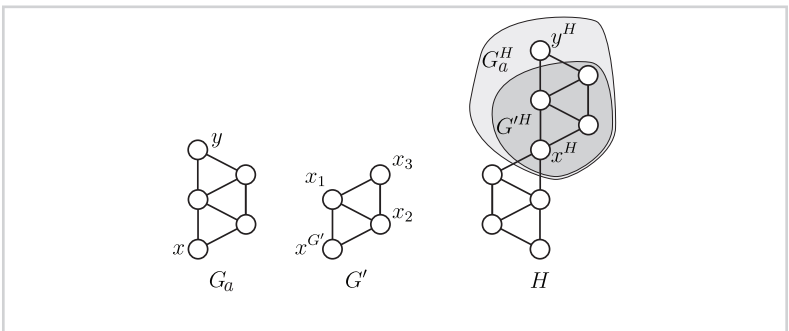


Figure 5.3

Reducing the k-node graphlet G_a to a $k-1$ node graphlet G'.

Fig. 5.3 illustrates reducing the k-node graphlet G_a to a $k-1$ node graphlet G'. O_{59} appears in $G_a = G_{24}$; the node in O_{59} is labelled x. Removal of node y results in $G' = G_7$. The node in G' that corresponds to x, $x^{G'}$, belongs to O_{12}. We assigned labels x_1, x_2 and x_3 to the remaining nodes of G'. Embeddings of G_a and G' in H are referred to as G_a^H and G'^H, respectively. The nodes corresponding to x and y are marked by x^H and y^H.

We now go in the opposite direction: starting with G', we consider its possible extensions to G_a. Let $E \subset V_{G'}$ be a set of nodes such that adding a new vertex y

connected to all vertices in E yields G_a with x in orbit O_i (Fig. 5.4). Let $\mathscr{E}$ be a set of all such subsets E.

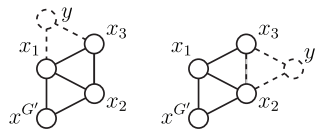


Figure 5.4

Extensions of the $k-1$-node graphlet G' to G_a.

Fig. 5.3 shows extensions of the $k-1$-node graphlet G' to G_a. G_7 can be extended to G_{24} by attaching y to either x_1 and x_3 or to x_2 and x_3, hence $\mathscr{E} = \{\{x_1, x_3\}, \{x_2, x_3\}\}$. With respect to (5.2), x_1 and x_3 (as well as x_2 and x_3) have one common neighbour in G', so $c(E^{G'}) = 1$ and $\sum_{E\in\mathscr{E}} \left(c(E^H) - c(E^{G'})\right) = (c(x_1, x_3) - 1) + (c(x_2, x_3) - 1)$. The right side in (5.1) sums this over all unique occurrences of $G' = G_7$ with x in O_{12} within H.

Let G'^H be some particular occurrence of G' in H. To count o_i for the node x^H (the node in H to which x maps), we need to explore the extensions of G'^H to G_a^H. A necessary (but insufficient) condition to put x^H into O_i is that the additional node y is a common neighbour of all vertices E^H for one of $E^H \in \mathscr{E}^H$ (with respect to the particular occurrence of G' in H). There are at most

$$\sum_{E\in\mathscr{E}} \left(c(E^H) - c(E^{G'})\right) \tag{5.2}$$

candidate nodes y; $c(E^{G'})$ represents the number of neighbours of E that are already in G' (*i.e.* x_i) and cannot be mapped to y. Equation (5.2) represents the term in the sum in the right side of the relation. To compute the total orbit count o_i for x, we sum (5.2) over all occurences of G' in H (Fig. 5.4).

Condition $y \in N(E^H)$ (for some $E^H \in \mathscr{E}^H$) is not sufficient. Node y can also be connected to any of the other $k-1-|E|$ ($E \in \mathscr{E}$) nodes in G'^H, resulting in $2^{k-1-|E|}$ possible graphlets and orbits for x. The counts for these orbits are summed on the left side of the relation (Fig. 5.5).

Dashed lines in Fig. 5.5 represent edges required by condition $y \in N(x_1, x_3)$. Dotted lines represent possible extra edges that make the resulting induced graph isomorphic to G_{26}, G_{68} or G_{70} instead of G_{24}, with x in orbits o_{65}, o_{68} or o_{70} instead of o_{59}; these orbits appear on the left-hand side of (5.1).

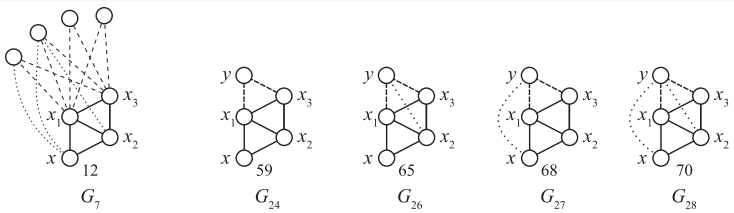


Figure 5.5

Extended graphlets with extra edges.

While adding the orbit counts on the left-hand side, we need to account for the over-counts, that is, the number of times that (5.2) counts the same occurrence of $G^* = G_{m(p)}$ (the graphlet containing the orbit p) within H. G^* is obtained by extending G' with $y \in N(E)$. The coefficients on the left-hand side thus equals the number of ways in which G' can be extended to G^* with a fixed node x (Fig 5.6).

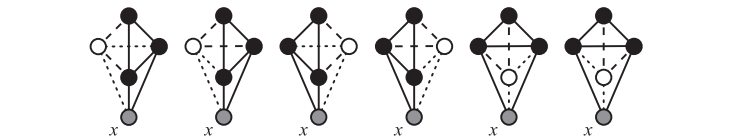


Figure 5.6

Symmetries and coefficients.

The coefficient at o_{70} in 5.1 is 6 because each induced embedding of $G^* = G_{m(70)} = G_{28}$ in H is counted six times. First, there are three different choices for y (the white node), therefore each embedding of G_{28} results in three corresponding appearances of graphlet G_7 (solid edges) with x in orbit O_{12}. For each occurrence, both extensions (dashed edges) lead to x in orbit O_{70}.

In general, we have to consider all induced occurrences of G' in G^* (with a fixed point x), which is the same as considering nodes $z \in V_{G^*}$ whose removal results in G' with $x^{G'}$ in orbit O_p. For every such case we increase the coefficient by the number of extensions $E \in \mathscr{E}$ such that node z is connected to the extension nodes, i.e. $N(z) \supseteq E^{G^*}$.

The general procedure for relating the orbit count o_i with counts of orbits with higher indices is outlined in Algorithm 1.

Algorithm 1

Derive an equation for orbit O_i.

function EQUATION(O_i)
  $G_a \leftarrow G_{m(i)}$ ▷ Let G_a be the graphlet that contains O_i
  $x \in O_i$ ▷ and x one of the nodes in O_i.
  $y \leftarrow$ SELECTY(x) ▷ Alg. 2 - Pick node y such that $y \neq x$
  $G' \leftarrow G_a \setminus y$ ▷ and G' is a connected graph.

  $r \leftarrow \sum_{G'^H : x^H \in O_i}$ (Equation 5.2) ▷ The right side of equation sums over.
  ▷ all occurrences of G' in the host graph.

  for $p \in \mathscr{O}$ *do* ▷ Construct left side of the equation.
    $G^* = G_{m(p)}$ ▷ Graphlet containing orbit O_p.
    $f_p \leftarrow 0$ ▷ Overcount coefficient of orbit O_p.
    for $z \in G^* : (G^* \setminus z) \cong G'$ *do* ▷ Is z in the same orbit as y
    ▷ given a fixed point x?
      $f_p \leftarrow f_p + |\{E \in \mathscr{E} : N(z) \supseteq E\}|$ ▷ By how many extensions?
    end for
  end for
  $l \leftarrow \sum_p f_p \cdot o_p$ ▷ Left side is a weighted sum of orbit counts.

  return equation $l = r$
end function

5.3.2 *Additional constraints on selection of y*

In the preceding derivation, the only limitation on selection of vertex y was that the remaining graphlet is still connected. Different choices of y yield different equations. With the coefficients independent of the host graph and known in advance, the time consuming part of using these equations to calculate orbit counts is the computation of the right-hand side terms. To speed it up, we impose some additional constraints on the choice of the node y: the restraints will be such that the right-hand sides will contain only the counts $c(S)$ in which either $|S| < k-2$, or equal $|S| = k-2$ with the

nodes in S forming a connected subgraph of G_k. This will allow pre-calculation and caching of all $c(S)$ needed for computation of right-hand sides.

For efficient precomputation, vertex $y \neq x$ must meet the following criteria:

C1. $d(y) \leq k - 2$,

C2. $G \setminus \{y\}$ is a connected graph,

C3. if $d(y) = k - 2$, the neighbours of y induce a connected graph,

where $d(y)$ represents the degree of y.

Theorem 1: A vertex that meets criteria C1, C2 and C3 exists in any graphlet with $k \geq 4$ vertices and all possible x, except for complete graphlets (all vertices violate the first condition) and for the cycle on four points, C_4 (all vertices violate the last condition).

Let L_i represent the set of vertices at a distance i from x (see Fig 5.7). Let ℓ_i be the vertex in L_i with the smallest degree. Let L_u be the last non-empty set, and, accordingly, ℓ_u the vertex with the smallest degree among the vertices farthest from x. We will show that ℓ_u fulfils the conditions in most cases, except in some for which we can use ℓ_{u-1}.

Algorithm 2

Selection of node y.

```
function SELECTY(x)
    u ← distance to the furthest node from x
    L_u, L_{u-1} ← set of nodes at distance u and u − 1, respectively
    ℓ_u, ℓ_{u-1} ← lowest degree node from L_u and L_{u-1}, respectively
    if ℓ_u satisfies the criteria then
        y ← ℓ_u
    else
        y ← ℓ_{u-1}
    end if
    return y
end function
```

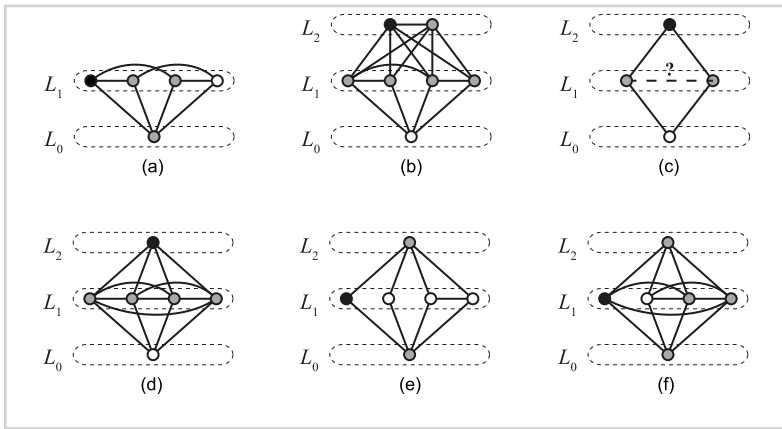


Figure 5.7

Illustrations of different cases.

In Fig. 5.7 the node at the bottom level L_0 *represents node* x*. The selected node* y *is colored black and its neighbours are indicated with a gray color. (a)* $u = 1$*. (b)* $u = 2 \wedge |L_2| > 1$*. (c)* $u = 2 \wedge |L_2| = 1 \wedge k = 4$*. (d)* $u = 2 \wedge |L_2| = 1 \wedge k \geq 5 \wedge d(\ell_1) = k - 1$*. (e)* $u = 2 \wedge |L_2| = 1 \wedge k \geq 5 \wedge d(\ell_1) < k-2$*. (f)* $u = 2 \wedge |L_2| = 1 \wedge k \geq 5 \wedge d(\ell_1) = k-2$*.*

Each node $v \in L_i$ $(i > 0)$ has at least one neighbour in L_{i-1}, since the first node in every shortest path from v to x belongs to L_{i-1}. Consequently, all L_i for $i \leq u$ are non-empty. Note also that vertices from L_i are adjacent only to vertices L_{i-1}, L_i, and L_{i+1} since any edge from L_i to L_j with $j < i - 1$ would imply a shorter path from the node in L_i to x.

Lemma 1: A vertex $v \in L_i$ can have a degree of at most $k - i$.

The vertex v is not adjacent to any vertex in L_j, where $0 \leq j < i - 1$. Since L_j are non-empty, there are at least $i - 1$ non-adjacent vertices to v, so the degree of v is at most $k - 1 - (i - 1) = k - i$.

As a consequence, if $d(\ell_u) = k - 2$, then $u \leq 2$.

Lemma 2: If $d(v) = k - 2$ and $v \in L_2$, then v is adjacent to all vertices except x.

A vertex in L_2 is not adjacent to x by definition of L_2, and there are no loops, so to have a degree of $k - 2$ it must be adjacent to all other vertices.

Lemma 3: If G is not a complete graph, then $d(\ell_u) \leq k-2$

For $u > 1$, the lemma follows directly from Lemma 1, so we only need to prove it for $u = 1$. For contrapositive, assume that $d(\ell_1) = k-1$. Since ℓ_1 has the smallest degree in L_1, all vertices in L_1 have a degree of $k-1$. Furthermore, x has degree $k-1$ since all vertices in L_1 are adjacent to it by definition of L_1. Hence, G is a complete graph.

The last lemma ensures that the farthest vertex with the lowest degree, ℓ_u, fulfills the first condition. It also fulfills the second one: all vertices are connected to x with the shortest paths of lengths at most u, which cannot include ℓ_u, thus the removal of ℓ_u keeps them connected (at least) via x.

We will prove that ℓ_u also fulfills the third condition, except for one special case ($d(\ell_u) = k-2$ and $u = 2$ and $|L_2| = 1$ and $k \geq 5$ and $d(\ell_1) \leq k-2$), in which we choose another suitable vertex. We will consider six different cases, which are (except for the trivial first case) illustrated in Fig 5.7.

1. $d(\ell_u) < k-2$*:* Condition (C_3) does not apply.

2. $d(\ell_u) = k-2$ *and* $u = 1$*:* Since all vertices except x are in L_1, they are adjacent to x (Fig 5.7a). x itself is among the neighbours of ℓ_1, hence neighbours of ℓ_1 are connected through x.

3. $d(\ell_u) = k-2$ *and* $u = 2$ *and* $|L_2| > 1$*:* Since ℓ_2 is the vertex with the smallest degree in L_2, all vertices in L_2 must have a degree of $k-2$ and are adjacent to all vertices except x by Lemma 2 (Fig 5.7b). The neighbour set of ℓ_2 is $L_1 \cup L_2 \setminus \ell_2$. Since $|L_2| > 1$, there exists a vertex $v \in L_2$ s.t. $v \neq \ell_2$. v is adjacent to all nodes from $L_2 \cup L_1$, therefore $L_1 \cup L_2 \setminus \ell_2$ is connected.

4. $d(\ell_u) = k-2$ *and* $u = 2$ *and* $|L_2| = 1$ *and* $k = 4$*:* L_1 contains two vertices; both are adjacent to x by definition of L_1 and to ℓ_2 since $d(\ell_2) = k-2 = 2$. ℓ_2 is not adjacent to x by definition of L_2. This leaves only two possible graphs, the cycle C_4 and a diamond (Fig 5.7c). For the former, the vertex with the required properties does not exist. For the diamond, ℓ_u fulfills all three conditions.

5. $d(\ell_u) = k-2$ *and* $u = 2$ *and* $|L_2| = 1$ *and* $k \geq 5$ *and* $d(\ell_1) = k-1$*:* The neighbour set of ℓ_2 is the entire L_1 (Fig 5.7d). Since the smallest degree in L_1 is $k-1$, L_1 is a complete graph and therefore connected.

6. $d(\ell_u) = k-2$ *and* $u = 2$ *and* $|L_2| = 1$ *and* $k \geq 5$ *and* $d(\ell_1) \leq k-2$*:*
The graph consists of $L_0 = \{x\}$, $L_2 = \{\ell_2\}$, and of L_1 with at least 3 vertices since $k \geq 5$ (Fig 5.7e). All nodes in L_1 are adjacent to x by definition of L_1 and to ℓ_2 by Lemma 2 since we assume $d(\ell_2) = k-2$.

In this case, ℓ_u does not always fulfil the conditions, so we choose the lowest degree vertex from L_1, ℓ_1. It fulfils the condition (i) by assumptions of this special case. As for condition (ii), the graph $G \setminus \ell_1$ is still connected since all points in L_1 are adjacent to x. Since $|L_1| \geq 3$ and $d(\ell_u) = d(\ell_2) = k-2$, vertices x and ℓ_2 are connected through the remaining vertices in $L_1 \setminus \ell_1$.

Condition (iii) needs to be verified just for the case when $d(\ell_1) = k-2$ (Fig 5.7f). The neighbours of ℓ_1 include x, ℓ_2 and all vertices from L_1 except one. Since $|L_1| \geq 3$, L_1 must include at least one other neighbour of ℓ_1, which thus connects x and ℓ_u.

We have covered all possible cases: the degree of ℓ_u cannot exceed $k-2$ due to Lemma 3 (assuming the graph is not complete), and when $d(\ell_u) = k-2$, u cannot exceed 2 due to Lemma 1.

We have proven that the vertex with the smallest degree in L_u, ℓ_u, fulfills the given conditions in all cases except when $d(\ell_u) = k-2$ and $u = 2$ and $|L_2| = 1$ and $k \geq 5$ and $d(\ell_1) \leq k-2$. In the latter case, the conditions are fulfilled by ℓ_1. Complete graphlets and C_4 are handled differently.

5.3.3 *Equation for a cycle on 4 nodes*

A cycle on 4 nodes, C_4, is treated separately since there is no suitable node y with the required properties. For C_4 (O_8) we choose one of the nodes adjacent to x for the role of y, resulting in

$$2o_8 + 2o_{12} = \sum_{\substack{x_1,x_2: \\ x,x_2 \in N(x_1), \\ H[\{x,x_1,x_2\}] \cong G_1}} [c(x,x_2) - 1]\,. \tag{5.3}$$

Note that this choice violates the third condition that the neighbours of y should induce a connected graph. The equation 5.3 contains a term on the right side that corresponds to the number of common neighbours of node x and some other node x_2

at distance 2 from x. The algorithm stores precomputed values for all such pairs x and x_2, which would require $O(nd^2)$ space and increase the algorithm's space complexity. However, we can still handle this case without consequences for the time and space complexity. We achieve this by reusing $O(n)$ space and recomputing the number of common neighbours every time the algorithm starts a computation of orbit counts for a different node of interest x. This optimization is necessary to keep the space requirement at $O(nd)$ for counting four-node graphlets and does not impact the time complexity.

5.3.4 *System of equations*

In the constructed system of equations, each orbit is related to orbits from graphlets with higher number of edges. This yields a triangular system of equations: we have one equation for every orbit O and these equations include as terms only the orbit O and other orbits belonging to graphlets with a larger number of edges (*e.g.*, the orbit 59 in (5.1) is related to orbits 65, 68 and 70).

The system has $\mathscr{O}_k - 1$ linear equations for $\mathscr{O}_k$ orbit counts. To solve it, one orbit count must be enumerated directly. The networks that we encounter in practical applications are usually sparse, which makes the complete graphlet (clique) a good candidate. Because of its rarity and symmetricity, we can efficiently restrict the enumeration.

Enumerating the orbit in the graph with the largest number of edges also simplifies solving the given triangular system of equations.

5.3.5 *Extension to edge orbits*

Edge orbits (Fig 5.8) can be defined in a similar way as node orbits. We can use the same approach for setting up the corresponding system of equations. Since the system does not refer to a single x but to an edge (x_a, x_b), the selected node y must not coincide with either of these endpoints. We can set $x = x_a$ and show that we can always choose a node $y \neq x_b$ with the required properties.

Since x_b is always in L_1, we have to analyze only the cases where we choose y from L_1. This happens in cases 1, 2, and 6 from the proof in section about additional constraints on y. We need to check that there at least two suitable vertices for y, so if one of them is x_b, the other is chosen as y.

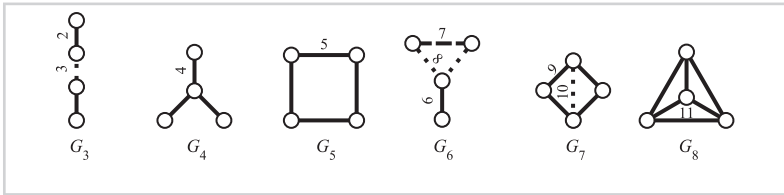


Figure 5.8

Edge orbits in four-node graphlets. Different styles of lines within a graphlet indicate edge orbits.

1. $d(\ell_u) < k-2$: We need to consider only the case when $u = 1$. Since $d(\ell_u) < k-2$, all vertices in L_1 satisfy condition (i). The remaining graph is connected through x (condition (ii)), and condition (iii) does not apply.

2. $d(\ell_u) = k-2$ *and* $u = 1$: Recall that the graph is not complete. Since all vertices in L_1 are connected to x, there must be at least one pair in L_1 that is not connected and thus has a degree of $k-2$. These two vertices satisfy condition (i). Conditions (ii) and (iii) again hold since all vertices are connected through x.

6. $d(\ell_u) = k-2$ *and* $u = 2$ *and* $|L_2| = 1$ *and* $k \geq 5$ *and* $d(\ell_1) \leq k-2$: Since the vertex in L_2 has a degree of $k-2$, it is connected to all vertices in L_1; nodes in L_1 are connected to x. The nodes in L_1 do not induce a complete graph ($d(\ell_1) \leq k-2$), so there must again exist a disconnected pair in L_1, which satisfies all conditions like in above case 2.

For C_4, one of the nodes in L_1 is x_b and the other is y.

5.4 Algorithm

Coefficients on the left-hand side of the relations are related to symmetry properties of the graphlets and not to the host graph H. The terms on the right sides of equations depend on the host graph H. Their computation requires enumeration of all graphlets of size $k-1$ and adding up their possible extensions.

The first step is pre-computation and storing of $c(\mathscr{S})$ for all subsets S with up to $k-3$ vertices and for all connected subsets of $k-2$ vertices. These conditions match the criteria for selection of y, so the precomputed values $c(\mathscr{S})$ represent the terms in the sum on the right-hand sides of equations.

This is followed by direct enumeration of cliques with k vertices. This enumeration does not have to be extremely fast, but just fast enough not to dominate the time

complexity of the entire graphlet counting algorithm. For this purpose we can employ some of the approaches to listing cliques [39, 57].

Following this precomputation, the next two steps are repeated for each vertex $x \in V$.

Computation of sums on the right-hand sides of equations. Computation is implemented as enumeration of $(k-1)$-node graphlets touching x, as specified by the conditions under the sums. For each graphlet, the terms in the sum consist of the counts $c(\mathscr{S})$ precomputed in the previous step.

For $k \leq 5$, the number of graphlets with $k-1$ nodes is small, so it is feasible to design efficient individual procedures for enumerating them. Because the graphlets are small, these procedures involve early pruning of non-viable candidates and completely avoiding any isomorphism testing. Medium-sized graphlets ($k = 5$ or 6) require graphlet recognition of enumerated connected subgraphs, however these patterns can be efficiently distinguished with the use of some trivial invariants such as a *degree sequence*. Enumeration of larger graphlets would benefit from efficient methods for isomorphism testing.

Solving the system of equations. The system is triangular, with each equation relating one orbit to those with larger number of edges, Since the count for the highest orbit, which belongs to the clique, is computed by direct enumeration, the system can be solved by decreasing orbit indices.

All orbit counts, coefficients and free terms are integers, thus the computation is numerically stable.

5.4.1 Time- and space-complexity

We will analyze the worst-case complexity and the expected complexity on random Erdős-Rényi graphs, followed by empirical verification.

Worst-case complexity

We will evaluate the worst-case time complexity of the algorithm in terms of the number of nodes (n) and the maximum degree of a node (d) in the host graph. We treat the size of the graphlets, k, as a constant. We assume that the graph is stored as a list of adjacent nodes together with a hash table for checking whether two nodes are connected in constant time. The algorithm consists of four steps.

Precomputation of common neighbours. We need to precompute the number of common neighbours of sets of $k-3$ or fewer nodes and of connected sets of $k-2$ nodes to efficiently construct right sides of our equations. To achieve this we enumerate all subsets of $k-2$ or fewer neighbours for every node. This results in time complexity $O(nd^{k-2})$. Storing the number of common neighbours of sets of at most $k-3$ nodes with the above method requires $O(nd^{k-3})$ space. Because we request that in the case of $k-2$ nodes, these nodes induce a connected subgraph, we can limit their number to the number of $(k-2)$-node induced connected subgraphs, which is also $O(nd^{k-3})$.

Enumeration of cliques. We will refer to the time complexity of counting k-node cliques in this step as $O(T_k)$. A worst-case time complexity is $O(nd^{k-1})$ and requires constant space. However, this enumeration can be implemented very efficiently in practical applications on sparse networks that contain few cliques.

Enumerating all $(k-1)$-node graphlets and counting their extensions. This step computes the right sides of the system of equations. It requires constant space, since the space is reused for each vertex, and runs in $O(nd^{k-2})$ time needed for enumeration of $k-1$-node graphlets.

Solving the system of equations. Solving the system requires constant time and space.

Overall, the algorithm has a $O(nd^{k-2}+T_k)$ time complexity while requiring $O(nd^{k-3})$ space. In the worst case, the time complexity is the same as that of a simple exhaustive enumeration method, $O(T_k) = O(nd^{k-1})$. However, the term T_k is much smaller in practice.

Expected time complexity in random graphs

Although the worst-case time complexity of the algorithm is equal to that of brute-force graphlet enumeration, the actual performance on real-world networks and on random graphs is much better. We analyzed the expected time complexity on random Erdős-Rényi graphs with n nodes and edge probability p. Throughout this analysis we will assume that $np > 1$, otherwise the graph is likely to have more than one component which can be processed independently.

The precomputation consists of iterating over central nodes, enumerating all sets of $t \leq k-2$ neighbours and incrementing the number of common neighbours of the

leaf nodes. The t nodes have to be connected to the central node, which happens with probability p^t. The expected time complexity of this step is $O(n \sum_{t=1}^{k-2} n^t p^t)$. Assuming $np > 1$, we can simplify it to $O(n^{k-1}p^{k-2})$.

In the second step, the algorithm enumerates all subgraphs with $k-1$ nodes. It does so incrementally by first enumerating smaller connected subgraphs of size t and extending them to larger connected subgraphs. The expected time complexity is therefore proportional to the expected number of connected subgraphs with $t \leq k-1$ nodes. We need to estimate the probability that a set of t nodes induces a connected subgraph. We can view the process of building every such subgraph by consecutively attaching a new node to at least one of the existing nodes. This of course will overestimate the number of connected subgraphs by some constant because every such subgraph can be built in several different orders of attaching nodes. The probability that an edge exists from some newly added node to at least one of the i existing nodes is $1-(1-p)^i$. The expected number of enumerated subgraphs is therefore $O(\sum_{t=1}^{k-1} n^t \prod_{i=1}^{t-1}(1-(1-p)^i)) = O(\sum_{t=1}^{k-1} n^t p^{t-1})$. Assuming $np > 1$, the expected time complexity is $O(n^{k-1}p^{k-2})$.

The total expected time complexity for setting-up the system of equations in Erdős-Rényi graphs with n nodes and edge probability p is thus $O(n^{k-1}p^{k-2})$. In practice, we observe graphlets with 4 and 5 nodes. The expected time complexities for these cases are $O(n^3p^2)$ and $O(n^4p^3)$, respectively.

Empirical evaluation of time complexity

We evaluated the performance of our algorithm for counting 4- and 5-node graphlets on random Erdős-Rényi graphs.

We measured the time needed for counting node- and edge-orbits with the Orca algorithm and compared it to a bruteforce enumeration. For the latter we used an implementation from GraphCrunch. Orca outperforms exhaustive enumeration by an order of magnitude (Tables 5.2 and 5.3). The running times for counting node-orbits and edge-orbits are practically the same in the case of counting 4- or 5-node graphlets in random graphs with 1 000 nodes and of increasing density. The size of the graphs ($n = 1\,000$) was chosen arbitrarily to put the run times in the range of a couple of seconds. In the remainder of this section we focus on counting node-orbits.

Second, we compare the running times of counting orbits of 4-node and 5-node graphlets on random graphs with 10 000 nodes and up to 800 000 edges. These graphs

Table 5.2

Comparison of run times for counting node- and edge-orbits of 4-node graphlets.

	four-node graphlets			
edges [thousands]	*50*	*100*	*150*	*200*
bruteforce [s]	27.24	236.83	811.52	1876.98
node-orbits [s]	0.70	2.40	6.16	14.01
edge-orbits [s]	0.69	2.33	6.12	14.21

Table 5.3

Comparison of run times for counting node- and edge-orbits of 5-node graphlets.

	five-node graphlets				
edges [thousands]	*5*	*10*	*15*	*20*	*25*
bruteforce [s]	0.87	12.75	61.57	191.94	461.53
node-orbits [s]	0.23	1.03	2.93	6.85	13.88
edge-orbits [s]	0.22	0.91	2.54	5.90	11.78

are sparse as the number of edges represents only about 1.6% of all possible edges; as such they represent a realistic case of large network analysis. A logarithmic plot of execution times in Fig 5.9 shows polynomial dependencies on the size of graphlets. Dotted lines correspond to a bruteforce enumeration method and solid lines to our Orca algorithm. The line corresponding to bruteforce counting of 4-node graphlets aligns nicely with counting 5-node graphlets using Orca. This reflects the enumeration of 4-node graphlets used to compute 5-node graphlet counts. The only difference is by a constant factor.

The slope of the Orca's lines should be 2 and 3, respectively, according to the expected time complexities ($O(n^3p^2)$, $O(n^4p^3)$). However, this is clearly not the case in Fig 5.9. Further experiments show that this is the result of CPU cache misses when accessing the precomputed lookup tables. We performed a similar experiment with disabled CPU cache. Because of the slowdown, we decreased the number of nodes to 1 000 and maintained average degree of nodes. The measurements with disabled CPU cache in Fig 5.10 line up with the expected slopes of 2 and 3.

Finally, we probed for the region in which the enumeration of cliques begins dominating the time complexity. We performed the experiment for counting 4-node graphlets in graphs with 1 000 nodes and increasing edge probabilities p. In Fig 5.11 the plot

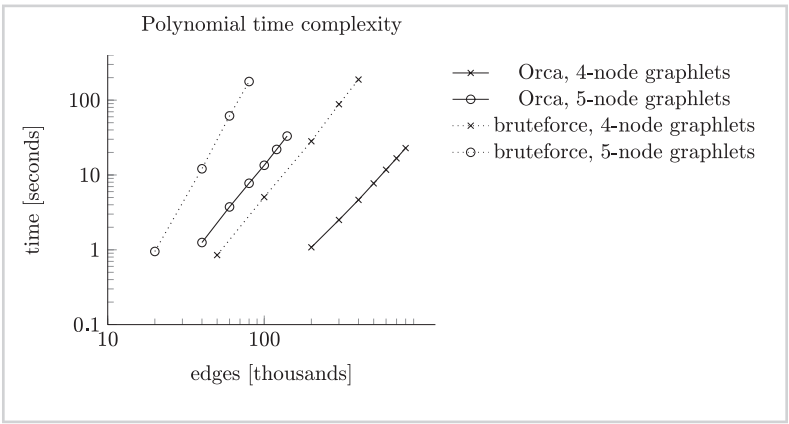


Figure 5.9

Comparison of counting times for orbits of 4- and 5-node graphlets on sparse graphs with 10000 nodes and varying densities.

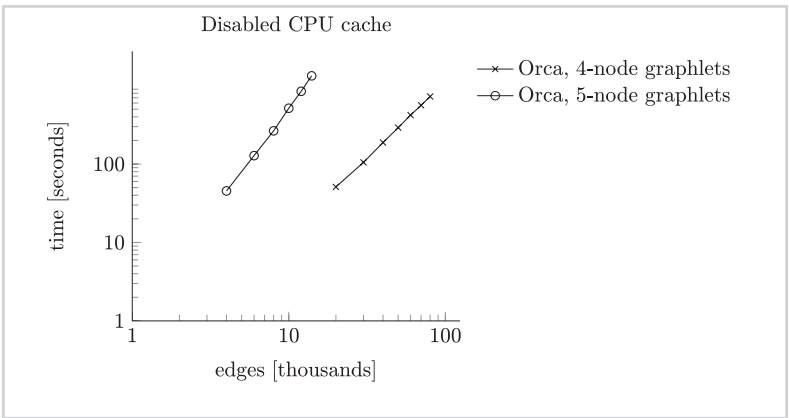


Figure 5.10

Running times with disabled CPU cache.

follows a straight line up to around $p = 0.07$ and another steeper line from $p = 0.3$ onwards. This is consistent with the contribution of the step of enumerating cliques. Random sparse graphs contain fewer cliques whose enumeration is efficient and does not significantly affect the running time. However, as the graphs become denser, this becomes the bottleneck of the algorithm.

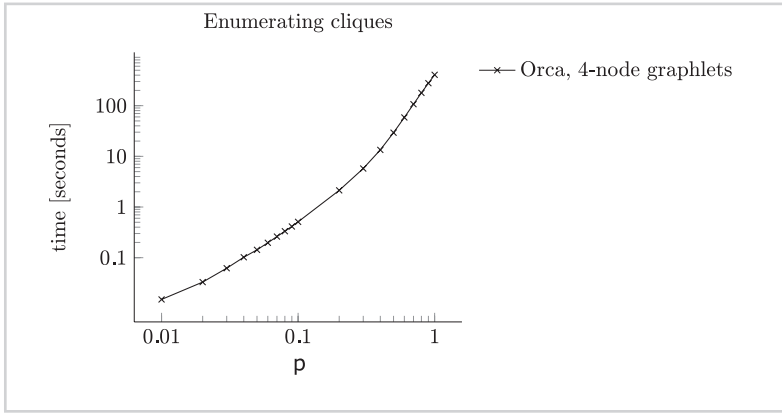


Figure 5.11

Enumerating cliques. The effect of enumerating cliques on running times in random graphs of increasing density.

5.5 Final remarks

The source code of the algorithm in C++, which computes the node and edge orbits for $k = 4$ and $k = 5$, along with the randomly generated data used in experiments is available at `https://github.com/thocevar/orca`. The corresponding R package `orca` is also available on CRAN. Parts of this algorithm that have been presented previously are also already included in the GraphCrunch package [20].

6

Graphlet counting in dynamic graphs

This chapter presents our work on adaptation of Orca for graphlet counting in dynamic graphs. We developed two dynamic versions of Orca, which also illustrates the versatility of our graphlet counting approach. The first one supports individual operations of adding or removing an edge and querying individual nodes to obtain their orbit counts. The second version continually maintains the orbits counts of all nodes subject to additions or removals of individual edges.

Generating large random networks with approximately the given graphlet counts is an application of such dynamic algorithms, which was also the motivation for this research. If the researcher has a real-world network, he or she can use Orca or a similar algorithm for extraction of possibly approximate graphlet counts, and then use the presented approach to generate an arbitrary number of topologically similar, but random networks.

However, such approaches should be used with caution to avoid limiting the analysis to too similar networks. An extreme example would be one where a researcher is observing the number of triangles and generating random networks that match in the number of 4-node graphlets. As a result, the number of triangles will be the same in all the generated random networks, which makes such random network model unsuitable in this case.

6.1 Generating random graphs

Networks arising in different contexts have different characteristics. The simplest random model of graphs is the Erdős-Rényi (ER) model [58]. The are two versions of the ER model. In the first version all edges are independent and exist with the same probability p. In the second version with a fixed number of edges all labeled graphs with the given number of nodes and edges are equally likely. Such model does not reflect the properties observed in networks that commonly arise in real-world situations. For instance, the degree distribution in ER graphs is binomial, while in many real-world networks it follows a power law. Networks exhibiting the latter property are called scale-free networks [59].

To empirically verify the suitability of some new algorithm in a particular area, like analysis of protein-protein interaction (PPI) networks, we identify or assume the corresponding type of the network and then use some procedure for generating random networks of such kind. Assuming that PPI networks are similar to scale-free network (degree distribution follows a power law), preferential attachment procedure [59] can

be used to generate an arbitrary number of random instances of PPI-like networks.

The problem with this approach is that it may be difficult to identify the correct kind of network. For instance, authors in [1] argue that geometric graphs are a better match for PPI networks. In general, the network may not sufficiently match any of the common network types, or it may belong to a class of networks but exhibit some peculiar patterns, like an unexpectedly high or low number of triangles (social networks or adversarial networks) or a large number of nodes of degree two (places on road networks).

Network motifs [4] are another concept where a need for generating random graphs with certain properties arises. A subgraph or a pattern is referred to as a network motif if its presence in a network is significantly higher than expected. The expected occurrence of a pattern is usually derived from randomized or random networks of its type.

How do we generate random networks with prescribed properties without relying on common graph models such as ER? One possible description of a graph type could be its degree distribution. Havel-Hakimi [60] algorithm produces a graph with a desired degree distribution by repeatedly connecting the node with the largest degree d_n to the next d_n largest nodes. However, it is not clear how to adapt the approach to generate uniformly distributed samples. Blitzstein and Diaconis [61] proposed a randomized version; since the generated simple graphs are not distributed uniformly, the obtained samples are re-weighted with sequential importance sampling. Milo et al. [62] provide a general review of approaches for generation of graphs with given degree sequences, and identify two common approaches: switching and matching. The former is a Markov chain Monte Carlo approach that starts with a given graph and randomizes it through a sequence of edge switches. The latter (also known as the pairing or configuration model) starts with an empty graph containing edge "stubs" and repeatedly connects them into pairs. When the resulting graph contains loops or multiple edges, the process is restarted. McKay and Wormald [63] tackled this problem by attempting to remove loops and double edges, while still maintaining the uniform distribution of graphs. Bayati et al. [64] introduced bias when making new pairings to increase the probability of generating a simple graph. Britton et al. [65] proposed several methods for generating graphs with approximate degree distributions subject to some restrictions on valid distributions.

Pržulj [1] introduced graphlets as an alternative topological property for compari-

son of networks. We will generate random graphs with desired properties by iteratively improving current solution with local/small changes. First, we need to choose a distance measure, which will reflect how close to the desired solution is the current state. We based our measure on the Relative graphlet frequency distance [1] but removed the "relative" part. Distance between graphs G and H is defined as

$$D(G,H) = \sum_{i=0}^{8} |F_i(G) - F_i(H)| \tag{6.1}$$

$$F_i(G) = \log(N_i(G) + 1),$$

where $N_i(G)$ denotes the count of graphlet of type i in graph G. The distance measure is based on graphlets with at most 4 nodes to simplify the dynamic version of the graphlet counting algorithm used in the process. However, the same approach could be employed to design an efficient 5-node graphlet counting methods for dynamic networks, which could be used in combination with a distance measure based on frequencies of 5-node graphlets.

We will use a simple hill-climbing optimization technique, where at each step we perform a random modification of the network. We continue with the modified network in case it improves our distance measure and revert the change otherwise. Such setting requires an efficient method for graphlet counting in changing networks, which is the subject of the next section. Note that our approach described in the following sections is general enough to be used with other local search optimization techniques besides hill-climbing, e.g. simulated annealing [66], and with other distance measures.

6.2 Dynamic Orca

Our method for generating random networks works by iteratively improving the current solution by adding or removing individual edges. To support this, the graphlet counting algorithm must perform the updates after these operations in a reasonable time, which can be achieved by exploiting the locality of changes. This motivated the adaptation of Orca for dynamic networks.

An often observed subgraph in the area of social network analysis is a triangle. An efficient triangle enumeration approach is to find a subset of heavy-hitters: nodes with degree at least $\sqrt{m}$, where m represents the number of edges in the network. Now we

can design two separate methods: one for finding triangles consisting only of heavy-hitter nodes and one for all the others [67]. Eppstein et al. [68] approached the problem of maintaining 3-node subgraph statistics in dynamic networks using a similar partition of nodes into high- and low-degree nodes based on h-index of the graph (maximum number such that the graph contains h nodes of degree at least h). They extended their work to directed 3-node subgraphs and 4-node subgraphs [69]. Lin et al. [70] proposed a similar data structure, which partitions node's neighbours into those with higher and lower degrees than itself, and can be used for counting 4-node subgraphs. The complexity of operations is related to the h-index and arboricity of the graph.

The developed methods described in the following sections can be used to design dynamic graphlet counting algorithm for larger graphlets and are not limited to 4-node graphlets.

6.2.1 Overview of Orca

Orca is an algorithm that counts the 4-node graphlets by enumerating only triangles and 3-node paths in the network. It achieves this by exploiting relations between orbit counts. We will only briefly summarize the main point of the algorithm here and refer the reader to papers presented in previous chapters, where the algorithm is discussed in detail.

The algorithm for counting 4-node graphlets is based on obtaining a system of equations relating orbit counts for a given node x. One example of such equation is

$$2o_{13} + 6o_{14} = \sum_{y,z:\, y<z, G[\{x,y,z\}]\cong G_2} (c(x,y) - 1) + (c(x,z) - 1).$$

All equations have linear combinations of orbit counts on the left side, which are fixed (prescribed by the algorithm). The right sides of equations depend on the graph but can be computed efficiently. Note that all summations are over 3-node graphlets and consist of expressions $c(a,b)$—the number of common neighbours of two nodes in the graph. Adaptation of the algorithm for a dynamic setting therefore reduces to the problem of maintaining values $c(a,b)$ under addition and removal of edges in the graph.

6.2.2 Dynamic operations

The dynamic setting of the graphlet counting problem consists of the following operations:

- add edge (x, y)
- remove edge (x, y)
- compute orbit counts of node x

The last operation is the most straight-forward. Orca computes orbit counts by building and solving a system of equations for each node. This process is independent and can be parallelized. So if we are interested in orbit counts of only one particular node x, we simply build and solve the corresponding system of equations. To do this, we need accurate values for the number of common neighbours between pairs of nodes.

Maintaining values $c(x, y)$ where $(x, y) \in E$ is the subject of the first two operations. We will describe the case of adding an edge; removal of an edge is similar. Let us assume we add an edge (x, y). First, we have to compute $c(x, y)$ as an intersection of neighbours of nodes x and y. Second, the addition of this edge increases the values $c(x, z), z \in N(x, y)$ and analogously $c(y, z)$. We can update all these values by enumerating the triangles containing the newly added edge, which can be done in $O(d)$.

Let us compare this dynamic approach with the static setting, where we are given a graph and have to compute orbit counts of all nodes. By repeatedly adding edges and at the end querying all nodes for their orbit counts, we in fact achieve the same time complexity as the original Orca in a static setting.

6.2.3 Maintaining graphlet counts

The proposed optimization approach for generating random graphs requires that we evaluate the distance measure after every modification, which in turn requires counting the graphlets. We could employ the previously described dynamic version of ORCA for this purpose, but it is still rather slow. To see why, consider the addition of an edge (x, y). This has an effect on orbit counts of all nodes that are within distance of 2 from either x or y, and should be recomputed. Furthermore, the query for orbit counts

of some node z essentially enumerates all 3-node graphlets containing z, which again consist of nodes at most 2 edges away from z. Together this means that we consider a neighbourhood within distance 4 from x and y, which could very well be almost entire graph.

To overcome this, we modify the dynamic algorithm even further so that it maintains orbit counts of individual nodes and consequently their graphlet counts after every addition or removal of an edge. Looking at equations, we can see that they consist of sums over 3-node graphlets. We can avoid recomputing entire sums by subtracting and adding terms in these sums that actually changed due to the current change of edge (x, y). If a 3-node graphlet contains neither x nor y, the term in the sum that corresponds to this 3-node graphlet will not change. Therefore, we can enumerate all 3-node graphlets that contain at least one of the nodes x or y, and update the corresponding sums. Effectively, we now consider only a neighbourhood within distance 2 from x and y.

6.3 Experiments

We performed several experiments to measure the speed and accuracy of our random network generation method. Graphlets are mostly employed in the field of bioinformatics, typically when analyzing protein-protein interaction (PPI) networks. Therefore, we chose as a target graphlet distribution the one obtained from a PPI network of bacteria E. coli. All the experiments were performed on an average desktop computer (Intel Core 2, 2.67GHz).

The generated network is obviously influenced by our choice of the initial network. We can start with a completely empty network or choose a starting point that already possesses some characteristics of the target network. For example, random geometric graphs (GEO) were demonstrated to be a better model of PPI networks than the previous top choice of scale-free models [1]. However, a starting point with a better distance score does not necessarily lead to a better final score as visualized in Figure 6.1. The initial ER and GEO networks had a similar number of nodes and edges as the target network, while the other PPI network had approximately twice as many nodes and twice as many edges. We do not show the starting scores because they are as high as 12.7 in case of an empty starting network which compresses the lines in the figure and makes it uninformative. The PPI network of another organism (yeast, S. cerevisiae) turns out to be the best choice. Table 6.1 shows the target graphlet counts and the

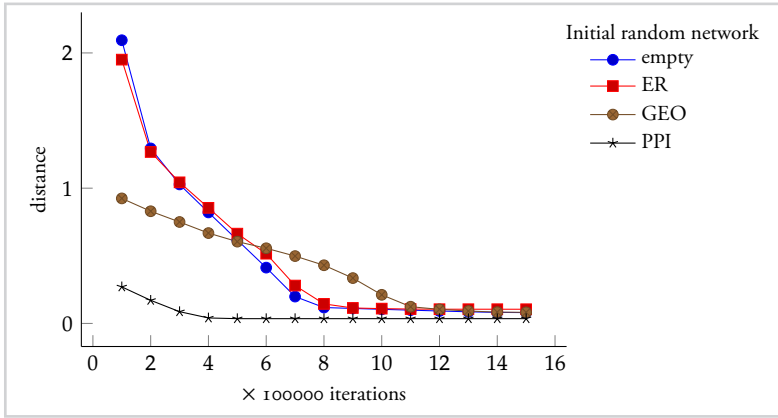


Figure 6.1

Distance convergence for different initial networks.

final graphlet counts obtained from various starting networks after 150000 iterations.

Next, we analyzed what is happening through iterations with individual graphlet counts when starting with a random geometric network, which is supposed to be a good approximation for PPI networks. Figure 6.2 shows graphlet counts on a logarithmic scale, which corresponds to their contribution in the chosen distance measure.

Table 6.1

Final graphlet counts for different initial networks.

source	score	G_0	G_1	G_2	G_3	G_4
target	0.0	11626	418270	18225	11110782	9383348
empty	0.0801	11626	416156	21803	12541517	9382769
ER	0.1048	11626	418252	22967	12797580	9381493
GEO	0.0819	11626	418249	24796	11976090	9379476
PPI	0.0352	11626	424358	21733	11110365	9382954

source	score	G_5	G_6	G_7	G_8
target	0.0	445186	2216446	285736	22376
empty	0.0801	445026	2799969	285720	18673
ER	0.1048	443079	2880211	285734	16514
GEO	0.0819	387973	2749137	285672	22358
PPI	0.0352	445182	2216415	251844	22375

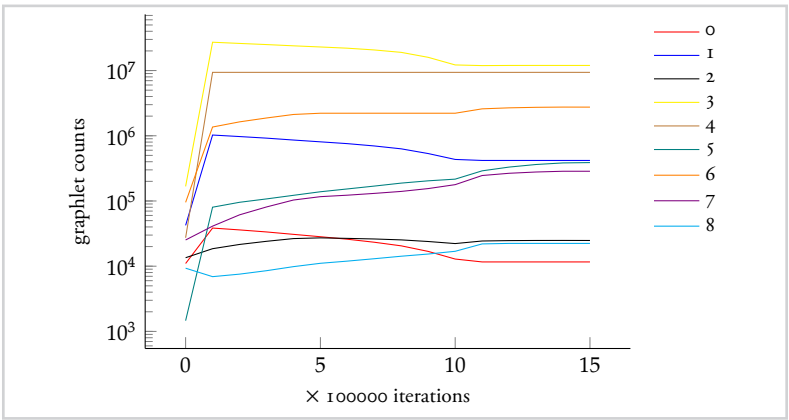


Figure 6.2

Change in individual graphlet counts over iterations starting from a random geometric network.

The initial sharp increase in the number of edges accommodates the difference in the number of most graphlets (Table 6.2). These values then slowly decrease until they settle at around 1.4 million iterations. An exception to this is graphlet G_8 (clique on 4 nodes, K_4). Random geometric graphs already contain a substantial number of cliques, whose number even decreases before it starts to rise to the desired value.

Table 6.2

Graphlet counts obtained by starting with a random geometric network.

GEO	score	G_0	G_1	G_2	G_3	G_4
start	2.761	11000	42251	13492	167375	27308
finish	0.0819	11626	418249	24796	11976090	9379476
target	0.0	11626	418270	18225	11110782	9383348

GEO	score	G_5	G_6	G_7	G_8
start	2.761	1462	95264	25235	9321
finish	0.0819	387973	2749137	285672	22358
target	0.0	445186	2216446	285736	22376

It might seem strange that the initial network contains much fewer graphlets for every graphlet type (except G_0) despite having an almost the same number of edges. However, consider as an example a pair of graphs: S_6 (a star graph with one central and five leaf nodes) and P_6 (a path graph on 6 nodes). If we limit our observation to

3-node graphlets, we see that S_6 contains at least as many graphlets of every type as P_6 despite the same number of nodes and edges.

To compare the speed of different dynamic graphlet counting methods described in previous sections, we implemented three solutions and ran each of them for 100000 iterations. All solutions started with an empty network. They differ only in the graphlet counting method and give the exact same results. The first method (batch) was a straight-forward use of the original ORCA algorithm, which recomputes the graphlet counts from scratch on every iteration. The second method (dynamic) is the adaptation of ORCA for the dynamic problem setting. The final, third method (maintaining), maintains graphlet counts on every change of the network. The last method

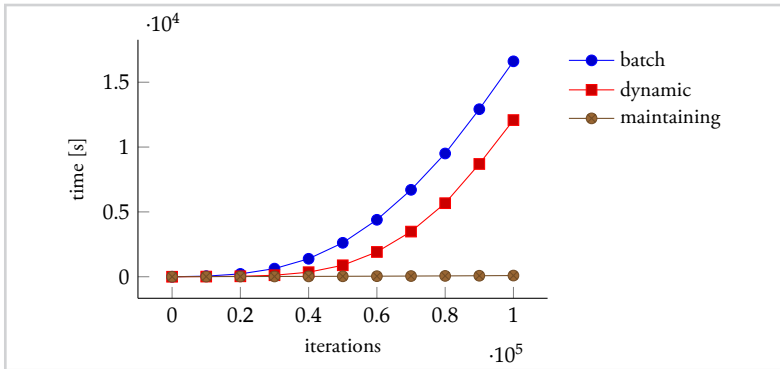


Figure 6.3
Speed comparison.

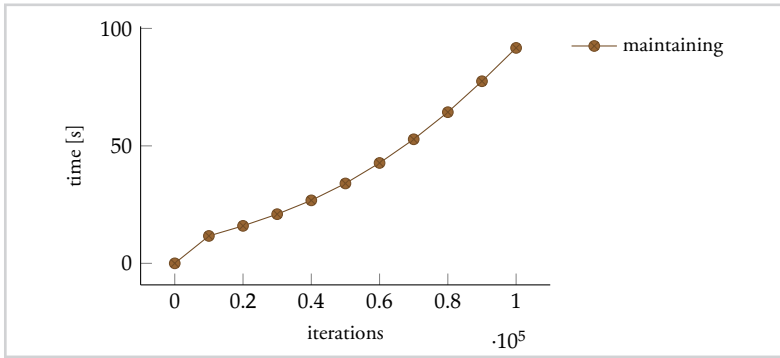


Figure 6.4
Behavior of the fastest method.

is by far the fastest among them (Figures 6.3 and 6.4). As the number of iterations grows, the number of edges in the network increases and so do the times required for computing graphlet counts, which is why the plots are not linear.

Finally, we also analyzed the properties of a random network obtained by our optimization procedure (starting from a random geometric network). Does a network obtain this way match the target network in other features besides the optimized graphlet counts? Similar graphlet counts are supposed to indicate similar local topological structure which should in turn reflect in other local or node-centric statistics. We observed distributions of three such statistics - node degree, clustering coefficient [71] and PageRank [72]. To obtain a distribution we divided the range of computed values for each statistic into 10 buckets and counted how many nodes belong to each of them. Triplets of bars (gray, black, red) correspond to the starting, final and target distribution, respectively. The final distribution of node's clustering coefficients (Figure 6.6) is much closer to the target one than the one we started with. The difference is even more pronounced with node degrees (Figure 6.5) and PageRank (Figure 6.7), where most of the nodes in the initial network have very low values in contrast to the target and final networks.

Another interesting observation is that the similarity of 4-node graphlet counts, which we are trying to optimize, also leads to similar 5-node graphlet counts (Figure 6.8). To determine whether this is always the case and to what degree, would require further research. Intuitively, the number of 2-node graphlets (the number of edges) does not tell us much about the number of 3-node graphlets. On the other hand, 4-node graphlets seem to restrict the space of 5-node graphlets much more. We would conjecture that this influence grows with the size of graphlets.

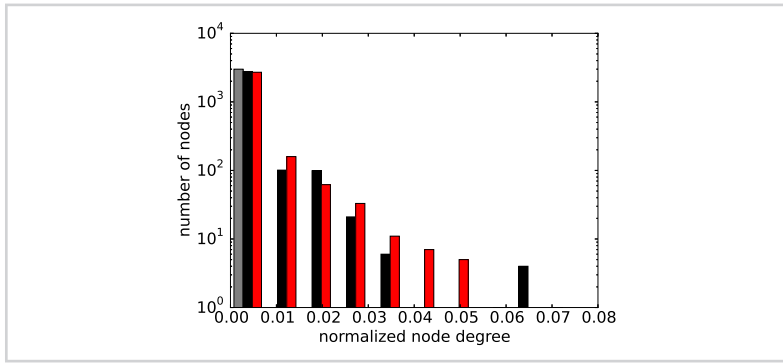


Figure 6.5

Distribution of node degrees normalized by a maximum possible value $n-1$.

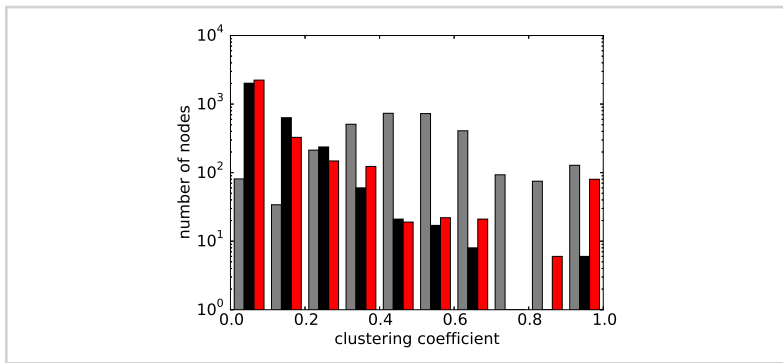


Figure 6.6

Distribution of clustering coefficients.

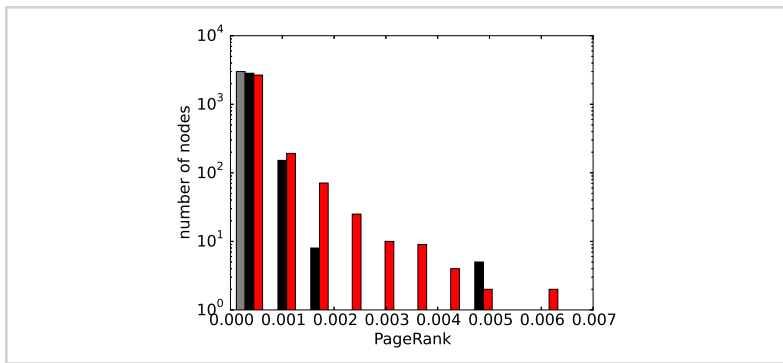


Figure 6.7

Distribution of PageRank.

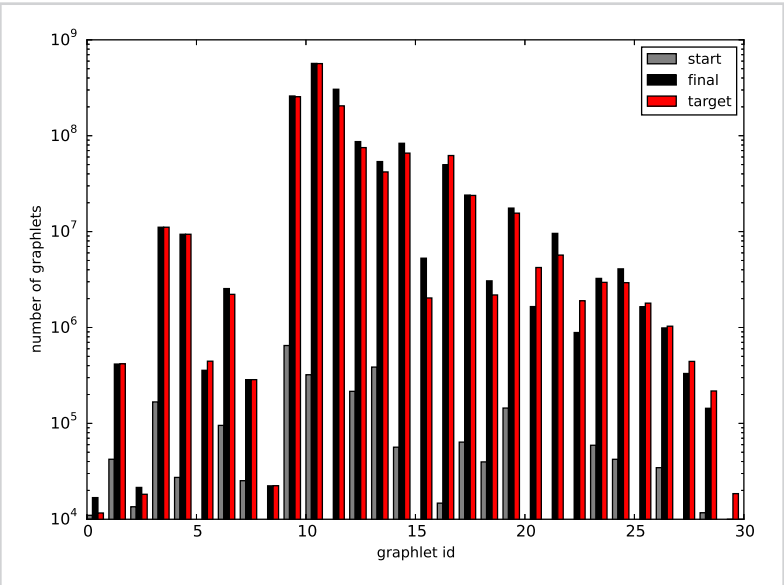


Figure 6.8

5-node graphlet counts obtained by optimizing 4-node graphlet counts.

Conclusion

7

Research in this dissertation presents an improvement over pure enumeration techniques for graphlet counting. In practical terms, the algorithm Orca counts graphlets in large PPI networks 50–100 times faster than other state-of-the-art algorithms. On the largest examined human PPI network we observed a speed-up by a factor of 1800. As such, it is a significant stepping stone towards analyzing larger and denser networks.

7.1 Influence

Orca was incorporated into version 1.02[1] of the GraphCrunch package [20] for graphlet analysis and has been already employed in several research projects. Researchers used our tool to build an Integrated Interactions Database—a database providing tissue-specific protein–protein interactions for model organisms and human [73], design a topology-based distance measure for alignment of networks [74], develop an entity resolution method based on inferred relationships [75], etc.

On the other hand, it also presents a basis for further development of efficient pattern counting methods in graphs. Authors of [41] attempted to generalize the equations used by Orca, which we resolved in [35]. Our 4-node counting approach has been already surpassed by Ortmann et al. [76]. The authors exploited the arboricity of graphs to reduce the run time. As one of the fastest methods, Orca is also used as a benchmark for parallel and approximation algorithms [77, 78].

7.2 Future work

Graphlet analysis is currently used mostly in bioinformatics. By evaluating our methods on a larger and more diverse set of networks from other areas we could demonstrate that there exist efficient tools for obtaining these graphlet-based statistics and expose the graphlet analysis approaches to other fields of research that deal with analysis of networks.

One large unexplored research venue is that of counting weighted graphlets. The edges or relations in networks are often labeled with a number. These numbers can reflect a reliability of an edge or in a more formal mathematical setting the probability of an edge in a graph. Can we use similar techniques as those presented in this dissertation to compute an expected number of graphlets in such probabilistic graphs or does it require a new approach? What about some other interpretation of weights?

[1] http://www0.cs.ucl.ac.uk/staff/natasa/graphcrunch/

The problem might be easier to solve if we limit the weights to a small discrete set, which would result in a different and larger set of orbits. A similar problem is that of counting orbits of directed graphlets [79, 80]. We should be able to successfully apply our graphlet counting approach to both mentioned problems.

By adding the equation $o_{72} = C$, where C is obtained through enumeration of cliques, we get a square system of equations that can be represented in matrix form $\mathbf{A}\mathbf{o} = \mathbf{b}$. $\mathbf{A}$ is a matrix of integer coefficients, $\mathbf{o}$ a vector of orbit counts that we want to determine and $\mathbf{b}$ a vector of computed right sides of equations. We know that the solution $\mathbf{o}$ of the system should consist of nonnegative integral values. Can we use this fact together with some other properties of $\mathbf{A}$ to further speed-up the approach?

Research is never complete and there is still plenty of work to be done in connection with pattern counting in graphs.

Razširjeni povzetek

A

A.1 Uvod

Omrežja se pojavijo na številnih področjih, kjer jih običajno modeliramo z grafi. Majhni vzorci v teh grafih nam lahko pomagajo odkriti osnovne gradnike omrežij, kar vodi do boljšega razumevanja strukture in lastnosti omrežja. Kljub temu pa odkrivanje in štetje vzorcev ostaja računsko zahteven problem. Pogostosti vzorcev lahko ocenimo s pomočjo vzorčenja, vendar nas v tem delu zanima njihovo natančno število. To pa je motivacija za raziskovanje učinkovitih metod štetja vzorcev v redkih omrežjih.

Majhnim povezanim vzorcem običajno rečemo *grafki* [1] in jih v omrežju opazujemo kot inducirane podgrafe. Vozlišča grafkov pa delimo naprej glede na njihove *orbite* [2] oz. ekvivalenčne razrede glede na delovanje grupe avtomorfizmov na vozliščih. Slika 3.1 prikazuje vse grafke s 4 in 5 vozlišči ter njihove orbite. Kolikokrat posamezno vozlišče v omrežju nastopa v vsaki izmed orbit grafkov, predstavlja neke vrste opis strukture lokalne okolice tega vozlišča. Na porazdelitev grafkov in orbit lahko gledamo tudi kot na posplošitev stopnje vozlišča, ki ustreza prvemu netrivialnemu grafku z dvema med seboj povezanima točkama. Podobno orbite povezav [3] prestavljajo neke vrste posplošitev števila skupnih sosedov krajišč povezave oz. števila trikotnikov, ki vključujejo to povezavo, kar je pomembna statistika v analizi omrežij.

Definicija problema. V enostavnem grafu G želimo za vsako vozlišče izračunati, kolikokrat nastopa v vlogi vsake od orbit grafkov velikosti k, ki se v G pojavijo kot inducirani podgrafi. Slika 4.2 ilustrira problem na enostavnem omrežju.

Analiza omrežij z grafki se je do sedaj uporabljala večinoma v bioinformatiki, vendar ni omejena na to področje [9]. Glavni primer uporabe grafkov so omrežja proteinskih interakcij (angl. protein-protein interaction ali PPI). S pomočjo grafkov so poiskali boljše naključne modele PPI omrežij [1], odkrivali skupine proteinov in napovedovali njihove lastnosti [10]. Poleg tega so napovedovali tudi, kateri geni so povezani z rakavimi boleznimi [11] in kateri s procesom staranja [12]. Grafki so nam lahko v pomoč tudi pri reševanju drugih problemov, na primer pri poravnavi omrežij. S tem lahko napovemo biološke funkcije poravnanih proteinov ali rekonstruiramo filogenetsko drevo. Primeri algoritmov za poravnavo omrežij na osnovi grafkov so GRAAL [13], H-GRAAL [14] in MI-GRAAL [16].

V bioinformatiki se je razvilo več orodij za štetje grafkov: FANMOD [17], Graphlet-Counter [18], GraphCrunch [20, 21], RAGE [22]. Vsa omenjena orodja temeljijo na

naštevanju grafkov. Kljub omejitvi analize na vzorce z največ petimi točkami to že predstavlja problem na večjih PPI omrežjih. Na primer, ocenjen čas štetja grafkov s petimi točkami v omrežju proteinskih interakcij v človeku iz podatkovne baze BioGRID [23] je več mesecev. Glede na hitro rast razpoložljive količine podatkov in velikosti omrežij torej potrebujemo nove algoritmične pristope.

Ti pristopi bodo morali temeljiti na štetju namesto na naštevanju. Klasičen rezultat štetja vzorcev v grafih se nanaša na štetje trikotnikov [25]. Kub matrike sosednosti A vsebuje število sprehodov dolžine 3 med pari točk. Skupno število trikotnikov v grafu je torej $\frac{1}{6}\sum_{x\in G} A^3_{x,x}$. Vsak trikotnik namreč štejemo dvakrat pri vsaki izmed njegovih treh točk. Časovna zahtevnost tega postopka je enaka časovni zahtevnosti množenja matrik sosednosti, ki je $o(n^3)$ [27, 28]. Podobno lahko v grafih odkrijemo kliko velikosti k v času $o(n^k)$ s prevedbo problema na odkrivanje trikotnikov [26]. Drugačen pristop k štetju vzorcev je predstavil Kloks [29] s sistemom enačb, ki nam omogočajo izračun števila pojavitev vseh šestih grafkov s štirimi točkami, če poznamo število vsaj enega izmed njih. Časovna zahtevnost izgradnje sistema enačb je enaka množenju kvadratnih matrik velikosti n. Zanimiv rezultat povezuje štetje vseh induciranih in vseh neinduciranih vzorcev. Problema sta namreč ekvivalentna, saj lahko število induciranih vzorcev izračunamo, če poznamo pogostosti vseh neinduciranih, in obratno [69].

Obstoječe metode temeljijo na izčrpnem naštevanju ali pa na hitrem množenju matrik. Pristopi z naštevanjem postajajo prepočasni na večjih omrežjih, pristopi z množenjem matrik pa niso primerni na redkih omrežjih, ki se pojavijo v praksi. Naše raziskovalno delo je bilo usmerjeno k razvoju algoritma, ki izboljša oz. preseže metode naštevanja in je zasnovan za štetje grafkov v redkih omrežjih.

Razviti algoritem temelji na izkoriščanju relacij med orbitami za učinkovito štetje. Sistemi enačb, ki sta jih predstavila Kloks [29] in Kowaluk [30] temeljijo na hitrem množenju matrik in zato niso primerni za redka omrežja. Ključ našega algoritma je v vzpostavitvi sistema enačb, ki ga je v redkih grafih mogoče zgraditi učinkovito.

A.2 Prispevki k znanosti

To poglavje povzema glavne prispevke k znanosti. Pri vsakem prispevku so omenjeni tudi pripadajoči znanstveni članki in poglavja disertacije, ki ga obravnavajo. Sem edini prvi avtor omenjenih člankov, ki so bili objavljeni v uglednih mednarodnih znanstvenih revijah. Vsi članki so bili objavljeni v revijah, ki so citirane v bazi Science Citation

Index (SCI) in so po faktorju vpliva (IF) razvrščene v zgornjo polovico lestvice na vsaj enem strokovnem področju.

- *Algoritem za štetje orbit točk iz grafkov na štirih in petih točkah*

  Razvil sem algoritem Orca za štetje grafkov in orbit. Algoritem je sposoben učinkovitega štetja orbit vozlišč in povezav v grafkih s štirimi ali petimi vozlišči. Prednost razvitega algoritma pred obstoječimi metodami štetja grafkov izhaja iz kombinatoričnega pristopa k štetju namesto enostavnega naštevanja vseh grafkov, ki se pojavijo v omrežju.

  Prispevek je predstavljen v poglavjih 3 in 4, ki vsebujeta preoblikovana članka [33] in [34].

- *Splošen algoritem za štetje orbit točk in povezav*

  Posplošil sem pristop k generiranju sistema enačb, ki predstavlja jedro algoritma Orca, in dokazal, da je mogoče tak sistem enačb sestaviti za poljubno velike grafke in ne le za grafke s štirimi ali petimi vozlišči.

  Prispevek je predstavljen v poglavju 5, ki vsebuje preoblikovan članek [35].

- *Algoritem za sprotno štetje orbit v dinamičnih omrežjih*

  Algoritem Orca sem prilagodil še za problem sprotnega štetja grafkov v omrežjih, kjer se dodajanja in odstranjevanja povezav prepletajo s poizvedbami o številu grafkov, v katerih nastopa določeno vozlišče. Algoritem je neposredno uporaben za generiranje naključnih omrežij z želenimi pogostostmi grafkov.

  V poglavju 6 so predstavljeni še neobjavljeni rezultati na temo dinamičnega štetja grafkov.

A.3 Orca

Razviti algoritem Orca je sposoben štetja orbit grafkov velikosti 4 in 5 in temelji na izkoriščanju relacij med posameznimi orbitami. Naš sistem enačb povezuje število orbit za razliko od drugih rešitev, ki povezujejo zgolj število grafkov ali celo nepovezanih vzorcev. Za štetje grafkov velikosti 5 algoritem našteje zgolj grafke velikosti 4. Enako velja za grafke velikosti 4, kjer algoritem Orca zahteva naštevanje grafkov s samo 3 vozlišči (trikotnik in pot dolžine 2).

Primer enačbe za štetje orbit grafkov velikosti $k = 4$ v vozlišču x je

$$2o_{13} + 6o_{14} = \sum_{y,z:\, y<z, G[\{x,y,z\}]\cong G_2} (c(x,y) - 1) + (c(x,z) - 1).$$

Z o_i označujemo število pojavitev vozlišča v i-ti orbiti, O_i. Podobno G_i predstavlja i-ti grafek. Številčenje grafkov in orbit sledi številčenju ob njihovi vpeljavi [2]. Funkcija $c(x, y)$ predstavlja število skupnih sosedov vozlišč x in y. V nadaljevanju se n nanaša na število vozlišč v omrežju, e na število povezav in d na največjo stopnjo vozlišča. Velikost opazovanih grafkov bomo označevali s k.

Predlagani sistem enačb algoritma Orca dobimo z opazovanjem razširitev manjših vzorcev z dodatno točko. Vse enačbe imajo na levi strani linearno kombinacijo orbit, ki so fiksne (določa jih algoritem), medtem ko so desne strani odvisne od omrežja, v katerem štejemo orbite. Vsota v zgornji enačbi vključuje naštevanje trikotnikov, ki vključujejo vozlišče x. Členi vsote pa vsebujejo števila skupnih sosedov parov točk, ki so predmet predprocesiranja. Enačbe so zasnovane tako, da vključujejo člene, ki jih je mogoče učinkovito izračunati in shraniti za kasnejšo uporabo. Da lahko te enačbe uporabimo, moramo izbrati in učinkovito prešteti eno izmed orbit, medtem ko bomo ostale lahko izračunali s pomočjo prej izpeljanih enačb. Izberemo klike, ki jih je v redkih grafih malo in jih lahko učinkovito naštejemo s katerim od znanih algoritmov [39, 40].

Časovna zahtevnost algoritma Orca za štetje orbit grafkov velikosti k v omrežju z n vozlišči in največjo stopnjo vozlišča d je $O(nd^{k-2} + T_k)$, kjer je T_k čas naštevanja klik velikosti k, ki je v praksi zanemarljiv, saj lahko izkoristimo redkost omrežij. Prostorska zahtevnost algoritma je $O(nd^{k-3})$. Za primerjavo je časovna zahtevnost drugih pristopov z naštevanje $O(nd^{k-1})$. Časovni zahtevnosti se razlikujeta za faktor d, kar se odraža tudi v opravljenih eksperimentih. Razviti algoritem smo primerjali z obstoječimi metodami na omrežjih interakcij proteinov [33]. V večjih PPI omrežjih je algoritem Orca 50 do 100-krat hitrejši od ostalih. Na največjem testiranem omrežju proteinskih interakcij v človeku pa je bila pohitritev kar 1800-kratna (d=9716).

Pristop smo uspešno prilagodili še za štetje orbit povezav. Relacije med orbitami povezav lahko izpeljemo na podoben način kot med orbitami točk. Pričakovana hitrost štetja orbit povezav pa je enaka hitrosti štetja orbit točk. Problem štetja orbit povezav bi sicer lahko prevedli na štetje orbit točk v povezavnem grafu, vendar bi bili tako dobljeni povezavni grafi in iskani vzorci precej večji. Primer enačbe, ki povezuje orbite

povezav v algoritmu Orca, je

$$e_{43} + 2e_{56} + e_{63} = \sum_{\substack{u,v\colon G[\{x,y,u,v\}]\cong G_5 \\ (x,y)\in E \,\wedge\, u\in N(x) \,\wedge\, v\in N(y)}} c(u,v).$$

Celoten seznam uporabljenih enačb je na voljo v dodatku B.

Izvorna koda algoritma v programskem jeziku `C++` je prosto dostopna na strani `https://github.com/thocevar/orca`. Poleg tega je na voljo tudi v obliki paketa za programski jezik `R` na repozitoriju CRAN[1].

A.4 Večji grafki

Posplošitev algoritma na štetje orbit grafkov poljubnih velikosti je zanimiva predvsem s teoretičnega vidika. Število obstoječih grafkov namreč hitro narašča, izračunane porazdelitve pa so redke in manj uporabne v praksi.

Enačbe uporabljene v Orci so bile prvotno pridobljene z obravnavo vseh možnih razširitev grafkov velikost $k-1$ z novo točko. Vsaka razširitev vodi do ene enačbe. Med njimi smo nato izbrali veliko množico med seboj neodvisnih enačb, ki jim je mogoče učinkovito izračunati desne strani, ki so odvisne od omrežja. Za posplošitev algoritma smo morali ubrati bolj sistematičen pristop. V vsakem grafku smo za vsako vozlišče x poiskali neko drugo vozlišče y, katerega odstranitev vodi do manjšega grafka, ki ga je nato mogoče učinkovito razširiti z vozliščem y nazaj v začetni grafek. Tako dobimo trikoten sistem enačb, ki so zagotovo neodvisne.

Analiza algoritma pokaže, da mora za učinkovit izračun števila skupnih sosedov v vsakem grafku za vsako vozlišče x obstajati neko drugo vozlišče y, za katerega velja:

1. $d(y) \leq k-2$,
2. $G \setminus \{y\}$ je povezan graf,
3. če je $d(y) = k-2$, sosedi vozlišča y inducirajo povezan podgraf.

Z obravnavo šestih primerov (Slika 5.7) lahko dokažemo, da skoraj vedno obstaja vozlišče y, ki zadošča zgornjim pogojem. Prva izjema je poln grafek, ki ni relevanten za izgradnjo enačb, saj polne grafke preštejemo neposredno. Druga izjema pa je cikel

[1] `https://CRAN.R-project.org/package=orca`

na štirih vozliščih $G_5 = C_4$, ki ga Orca obravnava ločeno brez negativnega vpliva na časovno zahtevnost. Dokaz vodi tudi do enostavnega pravila za izbiro vozlišča y. Med vozlišči, ki so najbolj oddaljena od vozlišča x, izberemo tisto z najmanjšo stopnjo. Če to vozlišče ne zadošča pogojem, jim bo zagotovo ustrezalo vozlišče z najmanjšo stopnjo med vozlišči, katerih razdalja do x je za 1 manjša.

Pričakovana časovna zahtevnost algoritma v naključnih Erdős-Rényi (ER) grafih je odvisna od pričakovanega števila povezanih podgrafov velikosti $k - 1$. V ER grafu z n vozlišči in verjetnostjo povezave p je ta časovna zahtevnost enaka $O(n^{k-1}p^{k-2})$ za konstanten k. Odvisnost naše ocene časovne zahtevnosti od verjetnosti povezav p smo potrdili tudi eksperimentalno (Slika 5.9).

A.5 Sprotno štetje in naključna omrežja

Omrežja, ki izhajajo z različnih področij, imajo različne lastnosti. Če želimo testirati nek postopek na omrežjih z nekega področja, običajno identificiramo njihove tipične lastnosti, nato pa bi želeli generirati več takih naključnih omrežij. Najenostavnejši model naključnih grafov je Erdős-Rényi (ER) model [58]. Žal pa tak model ne zajame lastnosti, ki jih opazimo v številnih omrežjih v praksi. Za številna omrežja je npr. značilna potenčna porazdelitev stopenj točk, ki jih lahko generiramo z Barabási-Albert (BA) modelom [59] po principu prednostne povezanosti (angl. preferential attachment).

Očitna osnovna lastnost omrežja je porazdelitev stopenj vozlišč. Algoritem Havel-Hakimi [60] z enostavnim požrešnim pristopom učinkovito sestavi ustrezen graf ali ugotovi, da to ni mogoče. Generiranje enakomerno porazdeljenih enostavnih grafov se izkaže za precej trši oreh. Naključne grafe lahko generiramo z neenakomerno porazdelitvijo in jih naknadno temu primerno utežimo [61]. Druga družina pristopov [62] začne s poljubnim ustreznim grafom, ki ga nato naključno spreminja, pri čemer ostajajo stopnje vozlišč ves čas nespremenjene. Tretji pristop vključuje občasno resetiranje procesa generiranja naključnega grafa [63, 64].

Pojavitve grafkov s štirimi vozlišči so še natančnejši opis strukture omrežja kot stopnje točk. Generiranja naključnih grafov, ki se čim bolj približajo želeni porazdelitvi, smo se lotili z enostavnim iterativnim optimizacijskim postopkom. Na vsakem koraku naredimo manjšo lokalno spremembo (dodamo ali odstranimo naključno povezavo) in preverimo, ali smo se s tem kaj približali rešitvi. Uporabljena mera podobnosti med

porazdelitvijo grafkov v trenutnem grafu G in ciljnem grafu H temelji na članku [1]:

$$D(G,H) = \sum_{i=0}^{8} |F_i(G) - F_i(H)|$$

$$F_i(G) = \log(N_i(G) + 1),$$

kjer $N_i(G)$ predstavlja število pojavitev grafka i v grafu G.

Pogosto štetje grafkov v optimizacijskem postopku je bila motivacija za prilagoditev algoritma Orca za dinamičen problem. Dinamično štetje grafkov sestoji iz operacij:

- dodajanja povezave (x, y)
- odstranjevanja povezave (x, y)
- štetja pojavitev vozlišča x v orbitah grafkov

Ključna naloga je vzdrževanje pravilne statistike števila skupnih sosedov $c(x, y)$, na katero vplivata prvi dve operaciji. Izkaže se, da lahko vrednosti v okolici spremenjene povezave učinkovito popravimo. Dinamičen algoritem Orca, ki začne s praznim grafom, mu doda vse povezave, na koncu pa prešteje orbite, ima enako časovno zahtevnost kot osnovni algoritem, ki deluje na celotnem grafu.

Tudi uporaba dinamičnega algoritma je časovno prepotratna za generiranje večjih omrežij. Sprememba povezave (x, y) namreč vpliva na rezultate štetja orbit v vozliščih, ki so od x ali y oddaljena največ 2. Vsaka poizvedba o številu orbit, v katerih nastopa posamezno vozlišče, prav tako obišče vsa vozlišča na razdalji 2. Za izračun novega števila grafkov torej ob vsaki spremembi obiščemo okolico na razdalji 4, kar pogosto predstavlja večino grafa in s tem ne pridobimo veliko v primerjavi z vnovičnim izvajanjem (statičnega) algoritma od začetka. Zato smo postopek optimizirali še za sprotno vzdrževanje števila grafkov ob vsaki spremembi povezave. Algoritem namesto vrednosti $c(x, y)$ neposredno popravlja vrednosti uporabljene v sistemu enačb. Tako zreducira obiskano področje grafa na vozlišča na razdalji 2 od x ali y, kar se odraža v izraziti pohitritvi (Slika 6.3).

Ker se grafki uporabljajo večinoma pri analizi omrežij, ki izhajajo iz bioinformatike, smo v eksperimentih za tarčno distribucijo izbrali število grafkov v PPI omrežju bakterije *E. coli*. Poleg meritev hitrosti razvitih algoritmov smo primerjali tudi vpliv izhodiščnega omrežja na končni približek (Tabela 6.1). Bolj kot je začetno omrežje

strukturno podobno ciljnemu, boljši približek bi pričakovali. Najboljši rezultat dosežemo z uporabo PPI omrežja drugega organizma (kvasovka *S. cerevisiae*). Presenetljivo nato sledi prazno omrežje z malenkostno prednostjo pred naključnim geometrijskim omrežjem, ki sicer velja za dober model PPI omrežij. Najslabša izhodiščna izbira je bil naključen ER graf. Poleg števila grafkov smo primerjali tudi, kakšne so porazdelitve drugih strukturnih lastnosti v generiranih naključnih omrežjih. Glede na izhodiščno naključno geometrijsko omrežje imajo končna omrežja bolj podobne porazdelitve stopenj točk, koeficientov gručenja (angl. clustering coefficient) in mere PageRank (Slike 6.5, 6.6, 6.7). Kot je razvidno s slike 6.8, je presenetljivo podobno tudi število grafkov s petimi točkami kljub optimizaciji števila pojavitev grafkov s štirimi.

A.6 Zaključek

Raziskovalno delo predstavljeno v tej disertaciji izkazuje očiten napredek štetje grafkov v primerjavi z metodami naštevanja. Algoritem Orca je v praksi 50 do 100-krat hitrejši pri štetju grafkov v večjih PPI omrežjih. Na največjem testiranem omrežju proteinskih interakcij v človeku pa je bila pohitritev kar 1800-kratna. Razviti algoritem tako omogoča analizo večjih in gostejših omrežij.

Orca je bila vgrajena v programski paket GraphCrunch [20], ki je namenjen analizi omrežij na osnovi grafkov. Poleg tega je bil algoritem že vključen v več raziskavah. Uporabili so ga pri izgradnji podatkovne baze proteinskih interakcij v različnih tkivih vzorčnih organizmov in človeka (Integrated Interactions Database [73]), za razvoj metode poravnave omrežij na osnovi njihove lokalne strukture [74], za določanje entitet na podlagi njihovih relacij z drugimi entitetami [75], itd.

Po bolj teoretični plati je naš algoritem osnova za nadaljnji razvoj učinkovitih metod štetja vzorcev v grafih. Avtorji članka [41] so poskušali posplošiti enačbe, ki jih uporablja Orca. Ortmann [76] je s svojim algoritmom, ki izkorišča nizko drevesnost (angl. arboricity) omrežij, že izboljšal našo rešitev štetja grafkov s štirimi vozlišči. Kot ena najhitrejših metod predstavlja Orca tudi osnovo za primerjavo pri razvoju vzporednih in aproksimacijskih algoritmov [77, 78].

Raziskovalno delo ni nikoli zaključeno in na področju štetja vzorcev v grafih ga je še precej.

Graphlet equations

B

B.1 Equations for node-orbit counts in 4-graphlets

Let $p(u,v)$ denote the number of paths on three nodes that start at node u, continue with v and end with some node t, which is not connected to u. We can compute $p(u,v)$ as $p(u,v) = deg(v) - 1 - c(u,v)$.

$$o_{12} + 3o_{14} = \sum_{y,z:\, y<z, G[\{x,y,z\}]\cong G_2} c(y,z) - 1$$

$$2o_{13} + 6o_{14} = \sum_{y,z:\, y<z, G[\{x,y,z\}]\cong G_2} (c(x,y) - 1) + (c(x,z) - 1)$$

$$o_{10} + 2o_{13} = \sum_{y,z:\, y<z, G[\{x,y,z\}]\cong G_2} p(y,z) + p(z,y)$$

$$2o_{11} + 2o_{13} = \sum_{y,z:\, y<z, G[\{x,y,z\}]\cong G_2} p(y,x) + p(z,x)$$

$$6o_{7} + 2o_{11} = \sum_{y,z:\, y<z, y,z\in N(x), G[\{x,y,z\}]\cong G_1} (p(y,x) - 1) + (p(z,x) - 1)$$

$$o_{5} + 2o_{8} = \sum_{y,z:\, y<z, y,z\in N(x), G[\{x,y,z\}]\cong G_1} p(x,y) + p(x,z)$$

$$2o_{6} + 2o_{9} = \sum_{y,z:\, x,z\in N(y), G[\{x,y,z\}]\cong G_1} p(x,y) - 1$$

$$2o_{9} + 2o_{12} = \sum_{y,z:\, x,z\in N(y), G[\{x,y,z\}]\cong G_1} c(y,z)$$

$$o_{4} + 2o_{8} = \sum_{y,z:\, x,z\in N(y), G[\{x,y,z\}]\cong G_1} p(y,z)$$

$$2o_{8} + 2o_{12} = \sum_{y,z:\, x,z\in N(y), G[\{x,y,z\}]\cong G_1} c(x,z) - 1$$

B.2 Equations for edge-orbit counts in 4-graphlets

$$2e_{10} + 2e_{11} = \sum_{z:\, G[\{x,y,z\}]\cong G_2} (c(x,y) - 1)$$

$$e_{9} + 4e_{11} = \sum_{z:\, G[\{x,y,z\}]\cong G_2} (c(x,z) + c(y,z) - 2)$$

$$e_8 + e_9 + 4e_{10} + 4e_{11} = \sum_{z:\, G[\{x,y,z\}] \cong G_2} (c(x) + c(y) - 4)$$

$$e_7 + e_9 + 2e_{11} = \sum_{z:\, G[\{x,y,z\}] \cong G_2} (c(z) - 2)$$

$$2e_6 + e_9 = \sum_{z:\, z \in N(y) \wedge G[\{x,y,z\}] \cong G_1} c(y,z) + \sum_{z:\, z \in N(x) \wedge G[\{x,y,z\}] \cong G_1} c(x,z)$$

$$2e_5 + e_9 = \sum_{z:\, z \in N(y) \wedge G[\{x,y,z\}] \cong G_1} (c(x,z) - 1) +$$

$$\sum_{z:\, z \in N(x) \wedge G[\{x,y,z\}] \cong G_1} (c(y,z) - 1)$$

$$2e_4 + 2e_6 + e_8 + e_9 = \sum_{z:\, z \in N(y) \wedge G[\{x,y,z\}] \cong G_1} (c(y) - 2) +$$

$$\sum_{z:\, z \in N(x) \wedge G[\{x,y,z\}] \cong G_1} (c(x) - 2)$$

$$2e_3 + 2e_5 + e_8 + e_9 = \sum_{z:\, z \in N(y) \wedge G[\{x,y,z\}] \cong G_1} (c(x) - 1) +$$

$$\sum_{z:\, z \in N(x) \wedge G[\{x,y,z\}] \cong G_1} (c(y) - 1)$$

$$e_2 + 2e_5 + 2e_6 + e_9 = \sum_{z:\, z \in N(y) \cup N(x) \wedge G[\{x,y,z\}] \cong G_1} (c(z) - 1)$$

B.3 Equations for node-orbit counts in 5-graphlets

Conditions, P_i, define the order of nodes and put x in orbit O_i; e.g., in P_{13} node x is in orbit O_{13}.

$$P_{14}(x,u,v,t) = u < v < t \wedge G[\{x,u,v,t\}] \cong G_8$$
$$P_{13}(x,u,v,t) = v < t \wedge (v,t) \notin E \wedge G[\{x,u,v,t\}] \cong G_7$$
$$P_{12}(x,u,v,t) = u < v \wedge (x,t) \notin E \wedge G[\{x,u,v,t\}] \cong G_7$$
$$P_{11}(x,u,v,t) = u < v \wedge u,v \notin N(t) \wedge G[\{x,u,v,t\}] \cong G_6$$
$$P_{10}(x,u,v,t) = x,u \notin N(t) \wedge G[\{x,u,v,t\}] \cong G_6$$

$$P_9(x,u,v,t) = v < t \wedge v,t \notin N(x) \wedge G[\{x,u,v,t\}] \cong G_6$$
$$P_8(x,u,v,t) = u < v \wedge u,v \in N(x) \wedge G[\{x,u,v,t\}] \cong G_5$$
$$P_7(x,u,v,t) = u < v < t \wedge u,v,t \in N(x) \wedge G[\{x,u,v,t\}] \cong G_4$$
$$P_6(x,u,v,t) = v < t \wedge x,v,t \in N(u) \wedge G[\{x,u,v,t\}] \cong G_4$$
$$P_5(x,u,v,t) = u,v \in N(x) \wedge t \in N(v) \wedge G[\{x,u,v,t\}] \cong G_3$$
$$P_4(x,u,v,t) = x,v \in N(u) \wedge t \in N(v) \wedge G[\{x,u,v,t\}] \cong G_3$$

Equations:

$$2o_{71} + 12o_{72} = \sum_{u,v,t:\, P_{14}(x,u,v,t)} c(x,u,v) + c(x,u,t) + c(x,v,t) - 3$$
$$o_{70} + 4o_{72} = \sum_{u,v,t:\, P_{14}(x,u,v,t)} c(u,v,t) - 1$$
$$4o_{69} + 2o_{71} = \sum_{u,v,t:\, P_{13}(x,u,v,t)} c(x,v,t) - 1$$
$$o_{68} + 2o_{71} = \sum_{u,v,t:\, P_{13}(x,u,v,t)} c(u,v,t) - 1$$
$$o_{67} + 12o_{72} + 4o_{71} = \sum_{u,v,t:\, P_{14}(x,u,v,t)} (c(x,u) - 2) + (c(x,v) - 2) + (c(x,t) - 2)$$
$$o_{66} + 12o_{72} + 2o_{71} + 3o_{70} = \sum_{u,v,t:\, P_{14}(x,u,v,t)} (c(u,v) - 2) + (c(u,t) - 2) + (c(v,t) - 2)$$
$$2o_{65} + 3o_{70} = \sum_{u,v,t:\, P_{12}(x,u,v,t)} c(u,v,t)$$
$$o_{64} + 2o_{71} + 4o_{69} + o_{68} = \sum_{u,v,t:\, P_{13}(x,u,v,t)} c(v,t) - 2$$
$$o_{63} + 3o_{70} + 2o_{68} = \sum_{u,v,t:\, P_{12}(x,u,v,t)} c(x,t) - 2$$
$$2o_{62} + o_{68} = \sum_{u,v,t:\, P_{8}(x,u,v,t)} c(u,v,t)$$
$$2o_{61} + 4o_{71} + 8o_{69} + 2o_{67} = \sum_{u,v,t:\, P_{13}(x,u,v,t)} (c(x,v) - 1) + (c(x,t) - 1)$$

$$o_{60} + 4o_{71} + 2o_{68} + 2o_{67} = \sum_{u,v,t:\, P_{13}(x,u,v,t)} (c(u,v) - 1) + (c(u,t) - 1)$$

$$o_{59} + 6o_{70} + 2o_{68} + 4o_{65} = \sum_{u,v,t:\, P_{12}(x,u,v,t)} (c(u,t) - 1) + (c(v,t) - 1)$$

$$o_{58} + 4o_{72} + 2o_{71} + o_{67} = \sum_{u,v,t:\, P_{14}(x,u,v,t)} c(x) - 3$$

$$o_{57} + 12o_{72} + 4o_{71} +$$
$$3o_{70} + o_{67} + 2o_{66} = \sum_{u,v,t:\, P_{14}(x,u,v,t)} (c(u) - 3) + (c(v) - 3) + (c(t) - 3)$$

$$3o_{56} + 2o_{65} = \sum_{u,v,t:\, P_{9}(x,u,v,t)} c(u,v,t)$$

$$3o_{55} + 2o_{71} + 2o_{67} = \sum_{u,v,t:\, P_{13}(x,u,v,t)} c(x,u) - 2$$

$$2o_{54} + 3o_{70} + o_{66} + 2o_{65} = \sum_{u,v,t:\, P_{12}(x,u,v,t)} c(u,v) - 2$$

$$o_{53} + 2o_{68} + 2o_{64} + 2o_{63} = \sum_{u,v,t:\, P_{8}(x,u,v,t)} c(x,u) + c(x,v)$$

$$2o_{52} + 2o_{66} + 2o_{64} + o_{59} = \sum_{u,v,t:\, P_{10}(x,u,v,t)} c(u,t) - 1$$

$$o_{51} + 2o_{68} + 2o_{63} + 4o_{62} = \sum_{u,v,t:\, P_{8}(x,u,v,t)} c(u,t) + c(t,v)$$

$$3o_{50} + o_{68} + 2o_{63} = \sum_{u,v,t:\, P_{8}(x,u,v,t)} c(x,t) - 2$$

$$2o_{49} + o_{68} + o_{64} + 2o_{62} = \sum_{u,v,t:\, P_{8}(x,u,v,t)} c(u,v) - 2$$

$$o_{48} + 4o_{71} + 8o_{69} + 2o_{68} +$$
$$2o_{67} + 2o_{64} + 2o_{61} + o_{60} = \sum_{u,v,t:\, P_{13}(x,u,v,t)} (c(v) - 2) + (c(t) - 2)$$

$$o_{47} + 3o_{70} + 2o_{68} + o_{66} +$$
$$o_{63} + o_{60} = \sum_{u,v,t:\, P_{12}(x,u,v,t)} c(x) - 2$$

$$o_{46} + 3o_{70} + 2o_{68} + 2o_{65} + o_{63} + o_{59} = \sum_{u,v,t:\, P_{12}(x,u,v,t)} c(t) - 2$$

$$o_{45} + 2o_{65} + 2o_{62} + 3o_{56} = \sum_{u,v,t:\, P_{9}(x,u,v,t)} c(v,t) - 1$$

$$4o_{44} + o_{67} + 2o_{61} = \sum_{u,v,t:\, P_{11}(x,u,v,t)} c(x,t)$$

$$2o_{43} + 2o_{66} + o_{60} + o_{59} = \sum_{u,v,t:\, P_{10}(x,u,v,t)} c(v,t)$$

$$o_{42} + 2o_{71} + 4o_{69} + 2o_{67} + 2o_{61} + 3o_{55} = \sum_{u,v,t:\, P_{13}(x,u,v,t)} c(x) - 3$$

$$o_{41} + 2o_{71} + o_{68} + 2o_{67} + o_{60} + 3o_{55} = \sum_{u,v,t:\, P_{13}(x,u,v,t)} c(u) - 3$$

$$o_{40} + 6o_{70} + 2o_{68} + 2o_{66} + 4o_{65} + o_{60} + o_{59} + 4o_{54} = \sum_{u,v,t:\, P_{12}(x,u,v,t)} (c(u) - 3) + (c(v) - 3)$$

$$2o_{39} + 4o_{65} + o_{59} + 6o_{56} = \sum_{u,v,t:\, P_{9}(x,u,v,t)} (c(u,v) - 1) + (c(u,t) - 1)$$

$$o_{38} + o_{68} + o_{64} + 2o_{63} + o_{53} + 3o_{50} = \sum_{u,v,t:\, P_{8}(x,u,v,t)} c(x) - 2$$

$$o_{37} + 2o_{68} + 2o_{64} + 2o_{63} + 4o_{62} + o_{53} + o_{51} + 4o_{49} = \sum_{u,v,t:\, P_{8}(x,u,v,t)} (c(u) - 2) + (c(v) - 2)$$

$$o_{36} + o_{68} + 2o_{63} + 2o_{62} + o_{51} + 3o_{50} = \sum_{u,v,t:\, P_{8}(x,u,v,t)} c(t) - 2$$

$$2o_{35} + o_{59} + 2o_{52} + 2o_{45} = \sum_{u,v,t:\, P_{4}(x,u,v,t)} c(u,t) - 1$$

$$2o_{34} + o_{59} + 2o_{52} + o_{51} = \sum_{u,v,t:\, P_4(x,u,v,t)} c(x,t)$$

$$2o_{33} + o_{67} + 2o_{61} + 3o_{58} + 4o_{44} + 2o_{42} = \sum_{u,v,t:\, P_{11}(x,u,v,t)} c(x) - 3$$

$$2o_{32} + 2o_{66} + o_{60} + o_{59} + 2o_{57} + 2o_{43} + 2o_{41} + o_{40} = \sum_{u,v,t:\, P_{10}(x,u,v,t)} c(v) - 3$$

$$o_{31} + 2o_{65} + o_{59} + 3o_{56} + o_{43} + 2o_{39} = \sum_{u,v,t:\, P_9(x,u,v,t)} c(u) - 3$$

$$o_{30} + o_{67} + o_{63} + 2o_{61} + o_{53} + 4o_{44} = \sum_{u,v,t:\, P_{11}(x,u,v,t)} c(t) - 1$$

$$o_{29} + 2o_{66} + 2o_{64} + o_{60} + o_{59} + o_{53} + 2o_{52} + 2o_{43} = \sum_{u,v,t:\, P_{10}(x,u,v,t)} c(t) - 1$$

$$o_{28} + 2o_{65} + 2o_{62} + o_{59} + o_{51} + o_{43} = \sum_{u,v,t:\, P_9(x,u,v,t)} c(x) - 1$$

$$2o_{27} + o_{59} + o_{51} + 2o_{45} = \sum_{u,v,t:\, P_4(x,u,v,t)} c(v,t)$$

$$o_{26} + 2o_{67} + 2o_{63} + 2o_{61} + 6o_{58} + o_{53} + 2o_{47} + 2o_{42} = \sum_{u,v,t:\, P_{11}(x,u,v,t)} (c(u) - 2) + (c(v) - 2)$$

$$2o_{25} + 2o_{66} + 2o_{64} + o_{59} + 2o_{57} + 2o_{52} + o_{48} + o_{40} = \sum_{u,v,t:\, P_{10}(x,u,v,t)} (c(u) - 2)$$

$$o_{24} + 4o_{65} + 4o_{62} + o_{59} + 6o_{56} + o_{51} + 2o_{45} + 2o_{39} = \sum_{u,v,t:\, P_9(x,u,v,t)} (c(v) - 2) + (c(t) - 2)$$

$$4o_{23} + o_{55} + o_{42} + 2o_{33} = \sum_{u,v,t:\, P_7(x,u,v,t)} c(x) - 3$$

$$3o_{22} + 2o_{54} + o_{40} + o_{39} + o_{32} + 2o_{31} = \sum_{u,v,t:\, P_6(x,u,v,t)} c(u) - 3$$

$$o_{21} + 3o_{55} + 3o_{50} + 2o_{42} + 2o_{38} + 2o_{33} = \sum_{u,v,t:\, P_7(x,u,v,t)} (c(u)-1) + (c(v)-1) + (c(t)-1)$$

$$o_{20} + 2o_{54} + 2o_{49} + o_{40} + o_{37} + o_{32} = \sum_{u,v,t:\, P_6(x,u,v,t)} c(x) - 1$$

$$o_{19} + 4o_{54} + 4o_{49} + o_{40} + 2o_{39} + o_{37} + 2o_{35} + 2o_{31} = \sum_{u,v,t:\, P_6(x,u,v,t)} (c(v)-1) + (c(t)-1)$$

$$2o_{18} + o_{59} + o_{51} + 2o_{46} + 2o_{45} + 2o_{36} + 2o_{27} + o_{24} = \sum_{u,v,t:\, P_4(x,u,v,t)} c(v) - 2$$

$$2o_{17} + o_{60} + o_{53} + o_{51} + o_{48} + o_{37} + 2o_{34} + 2o_{30} = \sum_{u,v,t:\, P_5(x,u,v,t)} c(u) - 1$$

$$o_{16} + o_{59} + 2o_{52} + o_{51} + 2o_{46} + 2o_{36} + 2o_{34} + o_{29} = \sum_{u,v,t:\, P_4(x,u,v,t)} c(x) - 1$$

$$o_{15} + o_{59} + 2o_{52} + o_{51} + 2o_{45} + 2o_{35} + 2o_{34} + 2o_{27} = \sum_{u,v,t:\, P_4(x,u,v,t)} c(t) - 1$$

B.4 Equations for edge-orbit counts in 5-graphlets

Conditions, P_i, define the order of nodes and put edge (x, y) in orbit E_i; e.g., in P_{13} edge (x, y) is in orbit E_{13}.

$$P_{11}(x,y,a,b) = a < b \wedge G[\{x,y,a,b\}] \cong G_8$$
$$P_{10}(x,y,a,b) = a < b \wedge (a,b) \notin E \wedge G[\{x,y,a,b\}] \cong G_7$$
$$P_{9a}(x,y,a,b) = a \in N(x) \wedge b \in N(y) \wedge (a,b) \in E \wedge (x,b) \in E \wedge G[\{x,y,a,b\}] \cong G_7$$
$$P_{9b}(x,y,a,b) = a \in N(x) \wedge b \in N(y) \wedge (a,b) \in E \wedge (y,a) \in E \wedge G[\{x,y,a,b\}] \cong G_7$$
$$P_{7}(x,y,a,b) = a \in N(x) \cap N(y) \wedge b \in N(a) \wedge G[\{x,y,a,b\}] \cong G_6$$
$$P_{6a}(x,y,a,b) = a < b \wedge (a,b) \in E \wedge a,b \in N(y) \wedge a,b \notin N(x) \wedge G[\{x,y,a,b\}] \cong G_6$$
$$P_{6b}(x,y,a,b) = a < b \wedge (a,b) \in E \wedge a,b \in N(x) \wedge a,b \notin N(y) \wedge G[\{x,y,a,b\}] \cong G_6$$
$$P_{5}(x,y,a,b) = a \in N(x) \wedge b \in N(y) \wedge G[\{x,y,a,b\}] \cong G_5$$
$$P_{4a}(x,y,a,b) = a < b \wedge a,b \in N(y) \wedge G[\{x,y,a,b\}] \cong G_4$$
$$P_{4b}(x,y,a,b) = a < b \wedge a,b \in N(x) \wedge G[\{x,y,a,b\}] \cong G_4$$
$$P_{3}(x,y,a,b) = a \in N(x) \wedge b \in N(y) \wedge G[\{x,y,a,b\}] \cong G_3$$
$$P_{2a}(x,y,a,b) = (a,b) \in E \wedge a \in N(y) \wedge G[\{x,y,a,b\}] \cong G_3$$
$$P_{2b}(x,y,a,b) = (a,b) \in E \wedge a \in N(x) \wedge G[\{x,y,a,b\}] \cong G_3$$

Equations:

$$2e_{66} + 6e_{67} = \sum_{a,b:\, P_{11}(x,y,a,b)} (c(x,y,a) + c(x,y,b) - 2)$$
$$e_{65} + 6e_{67} = \sum_{a,b:\, P_{11}(x,y,a,b)} (c(x,a,b) + c(y,a,b) - 2)$$
$$e_{64} + 2e_{66} = \sum_{a,b:\, P_{10}(x,y,a,b)} (c(x,a,b) + c(y,a,b) - 2)$$
$$2e_{63} + 2e_{65} = \sum_{a,b:\, P_{9a}(x,y,a,b)} (c(y,a,b) - 1) + \sum_{a,b:\, P_{9b}(x,y,a,b)} (c(x,a,b) - 1)$$
$$e_{62} + 2e_{66} + 3e_{67} = \sum_{a,b:\, P_{11}(x,y,a,b)} (c(x,y) - 2)$$
$$e_{61} + 2e_{65} + 4e_{66} + 12e_{67} = \sum_{a,b:\, P_{11}(x,y,a,b)} (c(x,a) + c(x,b) + c(y,a) + c(y,b) - 8)$$

$$e_{60} + e_{65} + 3e_{67} = \sum_{a,b:\, P_{11}(x,y,a,b)} (c(a,b) - 2)$$

$$2e_{59} + 2e_{65} = \sum_{a,b:\, P_{9a}(x,y,a,b)} c(x,a,b) + \sum_{a,b:\, P_{9b}(x,y,a,b)} c(y,a,b)$$

$$e_{58} + e_{64} + e_{66} = \sum_{a,b:\, P_{10}(x,y,a,b)} (c(a,b) - 2)$$

$$e_{57} + 2e_{63} + 2e_{64} + 2e_{65} = \sum_{a,b:\, P_{9a}(x,y,a,b)} (c(y,a) - 2) + \sum_{a,b:\, P_{9b}(x,y,a,b)} (c(x,b) - 2)$$

$$2e_{56} + 2e_{63} = \sum_{a,b:\, P_{5}(x,y,a,b)} (c(x,a,b) + c(y,a,b))$$

$$e_{55} + 4e_{62} + 2e_{64} + 4e_{66} = \sum_{a,b:\, P_{10}(x,y,a,b)} (c(x,a) + c(x,b) + c(y,a) + c(y,b) - 4)$$

$$2e_{54} + e_{61} + 2e_{63} + 2e_{65} = \sum_{a,b:\, P_{9a}(x,y,a,b)} (c(y,b) - 1) + \sum_{a,b:\, P_{9b}(x,y,a,b)} (c(x,a) - 1)$$

$$e_{53} + 2e_{59} + 2e_{64} + 2e_{65} = \sum_{a,b:\, P_{9a}(x,y,a,b)} (c(x,a) - 1) + \sum_{a,b:\, P_{9b}(x,y,a,b)} (c(y,b) - 1)$$

$$e_{52} + 2e_{59} + 2e_{63} + 2e_{65} = \sum_{a,b:\, P_{9a}(x,y,a,b) \vee P_{9b}(x,y,a,b)} (c(a,b) - 1)$$

$$e_{51} + e_{61} + 2e_{62} + e_{65} + 4e_{66} + 6e_{67} = \sum_{a,b:\, P_{11}(x,y,a,b)} (c(x) + c(y) - 6)$$

$$e_{50} + 2e_{60} + e_{61} + 2e_{65} + 2e_{66} + 6e_{67} = \sum_{a,b:\, P_{11}(x,y,a,b)} (c(a) + c(b) - 6)$$

$$3e_{49} + e_{59} = \sum_{a,b:\, P_{6a}(x,y,a,b)} c(y,a,b) + \sum_{a,b:\, P_{6b}(x,y,a,b)} c(x,a,b)$$

$$3e_{48} + 2e_{62} + e_{66} = \sum_{a,b:\, P_{10}(x,y,a,b)} (c(x,y) - 2)$$

$$2e_{47} + 2e_{59} + e_{61} + 2e_{65} = \sum_{a,b:\, P_{9a}(x,y,a,b)} (c(x,b) - 2) + \sum_{a,b:\, P_{9b}(x,y,a,b)} (c(y,a) - 2)$$

$$e_{46} + e_{57} + e_{63} = \sum_{a,b:\, P_{5}(x,y,a,b)} c(x,y)$$

$$e_{45} + e_{52} + 4e_{58} + 4e_{60} = \sum_{a,b:\, P_7(x,y,a,b)} (c(x,b) + c(y,b) - 2)$$

$$e_{44} + 2e_{56} + e_{57} + 2e_{63} = \sum_{a,b:\, P_5(x,y,a,b)} (c(x,a) + c(y,b))$$

$$e_{43} + 2e_{56} + e_{63} = \sum_{a,b:\, P_5(x,y,a,b)} c(a,b)$$

$$2e_{42} + 2e_{56} + e_{57} + 2e_{63} = \sum_{a,b:\, P_5(x,y,a,b)} (c(x,b) + c(y,a) - 4)$$

$$e_{41} + e_{55} + 2e_{58} + 2e_{62} + 2e_{64} + 2e_{66} = \sum_{a,b:\, P_{10}(x,y,a,b)} (c(a) + c(b) - 4)$$

$$e_{40} + 2e_{54} + e_{55} + e_{57} + e_{61} + 2e_{63} + 2e_{64} + 2e_{65} = \sum_{a,b:\, P_{9a}(x,y,a,b)} (c(y) - 2) + \sum_{a,b:\, P_{9b}(x,y,a,b)} (c(x) - 2)$$

$$e_{39} + e_{52} + e_{53} + e_{57} + 2e_{59} + 2e_{63} + 2e_{64} + 2e_{65} = \sum_{a,b:\, P_{9a}(x,y,a,b)} (c(a) - 2) + \sum_{a,b:\, P_{9b}(x,y,a,b)} (c(b) - 2)$$

$$e_{38} + 3e_{49} + e_{56} + e_{59} = \sum_{a,b:\, P_{6a}(x,y,a,b) \vee P_{6b}(x,y,a,b)} (c(a,b) - 1)$$

$$e_{37} + e_{53} + e_{59} = \sum_{a,b:\, P_{6a}(x,y,a,b) \vee P_{6b}(x,y,a,b)} c(x,y)$$

$$2e_{36} + e_{52} + 2e_{60} = \sum_{a,b:\, P_7(x,y,a,b)} c(a,b)$$

$$e_{35} + 6e_{48} + e_{55} + 4e_{62} + e_{64} + 2e_{66} = \sum_{a,b:\, P_{10}(x,y,a,b)} (c(x) + c(y) - 6)$$

$$e_{34} + 2e_{47} + e_{53} + e_{55} + 2e_{59} + e_{61} + 2e_{64} + 2e_{65} = \sum_{a,b:\, P_{9a}(x,y,a,b)} (c(x) - 3) + \sum_{a,b:\, P_{9b}(x,y,a,b)} (c(y) - 3)$$

$$e_{33} + 2e_{47} + e_{52} + 2e_{54} + 2e_{59} + e_{61} + 2e_{63} + 2e_{65} = \sum_{a,b:\, P_{9a}(x,y,a,b)} (c(b) - 3) + \sum_{a,b:\, P_{9b}(x,y,a,b)} (c(a) - 3)$$

$$2e_{32} + 6e_{49} + e_{53} + 2e_{59} = \sum_{a,b:\, P_{6a}(x,y,a,b)} (c(y,a) + c(y,b) - 2) + \sum_{a,b:\, P_{6b}(x,y,a,b)} (c(x,a) + c(x,b) - 2)$$

$$e_{31} + 2e_{42} + e_{44} + 2e_{46} + 2e_{56} + 2e_{57} + 2e_{63} = \sum_{a,b:\, P_5(x,y,a,b)} (c(x) + c(y) - 4)$$

$$e_{30} + 2e_{42} + 2e_{43} + e_{44} + 4e_{56} + e_{57} + 2e_{63} = \sum_{a,b:\, P_5(x,y,a,b)} (c(a) + c(b) - 4)$$

$$2e_{29} + 2e_{38} + e_{45} + e_{52} = \sum_{a,b:\, P_{2a}(x,y,a,b)} (c(y,b) - 1) + \sum_{a,b:\, P_{2b}(x,y,a,b)} (c(x,b) - 1)$$

$$2e_{28} + 2e_{43} + e_{45} + e_{52} = \sum_{a,b:\, P_{2a}(x,y,a,b)} c(x,b) + \sum_{a,b:\, P_{2b}(x,y,a,b)} c(y,b)$$

$$e_{27} + e_{34} + e_{47} = \sum_{a,b:\, P_{4a}(x,y,a,b) \vee P_{4b}(x,y,a,b)} c(x,y)$$

$$2e_{26} + e_{33} + 2e_{36} + e_{50} + e_{52} + 2e_{60} = \sum_{a,b:\, P_7(x,y,a,b)} (c(a) - 3)$$

$$e_{25} + 2e_{32} + e_{37} + 3e_{49} + e_{53} + e_{59} = \sum_{a,b:\, P_{6a}(x,y,a,b)} (c(y) - 3) + \sum_{a,b:\, P_{6b}(x,y,a,b)} (c(x) - 3)$$

$$e_{24} + e_{39} + e_{45} + e_{52} = \sum_{a,b:\, P_{2a}(x,y,a,b) \vee P_{2b}(x,y,a,b)} c(x,y)$$

$$e_{23} + 2e_{36} + e_{45} + e_{52} + 2e_{58} + 2e_{60} = \sum_{a,b:\, P_7(x,y,a,b)} (c(b) - 1)$$

$$e_{22} + e_{37} + e_{44} + e_{53} + e_{56} + e_{59} = \sum_{a,b:\, P_{6a}(x,y,a,b)} (c(x) - 1) + \sum_{a,b:\, P_{6b}(x,y,a,b)} (c(y) - 1)$$

$$2e_{21} + 2e_{38} + 2e_{43} + e_{52} = \sum_{a,b:\, P_{2a}(x,y,a,b) \vee P_{2b}(x,y,a,b)} c(a,b)$$

$$e_{20} + e_{40} + e_{54} = \sum_{a,b:\, P_3(x,y,a,b)} c(x,y)$$

$$e_{19} + e_{33} + 2e_{41} + e_{45} + 2e_{50} + e_{52} + 4e_{58} + 4e_{60} = \sum_{a,b:\, P_7(x,y,a,b)} (c(x) + c(y) - 4)$$

$$e_{18} + 2e_{32} + 2e_{38} + e_{44} + 6e_{49} + e_{53} + 2e_{56} + 2e_{59} = \sum_{a,b:\, P_{6a}(x,y,a,b) \vee P_{6b}(x,y,a,b)} (c(a) + c(b) - 4)$$

$$3e_{17} + 2e_{25} + e_{27} + e_{32} + e_{34} + e_{47} = \sum_{a,b:\, P_{4a}(x,y,a,b)} (c(y) - 3) + \sum_{a,b:\, P_{4b}(x,y,a,b)} (c(x) - 3)$$

$$2e_{16} + 2e_{20} + 2e_{22} + e_{31} + 2e_{40} + e_{44} + 2e_{54} = \sum_{a,b:\, P_3(x,y,a,b)} (c(x) + c(y) - 4)$$

$$e_{15} + 2e_{25} + 2e_{29} + e_{31} + 2e_{32} + e_{34} + 2e_{42} + 2e_{47} = \sum_{a,b:\, P_{4a}(x,y,a,b) \vee P_{4b}(x,y,a,b)} (c(a) + c(b) - 2)$$

$$2e_{14} + e_{18} + 2e_{21} + e_{30} + 2e_{38} + e_{39} + 2e_{43} + e_{52} = \sum_{a,b:\, P_{2a}(x,y,a,b) \vee P_{2b}(x,y,a,b)} (c(a) - 2)$$

$$e_{13} + 2e_{22} + 2e_{28} + e_{31} + e_{40} + 2e_{44} + 2e_{54} = \sum_{a,b:\, P_3(x,y,a,b)} (c(a) + c(b) - 2)$$

$$e_{12} + 2e_{21} + 2e_{28} + 2e_{29} + 2e_{38} + 2e_{43} + e_{45} + e_{52} = \sum_{a,b:\, P_{2a}(x,y,a,b) \vee P_{2b}(x,y,a,b)} (c(b) - 1)$$

BIBLIOGRAPHY

[1] N Przulj, D G Corneil, and I Jurisica. Modeling interactome: scale-free or geometric? *Bioinformatics*, 20 (18):3508–3515, dec 2004. ISSN 1367-4803.

[2] Natasa Przulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2): 177–183, jan 2007. ISSN 1367-4803.

[3] R W Solava, R P Michaels, and T Milenkovic. Graphlet-based edge clustering reveals pathogen-interacting proteins. *Bioinformatics (Oxford, England)*, 28(18):i480–i486, sep 2012. ISSN 1367-4811.

[4] R Milo. Network Motifs: Simple Building Blocks of Complex Networks. *Science*, 298(5594):824–827, oct 2002. ISSN 00368075.

[5] Ron Milo, Shalev Itzkovitz, Nadav Kashtan, Reuven Levitt, Shai Shen-Orr, Inbal Ayzenshtat, Michal Sheffer, and Uri Alon. Superfamilies of evolved and designed networks. *Science (New York, N.Y.)*, 303 (5663):1538–42, mar 2004. ISSN 1095-9203.

[6] N Kashtan, S Itzkovitz, R Milo, and U Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20 (11):1746–1758, jul 2004. ISSN 1367-4803.

[7] N Przulj, D G Corneil, and I Jurisica. Efficient estimation of graphlet frequency distributions in protein-protein interaction networks. *Bioinformatics (Oxford, England)*, 22(8):974–80, apr 2006. ISSN 1367-4803.

[8] Sebastian Wernicke. Efficient detection of network motifs. *IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM*, 3(4):347–59, 2006. ISSN 1545-5963.

[9] Krzysztof Juszczyszyn, Przemysław Kazienko, and Katarzyna Musiał. Local Topology of Social Network. In *KES '08 Proceedings of the 12th international conference on Knowledge-Based Intelligent Information and Engineering Systems, Part II*, pages 97–105, Zagreb, 2008.

[10] Tijana Milenković and Natasa Przulj. Uncovering biological network function via graphlet degree signatures. *Cancer informatics*, 6:257–273, jan 2008. ISSN 1176-9351.

[11] Tijana Milenkovic, Vesna Memisevic, Anand K Ganesan, and Natasa Przulj. Systems-level cancer gene identification from protein interaction network topology applied to melanogenesis-related functional genomics data. *Journal of the Royal Society, Interface / the Royal Society*, 7(44):423–37, mar 2010. ISSN 1742-5662.

[12] Tijana Milenković, Han Zhao, and Fazle E. Faisal. Global Network Alignment In The Context Of Aging. In *Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics - BCB'13*, pages 23–32, New York, New York, USA, 2013. ACM Press. ISBN 9781450324342.

[13] Oleksii Kuchaiev, Tijana Milenkovic, Vesna Memisevic, Wayne Hayes, and Natasa Przulj. Topological network alignment uncovers biological function and phylogeny. *Journal of the Royal Society, Interface / the Royal Society*, 7(50):1341–54, sep 2010. ISSN 1742-5662.

[14] Tijana Milenković, Weng Leong Ng, Wayne Hayes, and Natasa Przulj. Optimal network alignment with graphlet degree vectors. *Cancer informatics*, 9:121–37, jan 2010. ISSN 1176-9351.

[15] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2 (1-2):83–97, mar 1955. ISSN 00281441.

[16] Oleksii Kuchaiev and Natasa Przulj. Integrative network alignment reveals large regions of global network similarity in yeast and human. *Bioinformatics (Oxford, England)*, 27(10):1390–6, may 2011. ISSN 1367-4811.

[17] Sebastian Wernicke and Florian Rasche. FANMOD: a tool for fast network motif detection. *Bioinformatics*, 22(9):1152–1153, may 2006. ISSN 1367-4803.

[18] Christopher Whelan and Kemal Sönmez. Computing graphlet signatures of network nodes and motifs in Cytoscape with GraphletCounter. *Bioinformatics (Oxford, England)*, 28(2):290–1, jan 2012. ISSN 1367-4811.

[19] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498–504, nov 2003. ISSN 1088-9051.

[20] Tijana Milenković, Jason Lai, and Nataša Pržulj. GraphCrunch: A tool for large network analyses. *BMC Bioinformatics*, 9(1), jan 2008. ISSN 1471-2105.

[21] Oleksii Kuchaiev, Aleksandar Stevanović, Wayne Hayes, and Nataša Pržulj. GraphCrunch 2: Software tool for network modeling, alignment and clustering. *BMC bioinformatics*, 12(1):24, jan 2011. ISSN 1471-2105.

[22] D. Marcus and Y. Shavitt. RAGE – A rapid graphlet enumerator for large networks. *Computer Networks*, 56 (2):810–819, feb 2012. ISSN 13891286.

[23] Andrew Chatr-Aryamontri, Bobby-Joe Breitkreutz, Sven Heinicke, Lorrie Boucher, Andrew Winter, Chris Stark, Julie Nixon, Lindsay Ramage, Nadine Kolas, Lara O'Donnell, Teresa Reguly, Ashton Breitkreutz, Adnane Sellam, Daici Chen, Christie Chang, Jennifer Rust, Michael Livstone, Rose Oughtred, Kara Dolinski, and Mike Tyers. The BioGRID interaction database: 2013 update. *Nucleic acids research*, 41 (Database issue):D816–23, jan 2013. ISSN 1362-4962.

[24] Brahim Betkaoui, David B. Thomas, Wayne Luk, and Natasa Przulj. A framework for FPGA acceleration of large graph problems: Graphlet counting case study. In *2011 International Conference on Field-Programmable Technology*, pages 1–8. IEEE, dec 2011. ISBN 978-1-4577-1740-6.

[25] Alon Itai and Michael Rodeh. Finding a Minimum Circuit in a Graph. *SIAM Journal on Computing*, 7(4): 413–423, nov 1978. ISSN 0097-5397.

[26] Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

[27] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, aug 1969. ISSN 0029-599X.

[28] François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation - ISSAC '14*, pages 296–303, New York, New York, USA, 2014. ACM Press. ISBN 9781450325011.

[29] Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Information Processing Letters*, 74(3-4):115–121, may 2000. ISSN 00200190.

[30] Mirosław Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Counting and Detecting Small Subgraphs via Equations. *SIAM Journal on Discrete Mathematics*, 27 (2):892–909, may 2013. ISSN 0895-4801.

[31] P. V. O'Neil. Ulam's Conjecture and Graph Reconstructions. *The American Mathematical Monthly*, 77(1): 35, jan 1970. ISSN 00029890.

[32] Brendan D. McKay. Small graphs are reconstructible. *Australasian Journal of Combinatorics*, 15:123–126, 1997.

[33] Tomaž Hočevar and Janez Demšar. A combinatorial approach to graphlet counting. *Bioinformatics*, 30(4): 559–565, feb 2014. ISSN 1460-2059.

[34] Tomaž Hočevar and Janez Demšar. Computation of Graphlet Orbits for Nodes and Edges in Sparse Graphs. *Journal of Statistical Software*, 71(10), 2016. ISSN 1548-7660.

[35] Tomaž Hočevar and Janez Demšar. Combinatorial algorithm for counting small induced graphs and orbits. *PLOS ONE*, 12(2):e0171428, feb 2017. ISSN 1932-6203.

[36] Stanley Fields. High-throughput two-hybrid analysis. The promise and the peril. *FEBS Journal*, 272(21): 5391–5399, nov 2005. ISSN 1742-464X.

[37] T. Ito, T. Chiba, R. Ozawa, M. Yoshida, M. Hattori, and Y. Sakaki. A comprehensive two-hybrid analysis to explore the yeast protein interactome. *Proceedings of the National Academy of Sciences*, 98(8):4569–4574, apr 2001. ISSN 0027-8424.

[38] Lukasz Salwinski, Christopher S Miller, Adam J Smith, Frank K Pettit, James U Bowie, and David Eisenberg. The Database of Interacting Proteins: 2004 update. *Nucleic acids research*, 32(Database issue):D449–51, jan 2004. ISSN 1362-4962.

[39] Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, sep 1973. ISSN 00010782.

[40] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42, oct 2006. ISSN 03043975.

[41] Ine Melckenbeeck, Pieter Audenaert, Tom Michoel, Didier Colle, and Mario Pickavet. An Algorithm to Automatically Generate the Combinatorial Orbit Counting Equations. *PLOS ONE*, 11(1):1–19, jan 2016. ISSN 1932-6203.

[42] Tijana Milenković, Vesna Memišević, Anthony Bonato, and Nataša Pržulj. Dominating biological networks. *PloS one*, 6(8):e23016, jan 2011. ISSN 1932-6203.

[43] Wayne Hayes, Kai Sun, and Natasa Przulj. Graphlet-based measures are suitable for biological network comparison. *Bioinformatics (Oxford, England)*, 29(4): 483–91, feb 2013. ISSN 1367-4811.

[44] Alina Stoica and Christophe Prieur. Structure of Neighborhoods in a Large Social Network. In *2009 International Conference on Computational Science and Engineering*, pages 26–33. IEEE, 2009. ISBN 978-1-4244-5334-4.

[45] Mirosław Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Counting and detecting small subgraphs via equations and matrix multiplication. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1468–1476, 2011.

[46] Wayne W. Zachary. An Information Flow Model for Conflict and Fission in Small Groups. *Journal of Anthropological Research*, 33(4):452–473, dec 1977. ISSN 0091-7710.

[47] Kilian Thiel and Michael R. Berthold. Node Similarities from Spreading Activation. In *Bisociative Knowledge Discovery*, Lecture Notes in Computer Science, pages 246–262. Springer-Verlag Berlin Heidelberg, 2012.

[48] M Kuramochi and G Karypis. Frequent subgraph discovery. In *Proceedings 2001 IEEE International Conference on Data Mining*, pages 313–320. IEEE Comput. Soc, 2001. ISBN 0-7695-1119-8.

[49] Lars Backstrom and Jure Leskovec. Supervised random walks. In *Proceedings of the fourth ACM international conference on Web search and data mining - WSDM '11*, pages 635–644, New York, New York, USA, 2011. ACM Press. ISBN 9781450304931.

[50] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, may 2007. ISSN 15322882.

[51] Liva Ralaivola, Sanjay J. Swamidass, Hiroto Saigo, and Pierre Baldi. Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093–1110, oct 2005. ISSN 08936080.

[52] Peter Floderus, Mirosław Kowaluk, Andrzej Lingas, and Eva-marta Lundell. Induced Subgraph Isomorphism: Are Some Patterns Substantially Easier Than Others? In *18th Annual International Computing and Combinatorics Conference*, pages 37–48. Springer, Berlin, Heidelberg, 2012.

[53] Virginia Vassilevska and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. In *Proceedings of the 41st annual ACM symposium on Symposium on theory of computing - STOC '09*, pages 455–464, New York, New York, USA, 2009. ACM Press. ISBN 9781605585062.

[54] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3): 209–223, mar 1997. ISSN 0178-4617.

[55] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, jul 1995. ISSN 00045411.

[56] Noga Alon, Phuong Dao, Iman Hajirasouliha, Fereydoun Hormozdiari, and S Cenk Sahinalp. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24(13):241–249, jul 2008. ISSN 1367-4803.

[57] David Eppstein, Maarten Löffler, and Darren Strash. Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time. In *Algorithms and Computation*, volume 6506 of *Lecture Notes in Computer Science*, pages 403–414. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-17516-9.

[58] Paul Erdős and Alfréd Rényi. On random graphs 1. *Publ. Math. Debrecen*, 6:290–297, 1959.

[59] A. Barabasi and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512, oct 1999. ISSN 00368075.

[60] Seifollah Louis Hakimi. On Realizability of a Set of Integers as Degrees of the Vertices of a Linear Graph. I. *Journal of the Society for Industrial and Applied Mathematics*, 10(3):496–506, sep 1962. ISSN 0368-4245.

[61] Joseph Blitzstein and Persi Diaconis. A Sequential Importance Sampling Algorithm for Generating Random Graphs with Prescribed Degrees. *Internet Mathematics*, 6(4):489–522, mar 2011. ISSN 1542-7951.

[62] R Milo, N Kashtan, S Itzkovitz, M E J Newman, and U Alon. On the uniform generation of random graphs with prescribed degree sequences. *arXiv preprint cond-mat/0312028*, pages 1–4, 2003.

[63] Brendan D McKay and Nicholas C Wormald. Uniform generation of random regular graphs of moderate degree. *Journal of Algorithms*, 11(1):52–67, mar 1990. ISSN 01966774.

[64] Mohsen Bayati, Jeong Han Kim, and Amin Saberi. A Sequential Algorithm for Generating Random Graphs. *Algorithmica*, 58(4):860–910, dec 2010. ISSN 0178-4617.

[65] Tom Britton, Maria Deijfen, and Anders Martin-Löf. Generating Simple Random Graphs with Prescribed Degree Distribution. *Journal of Statistical Physics*, 124 (6):1377–1397, oct 2006. ISSN 0022-4715.

[66] William H Press, Brian P Flannery, Saul A Teukolsky, and William T Vetterling. *Numerical recipes*. Cambridge University Press, 1989. ISBN 978-0-521-88068-8.

[67] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, Cambridge, 2014. ISBN 9781139924801.

[68] David Eppstein and Emma S. Spiro. The h-Index of a Graph and its Application to Dynamic Subgraph Statistics. *Journal of Graph Algorithms and Applications*, 16(2):543–567, apr 2012. ISSN 1526-1719.

[69] David Eppstein, Michael T. Goodrich, Darren Strash, and Lowell Trott. Extended dynamic subgraph statistics using h-index parameterized data structures. *Theoretical Computer Science*, 447:44–52, aug 2012. ISSN 03043975.

[70] Min Chih Lin, Francisco J. Soulignac, and Jayme L. Szwarcfiter. Arboricity, h-Index, and Dynamic Algorithms. *Theoretical Computer Science*, 426-427:75–90, may 2010. ISSN 03043975.

[71] Duncan J Watts and Steven H Strogatz. Collective Dynamics of 'Small-World' Networks. *Nature*, 393 (6684):440–442, 1998. ISSN 00280836.

[72] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford InfoLab, 1999.

[73] Max Kotlyar, Chiara Pastrello, Nicholas Sheahan, and Igor Jurisica. Integrated interactions database: tissue-specific view of the human and model organism interactomes. *Nucleic Acids Research*, 44(D1): D536–D541, jan 2016. ISSN 0305-1048.

[74] Waqar Ali, Tiago Rito, Gesine Reinert, Fengzhu Sun, and Charlotte M. Deane. Alignment-free protein interaction network comparison. *Bioinformatics*, 30 (17):i430–i437, sep 2014. ISSN 1460-2059.

[75] Jonathan Mugan, Ranga Chari, Laura Hitt, Eric McDermid, Marsha Sowell, Yuan Qu, and Thayne Coffman. Entity resolution using inferred relationships and behavior. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 555–560. IEEE, oct 2014. ISBN 978-1-4799-5666-1.

[76] Mark Ortmann and Ulrik Brandes. Quad Census Computation: Simple, Efficient, and Orbit-Aware. In Adam Wierzbicki, Ulrik Brandes, Frank Schweitzer, and Dino Pedreschi, editors, *Advances in Network Science*, volume 9564 of *Lecture Notes in Computer Science*, pages 1–13. Springer International Publishing, Cham, 2016. ISBN 978-3-319-28360-9.

[77] Nesreen K. Ahmed, Jennifer Neville, Ryan A. Rossi, and Nick Duffield. Efficient Graphlet Counting for Large Networks. In *2015 IEEE International Conference on Data Mining*, pages 1–10. IEEE, nov 2015. ISBN 978-1-4673-9504-5.

[78] Ethan R. Elenberg, Karthikeyan Shanmugam, Michael Borokhovich, and Alexandros G. Dimakis. Distributed Estimation of Graph 4-Profiles. In *Proceedings of the 25th International Conference on World Wide Web - WWW '16*, pages 483–493, New York, New York, USA, 2016. ACM Press. ISBN 9781450341431.

[79] Anida Sarajlić, Noël Malod-Dognin, Ömer Nebil Yaveroğlu, and Nataša Pržulj. Graphlet-based Characterization of Directed Networks. *Scientific Reports*, 6 (1), 2016. ISSN 2045-2322.

[80] David Aparicio, Pedro Ribeiro, and Fernando Silva. Extending the Applicability of Graphlets to Directed Networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2017. ISSN 1545-5963.