ABSTRACT

O’BRIEN, MICHAEL PATRICK. A Multifaceted Approach to Improving the Practicality of Structural Graph Algorithms. (Under the direction of Blair D. Sullivan.)

Graph algorithms have become an integral part of modern data analytics, but existing approaches have struggled to scale to increasing network sizes. The theoretical computer science community has a rich history of research that circumvents these scalability issues through algorithms that exploit the structural sparsity of graphs. Because many real-world networks from diverse domains are known to share properties like sparsity, clustering, and heavy-tailed degree distributions, structure-based algorithms appear on the surface to be an attractive alternative. However, they come with their own set of problems, such as non-constructive proofs, massive constants hidden in big-O notation, and attention paid to exploiting structures that are unlikely to occur in real data. Consequently, there is a large gap between the most theoretically efficient algorithms and the most practical ones.

This dissertation focuses on alleviating practical barriers to the use of structural graph algorithms in large-scale data analytics, addressing problems on multiple different fronts. First, we show that some structural features can still be identified even in the presence of missing or unknown data. A second contribution is CONCUSS, a first-of-its-kind implementation of a subgraph isomorphism counting algorithm that exploits the bounded expansion structure of graph classes. Through a thorough experimental evaluation of CONCUSS, we establish conditions under which it is competitive with existing algorithms and highlight ways to enhance the general algorithmic framework. As extensions to this framework, we introduce $p$-linear colorings—an alternative graph coloring used to identify bounded expansion structure—as well as practical dynamic programming algorithms to perform a variant of local search in fixed-parameter tractable time in graph classes with bounded expansion. Despite these successes, we prove that recognizing classes of bounded expansion by measuring the density of shallow topological minors is likely not possible in polynomial time. Finally, to demonstrate a concrete way in which structure-based algorithms enable better analysis in scientific domains we build a framework for extracting neighborhoods from metagenomic sequencing data in an interdisciplinary collaboration with computational biologists. Overall, this work illustrates multiple effective strategies for bridging the gap between efficient and practical algorithms.
© Copyright 2018 by Michael Patrick O’Brien

All Rights Reserved
A Multifaceted Approach to Improving the Practicality of Structural Graph Algorithms

by
Michael Patrick O'Brien

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina
2018

APPROVED BY:

__________________________  _________________________
Carla D. Savage             Matthias F. Stallmann

__________________________  _________________________
Steffen Heber               Blair D. Sullivan
Chair of Advisory Committee
DEDICATION

To Dr. Mary Jane Cooper O’Brien, who received her doctoral degree at age 71.
BIOGRAPHY

Michael P. O’Brien grew up in Strongsville, OH. Before coming to North Carolina State University, he received a Bachelor’s degree in Computer Science with a minor in Chinese from the University of Notre Dame.
ACKNOWLEDGEMENTS

Many thanks to Dr. Felix Reidl for laying the groundwork for this dissertation, Dr. C. Titus Brown for time spent devoted to a student outside his lab, Dr. Blair D. Sullivan for advising and encouraging this work, and Shannon Warchol O’Brien for making the whole experience much easier and more enjoyable.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>viii</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>ix</td>
</tr>
</tbody>
</table>

## Chapter 1  Introduction ................................................. 1
  1.1 Motivation ....................................................... 1
  1.2 Overarching Challenges ........................................ 2
  1.3 Research Questions and Outline ............................... 3

## Chapter 2  Background ..................................................... 4
  2.1 Graph Theory .................................................... 4
  2.2 Sparse Graph Hierarchy ......................................... 5
    2.2.1 Treedepth .................................................. 6
    2.2.2 Excluded Minors .......................................... 8
    2.2.3 Bounded Expansion ....................................... 9
    2.2.4 Nowhere/Somewhere Density ................................ 11
    2.2.5 Degeneracy and Core Decompositions ..................... 12
  2.3 Random Graph Models .............................................. 13
    2.3.1 Asymptotic Properties ................................... 13
    2.3.2 RGMs with Bounded Expansion ............................ 13
  2.4 Parameterized Complexity ....................................... 15
    2.4.1 Relationship to Graph Structure ........................ 16
    2.4.2 Parameterized Lower Bounds ............................... 16

## Chapter 3  Locally Estimating Core Numbers ............................. 18
  3.1 Introduction .................................................... 19
  3.2 Related Work ................................................... 21
  3.3 Local Estimation ................................................ 21
    3.3.1 Neighborhood-based Estimation .......................... 21
    3.3.2 Structures Leading to Error .............................. 25
    3.3.3 Expected Behavior on Random Graphs ..................... 26
  3.4 Experimental Results ............................................ 27
    3.4.1 Methods ................................................... 28
    3.4.2 Results ................................................... 28
  3.5 Network Treatment ............................................... 33
    3.5.1 Problem Statement ...................................... 33
    3.5.2 Estimating $k$-core Exposure Probabilities ............ 35
  3.6 Conclusion ..................................................... 37

## Chapter 4  CONCUSS ......................................................... 40
  4.1 Introduction .................................................... 41
  4.2 Algorithmic Landscape for Classes of Bounded Expansion .... 41
  4.3 Subgraph Isomorphism Counting .................................. 42
LIST OF TABLES

Table 3.1  Summary statistics for real-world graphs used in evaluating local core estimators. ................................................................. 28
Table 3.2  Proportion of vertices in $N_{\delta}$. Values less than .01 rounded to 0. ............ 32
Table 4.1  Average run time and dynamic programming operations used for each color splitting heuristic. The statistics for $med$ and $min$ are reported as percent increases over $max$. ................................................................. 57
Table 4.2  Average execution time per operation and average depth of vertices in the treedepth decomposition. Rows for treedepth $t$ only include those operations incurred in counting in color sets of exactly $t$ colors. .............. 59
LIST OF FIGURES

Figure 2.1 An artistic rendering of the sparse graph hierarchy by Felix Reidl. .... 6
Figure 3.1 The core number $k(v)$ is the $y$-value at intersection of two functions. ... 23
Figure 3.2 $T_{2,4}$ is in blue. $T'_{2,4}$ is $T_{2,4}$ plus $w_1$, $w_2$ (red), and the dashed edges. ... 25
Figure 3.3 Core number distribution for the real world networks in Table 3.1. .... 29
Figure 3.4 Proportion of vertices with optimal core number estimate ratios for the propagating (solid green) and induced (dashed blue) estimators as a function of $\delta$. .......... 30
Figure 3.5 Number of vertices with core number estimate ratios less optimal that a given threshold from $\delta = 1$ (lightest line) to $\delta = 4$ (darkest line). $\hat{k}_\delta$ is shown in green while $\tilde{k}_\delta$ is shown in blue. Because the number of vertices with optimal ratios is frequently large (see Figure 3.4), the vertices with optimal ratios may not appear within the limits of the plot in order better capture the distribution of those vertices with suboptimal ratios. .......... 31
Figure 3.6 Average proportion of vertices in $N_\delta$ as a function of $\delta/\Delta$. .......... 32
Figure 3.7 Proportion of vertices with optimal core number estimate ratios for the propagating estimator (solid green) and the induced estimator (dashed blue) as a function of $\delta$. The $x$-axis has been normalized by the diameter. 34
Figure 3.8 $T_{3,3}$ with 13 of 16 vertices treated. $k(v) = k(u_1) = k(u_2) = k(u_3) = 3$. Although $v$, $u_1$, $u_2$, and $u_3$ have all of their neighbors treated, they only have core number 1 with respect to the treated subgraph. .......... 36
Figure 3.9 $P[X_k(d)] - P[\hat{X}_k]$ for the WPG graph at $p = 0.25$. Vertices with $P[X_k(d)(v)] = 0$ are omitted. .................. 38
Figure 3.10 Histogram of differences between $P[X_k(d)]$ before and after pruning for $\kappa = 7$ and $p = 0.25$. The $x$-axis gives the difference in probability, while the $y$-axis gives the proportion of vertices occurring in that bin. For clarity, only those vertices which are not pruned are considered in the plot. ................. 39
Figure 4.1 $P_4$ (far left) appears once as an induced subgraph of $G$ (center left) but there are two isomorphisms from $P_4$ to $G$ (center right and far right). 43
Figure 4.2 Algorithmic pipeline in CONCUSS. .................. 46
Figure 4.3 The highlighted $P_4$ only uses three colors and thus will be counted in multiple color sets of size four. .................. 49
Figure 4.4 Average difference/sum ratio in coloring size (top), COLOR time (middle), and DECOMPOSE, COMPUTE, and COMBINE time (bottom) between paired configuration options. .................. 51
Figure 4.5 Distribution of time spent in the different submodules of the COLOR module. .................. 52
Figure 4.6 Relationship between number of colors and total execution time using Inclusion-exclusion. .................. 52
Figure 4.7 $T_{\ell,s,r}$. .................. 54
Figure 4.8 Average difference/sum ratio between CONCUSS and NXVF2 on $T_{d,s,1}$ as a function of $s$, the average number of $P_1$s per tree vertex. Each small plot shows a fixed motif and value of $d$. Negative ratios indicate CONCUSS is faster. ................................................. 55

Figure 4.9 Average difference/sum ratio between CONCUSS and NXVF2 on $T_{d,s,4}$ as a function of $s$, the average number of $P_4$s per tree vertex. Each small plot shows a fixed motif and value of $d$. Negative ratios indicate CONCUSS is faster. ................................................. 56

Figure 4.10 Creating a new color class (black) using each of the three splitting heuristics ($\min$, $\med$, $\max$). ................................................................. 57

Figure 4.11 Observed vs. predicted average execution time per operation in the stochastic block graphs. We only include those operations incurred in counting in color sets of exactly $t$ colors. ................................................. 58

Figure 5.1 Linear colorings of graphs in Lemmas 8-9. ................................................................. 65

Figure 5.2 The graph $G$ and coloring $\psi$ for $\Phi = (x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_2)$. ................................................................. 72

Figure 6.1 The graph $H$ is a 1-shallow topological minor of $G$, as witnessed by the model marked with blue nails and golden paths. ................................................. 77

Figure 6.2 Three variable gadgets $D_i$, $D_j$, $D_k$ connected by the gray clause vertex $u_{ijk}$. Setting variable $x_k$ to true and $x_i$, $x_j$ to false corresponds to the contractions on the right-hand side. The apex vertex $a$ is not shown. ................................................. 81

Figure 6.3 A sketch of the construction for Theorem 11, with an exemplary connection of the variable-path $X_1$ to the first clause gadget (here, $x_1$ appears negatively in $C_1$). Dashed edges denote parts that are actually connected via decision gadgets. 3-paths between the grid $R$ and the clause gadgets $(A_i, B_i)$ are not drawn. ................................................. 87

Figure 8.1 Overhead purity at varying ranks of each $N_i$ as a function of the overhead of $N_i$. ................................................................. 111

Figure 8.2 Histogram of taxonomic purity values of the shadows of the binned CAtlas nodes. ................................................................. 114

Figure 8.3 Overhead purity at varying ranks of each $F_i$ as a function of the overhead of $F_i$. The colors of the points correspond to $\alpha, \beta$ pairs. ................................................. 116
1.1 Motivation

Computing mathematical properties of large network data sets is a key component to understanding fundamental relations between entities in fields such as neuroscience, national security, and finance. One particular method of gaining insight is to focus on the underlying structure of the graph. Many networks have been observed to exhibit structural sparsity, which means that they not only contain just a small fraction of the possible edges, but also that no large portion of the network is dense. This is further evidenced by the existence of properties common across unrelated domains, such as clustering, short paths, and heavy-tailed degree distributions. Measuring the structural features can identify important entities in the network or suggest properties of the process from which the data originated.

In addition to being interesting in its own right, structural sparsity can play a central role in scalable algorithm design. Many graph properties with a natural interpretation in the data domain are easy to formalize but difficult to compute and existing algorithms have struggled to scale. However, the absence of general polynomial-time algorithms does not necessarily imply that computing the property is equally difficult on all instances. For example, finding a maximum clique is NP-complete in general but trivial to solve on trees, where the only complete subgraph is a single edge. It is possible to design algorithms where the delineation in performance between “easy” and “hard” input graphs corresponds to the existence or absence
of a particular structural property. If networks from a domain have known structure, then an appropriate algorithm can “exploit” this structure to be more efficient than it would on an arbitrary graph.

While the structural graph theory and parameterized algorithms research communities have developed an extensive toolkit for identifying and utilizing structural sparsity, few of those tools have seen widespread use in the network science community. For theoretical computer scientists, the primary objective in algorithm design has historically been efficiency as measured by the worst-case asymptotic computational complexity. As a result, algorithms exploiting structural sparsity often have massive constants hidden in big-O notation and/or non-trivial implementation details left unaddressed. In the same vein, some results are based on meta-theorems that non-constructively prove the existence of efficient algorithms without a clear way to utilize them in practice. Furthermore, a significant portion of the research has studied structural properties that are conceptually interesting but are unlikely to occur in networks from common data sources. Simply put, the existing literature is ripe with efficiency but not necessarily with practicality. These shortcomings have all led network scientists to turn to heuristic and sampling approaches that vary significantly across domains and whose reasons for success are not well understood.

1.2 Overarching Challenges

This work is focused on bridging the gap between theoretical computer science and network science by addressing practical issues relating to structural sparsity in large-scale data analytics. Rather than honing in on a single aspect of practicality, we take a more holistic approach and consider the potential for progress on several distinct fronts. To this end, we have identified five overarching challenges that exist in the current state of the literature.

**Challenge 1** How can we identify structural features when data is missing or incomplete?

**Challenge 2** To what extent are existing algorithms for structurally sparse graphs practical?

**Challenge 3** How can new theoretical results address pitfalls in existing algorithms?

**Challenge 4** What problems can be solved efficiently on structurally sparse graphs without the use of “heavy machinery”?

**Challenge 5** How can structural graph algorithms be utilized effectively in interdisciplinary collaborations?

Though these challenges are broad and inherently impossible to fully answer, we will use them to guide the specific research questions presented in the next section.
1.3 Research Questions and Outline

We focus on the following six research questions selected to address concrete pieces of each of the five overarching challenges.

**Research Question 1** How can the network core structure be identified using limited information about the graph?

**Research Question 2** Are existing algorithms for counting subgraph isomorphisms in bounded expansion graph classes practical and implementable?

**Research Question 3a** Are there alternative coloring schemes for graph classes of bounded expansion that are algorithmically advantageous?

**Research Question 3b** Can bounded expansion classes be recognized by efficiently computing the density of shallow topological minors?

**Research Question 4** Can the structure of bounded expansion graph classes be exploited to create efficient algorithms for variants of local search?

**Research Question 5** How does bounded expansion structure enable efficient organization and querying of metagenomic sequencing data?

The remaining chapters of this dissertation will focus on answering these questions. Chapter 2 contains the necessary background on structural sparsity, computational complexity, and random graph models. In Chapter 3, we answer Research Question 1 by describing an algorithm for locally estimating the core number of a vertex and the algorithm’s impact. The algorithmic workflow of CONCUSS, a software tool that counts subgraph isomorphisms, is outlined in Chapter 4, along with an experimental evaluation of CONCUSS in order to answer Research Question 2. Building upon the weaknesses identified in these experiments, Chapter 5 is devoted to proving relationships between the structure of a graph and a novel graph coloring that fits the requirements of Research Question 3a. Chapter 6 explores algorithms for recognizing structurally sparse graphs through the density of their shallow topological minors; we give a negative answer to Research Question 3b via hardness reductions. In terms of positive algorithmic results, we answer Research Question 4 in the affirmative in Chapter 7. Our attention is focused on $k$-exchange, a variant of local search; we detail algorithms for $k$-exchanges for three problems in classes of bounded expansion. To address Research Question 5, we introduce in Chapter 8 a novel data structure used to query metagenomic sequencing data and empirically verify its usefulness on real-world data. Finally, the contributions of this work and future research directions are summarized in Chapter 9.
Summary

Improving practicality in structural graph algorithms necessitates understanding the theory of the sparse graph hierarchy—a taxonomy of relationships among various classes of sparse graphs. This work will primarily concern itself with three classes on the hierarchy: bounded treedepth, bounded expansion, and bounded degeneracy. Membership in a sparse class can confer algorithmic advantages in the form of parameterized algorithms, which provide efficient ways to solve otherwise intractable problems by exploiting graph structure. Prior work has shown that multiple common random graph models will asymptotically almost surely generate graphs that belong to classes of bounded expansion, motivating our choice to center a significant portion of this work around algorithms for that class.

2.1 Graph Theory

A graph $G$ is a set of vertices $V(G)$ and edges $E(G)$ that join pairs of vertices. We denote the edge between $u$ and $v$ as $uv$ and say that $u$ and $v$ are adjacent if such an edge exists. Unless otherwise noted, all graphs are undirected (the edge $uv$ is the same as $vu$) and simple (there is at most one edge between each pair of vertices and no vertex has an edge to itself). For convenience, we define $|G| = |V(G)|$ and $\|G\| = |E(G)|$. In places where it is unambiguous,
we let \( n = |G| \).

The subgraph induced on the vertex set \( X \) is the graph \( H \) such that \( V(H) = X \) and \( E(H) = \{ uv \in E(G) : \{ u, v \} \subseteq X \} \). We denote this as \( G[X] \) and say that \( H \) is an induced subgraph of \( G \). Vertex \( a \) is an apex with respect to a subgraph \( H \) if \( a \) is adjacent to every vertex in \( H \). The neighbors of vertex \( v \), denoted \( N(v) \), are the set of vertices \( u \) for which \( uv \in E(G) \) and the degree of \( v \), \( d(v) \), is equal to the size of \( N(v) \). The neighbors of \( v \) are also collectively referred to as the open neighborhood of \( v \) (in contrast with the closed neighborhood, defined as \( N(v) \cup \{ v \} \) and denoted \( N[v] \)).

We say \( P \) is a \( v_1v_\ell \)-path if \( V(P) = \{ v_1, \ldots, v_\ell \} \) and \( E(P) = \{ v_iv_{i+1} : 1 \leq i \leq \ell - 1 \} \); we will notate this as \( P = v_1, \ldots, v_\ell \). Given paths \( P = v_1, \ldots, v_\ell \) and \( Q = u_1, \ldots, u_\ell \) such that \( V(P) \cap V(Q) = \emptyset \) and \( v_\ell, u_1 \) are adjacent, the path \( P \cdot Q = v_1, \ldots, v_\ell, u_1, \ldots, u_\ell \) is the concatenation of \( P \) and \( Q \).

In a rooted tree \( T \), we let \( T_v \) be the subtree of \( T \) rooted at \( v \) and the leaf paths of \( T_v \) be the set of paths from a leaf of \( T_v \) to \( v \). For any pair of vertices \( u, v \in V(T) \) such that \( u \neq v \) such that \( u \in T_v \), \( u \) is said to be an descendant of \( v \) and \( v \) is said to be an ancestor of \( u \). The set of all ancestors of a vertex \( v \) is denoted \( A(v) \). If \( u \) is neither an ancestor nor a descendant of \( v \) in \( T \), \( u \) and \( v \) are unrelated. The depth of a vertex \( v \) in a rooted tree \( T \) is defined as \( 1 + |A(v)| \), and the depth of \( T \) is defined to be \( \max_{v \in V(T)} \text{depth}(v) \).

A coloring \( \phi \) of a graph \( G \) is a mapping of the vertices of \( G \) to colors \( 1, \ldots, k \) and has size \( |\phi| = k \). Coloring \( \phi \) is proper if no two adjacent vertices have the same color.

For edge \( uv \), the edge contraction operation replaces \( u \) and \( v \) with a new vertex \( w \) such that \( N(w) = N(u) \cup N(v) \). The edge subdivision operation replaces edge \( uv \) with a new vertex \( w \) and edges \( uw \) and \( wv \); if \( H \) is a graph obtained by subdividing edges of graph \( G \), we say that \( H \) is a subdivision of \( G \). The “inverse” of edge subdivision is smoothing, which replaces a vertex \( w \) with an edge \( uv \) such that \( \{ u, v \} \subseteq N(w) \).

### 2.2 Sparse Graph Hierarchy

The sparse graph hierarchy is a classification system of structurally sparse graphs formally organized\(^1\) by Nešetřil and Ossana de Mendez [74]. It organizes infinite classes of graphs into nested groups based on common structural properties. For example, the class of paths are a class of planar graphs as well as a class of trees. The relationships among some properties of sparse graphs are shown in Figure 2.1. We will describe classes as low or high on the hierarchy in a manner that coincides with the orientation of Figure 2.1: lower classes are more structurally restrictive. In the remainder of this section, we describe characteristics of the

\(^1\)Many notions of sparsity were already defined in the literature prior to the work by Nešetřil and Ossana de Mendez.
sparse graph classes on which this work is centered: bounded treedepth, excluded (topological) minors, bounded expansion, nowhere dense, and bounded degeneracy.

### 2.2.1 Treedepth

Classes of graphs with bounded treedepth [70] are relatively low on the sparse graph hierarchy, and characteristically have structure similar to that of a collection of shallow trees.

**Definition 1.** A treedepth decomposition of a graph $G$ is an injective mapping $\psi : V(G) \rightarrow V(F)$, where $F$ is a rooted forest and $uv \in E(G) \implies \psi(u)$ is an ancestor or descendant of $\psi(v)$. The depth of a treedepth decomposition is the depth of $F$. The treedepth of $G$, denoted $td(G)$ is the minimum depth of a treedepth decomposition of $G$.

In other words, a treedepth decomposition arranges the vertices of $G$ in such a way that no edge joins vertices from different branches of the tree. We can also equivalently define the treedepth using the closure of a tree.

**Definition 2.** Let $F$ be a rooted forest. The closure of $F$, denoted $clos(F)$ is the graph whose vertex set is $V(F)$ and vertices $u, v \in V(F)$ are adjacent if and only if $u$ is an ancestor of $v$ or $v$ is an ancestor of $u$ in $F$.

If $\psi$ is a treedepth decomposition mapping $G$ into $F$, $clos(F)$ contains all the possible edges consistent with that decomposition, giving rise to this straightforward proposition.
**Proposition 1.** The treedepth of $G$ is equal to the minimum depth of a rooted forest $F$ such that $G \subseteq \text{clos}(F)$.

Treedepth is also closely related to a special type of graph coloring. For any subgraph $H$, coloring $\phi$, and color $c$, if there is exactly one vertex $v \in H$ such that $\phi(v) = c$ we say $c$ appears uniquely in $H$ and $v$ is the center of $H$ with respect to $\phi$. A subgraph with no unique color is said to be non-centered.

**Definition 3.** A centered coloring of graph $G$ is a vertex coloring such that every connected subgraph $H$ has a center. We refer to the minimum number of colors needed for a centered coloring of $G$ as its centered coloring number, denoted $\chi_{\text{cen}}(G)$.

---

**Algorithm 1** centcolor($G, T$)

**Input:** Graph $G$, treedepth decomposition $T$ of $G$

**Output:** Canonical centered coloring $\phi$ of $G$ with respect to $T$

1. $k \leftarrow \text{depth}(T)$
2. **for all** $v \in V$ **do**
   3. $\phi(v) \leftarrow \text{depth}(v)$
4. **end for**
5. **return** $\phi$

---

**Algorithm 2** tddecomp($G, \phi$)

**Input:** Graph $G$, centered coloring $\phi$ of $G$

**Output:** Canonical treedepth decomposition $T$ of $G$ with respect to $\phi$

1. **if** $|G| = 0$ **then**
   2. **return** $\emptyset$
3. **end if**
4. $v \leftarrow \text{center}(\phi|_G)$
5. $\text{root}(T) \leftarrow v$
6. **for all** components $C \in G \setminus \{v\}$ **do**
7. $T' \leftarrow \text{tddecomp}(C, \phi|_C)$
8. $\text{parent}(\text{root}(T')) \leftarrow v$
9. **end for**
10. **return** $T$

---

Note that a centered coloring is also proper, or else there would be a connected subgraph of size two with no center. Given a treedepth decomposition of depth $k$, we can generate a
centered coloring using \( k \) colors by using Algorithm 1, which bijectively assigns the colors to levels of the tree and colors vertices according to their level. Likewise, given a centered coloring with \( k \) colors, we can generate a treedepth decomposition of depth at most \( k \) by choosing a center \( v \) to be the root and recursing on the components of \( G \setminus \{v\} \), as detailed in Algorithm 2. We refer to colorings and treedepth decompositions resulting from Algorithms 1 and 2 as canonical. Together, these algorithms imply the treedepth and the centered coloring number of a graph are equal.

**Proposition 2.** \([70]\) \( \text{td}(G) = \chi_{\text{cen}}(G) \).

### 2.2.2 Excluded Minors

**Graph minors** are at the heart of structural graph theory because much of the sparse graph hierarchy can be formally characterized by the presence or absence of particular graph minors.

**Definition 4.** Graph \( H \) is said be a minor of graph \( G \) if a series of vertex deletions, edge deletions, and edge contractions can transform \( G \) into \( H \).

For example, the famous theorem of Wagner \([101]\) states that the planar graphs are exactly those graphs that exclude the complete bipartite graph \( K_{3,3} \) and the complete graph \( K_5 \) as minors. Classes that exclude a minor, otherwise known as \( H \)-minor-free classes, generalize planar graphs and many of the other lower classes on the hierarchy; every class \( C \) in an \( H \)-minor-free class has an associated (non-empty) finite set of graphs \( C_{\text{ex}} \) such that no graph \( G \in C \) has a minor in \( C_{\text{ex}} \). By the celebrated Robertson and Seymour graph minor theorem \([85, 86]\), if a class of graphs \( C \) is minor-closed—meaning that every minor of a graph \( G \in C \) is also a member of \( C \)—then \( C \) is an \( H \)-minor-free class.

This theorem can be used to prove that bounded treedepth classes are also \( H \)-minor-free. Contracting an edge corresponds to merging a vertex with an ancestor in a treedepth decomposition, which maintains the validity of the treedepth decomposition and thus does not increase the treedepth. Since a centered coloring of \( G \) is also a centered coloring of its subgraphs, edge and vertex deletions cannot increase the treedepth either. Thus for any minor \( H \) of \( G \), \( \text{td}(H) \leq \text{td}(G) \) proving that bounded treedepth classes are minor-closed.

We can also characterize graph classes by excluding only certain types of minors. Particular focus has been paid to **topological minors**, which preserve topological embeddings.

**Definition 5.** Graph \( H \) is said to be a topological minor of graph \( G \) if a series of vertex deletions, edge deletions, and smoothings can transform \( G \) into \( H \).

Creating a topological minor \( H \) from graph \( G \) can be thought of as selecting a set of vertices of \( G \) to be the vertices of \( H \), which we call the nails, and a set of vertex-disjoint paths in \( G \) between each pair of nails that are adjacent in \( H \).
2.2.3 Bounded Expansion

Some classes that are not minor-closed are still structurally sparse in meaningful ways. Another class known as bounded expansion [71–74] can be characterized by looking at the way in which certain minors are constructed.

**Definition 6.** Graph $H$ is said to be an $r$-shallow minor of graph $G$ if there exist subsets $V_1, \ldots, V_\ell$ of $V(G)$ such that

1. The sets $V_1, \ldots, V_\ell$ are pairwise disjoint.
2. For each $V_i$ there exists a vertex $x \in V_i$ such that for all $y \in V_i$, the distance in $G[V_i]$ from $x$ to $y$ is at most $r$.
3. There is a bijection $\psi : V(H) \to \{V_1 \ldots V_\ell\}$ such that if $u, v \in H$ are adjacent, then some vertex in $\psi(u)$ is adjacent to a vertex in $\psi(v)$.

The set of all $r$-shallow minors of $G$ is denoted $G\nabla r$.

In other words, the $r$-shallow minors are formed by contracting small diameter subgraphs into the vertices of the minor while maintaining the connectivity between the contracted subgraphs. Classes of bounded expansion limit the edge density of the $r$-shallow minors, allowing dense minors to be formed only when sufficiently large radius subgraphs are contracted.

**Definition 7.** Define the greatest reduced average density (grad) to be

$$\nabla_r(G) = \max_{H \subseteq G \nabla r} \frac{\|H\|}{|H|}.$$

A class of graphs $\mathcal{C}$ is said to have bounded expansion if there exists some function $f(r)$ such that for each $G \in \mathcal{C}$, $\nabla_r(G) \leq f(r)$ for all $r$.

Although excluding graph $H$ as a topological minor is not in general equivalent to excluding $H$ as a minor, an analogous definition of shallow topological minors can also be used to characterize bounded expansion classes.

**Definition 8.** Graph $H$ is said to be an $r$-shallow topological minor of graph $G$ if there exists $U \subseteq V(G)$ and paths $P_1, \ldots, P_\ell$ in $G$, where $P_i = v_{i,1}, \ldots, v_{i,q_i}$ such that

1. For each $1 \leq i \leq \ell$, $P_i \cap U = \{v_{i,1}, v_{i,q_i}\}$.
2. For each $1 \leq i \neq j \leq \ell$, $(P_i \setminus \{v_{i,1}, v_{i,q_i}\}) \cap (P_j \setminus \{v_{j,1}, v_{j,q_j}\}) = \emptyset$.  

9
3. Each $q_i$ satisfies $q_i \leq 2r + 1$.

4. There exist bijections $\psi_V : V(H) \rightarrow U$ and $\psi_E : E(H) \rightarrow \{P_1, \ldots, P_\ell\}$ such that $u, v \in H$ are adjacent if and only if $\psi_E(uv) = P_i, v = v_{i,1}$ and $u = v_{i,0}$ for some $i$.

The set of all $r$-shallow topological minors of $G$ is denoted $G\nabla r$.

**Definition 9.** The topological greatest reduced average density (top-grd) to be

$$\nabla_r(G) = \max_{H \in G\nabla r} \frac{\|H\|}{|H|}.$$

**Proposition 3.** [71] A class of graphs $C$ has bounded expansion if and only if there exists some function $f(r)$ such that for each $G \in C$, $\nabla_r(G) \leq f(r)$ for all $r$.

Algorithms for classes of bounded expansion tend not to directly use the exclusion of shallow (topological) minors, but instead use a particular type of vertex coloring to identify structure. Since not all classes with bounded expansion also have bounded treedepth, we cannot assume they have centered colorings with a bounded number of colors. There is, however, a relationship between bounded expansion and a localized variant of centered colorings called $p$-centered colorings.

**Definition 10.** A $p$-centered coloring of graph $G$ is a vertex coloring such that every connected subgraph $H$ has a unique color or uses at least $p$ colors. The minimum number of colors needed for a $p$-centered coloring of $G$ is denoted $\chi_p(G)$.

By this definition and Algorithm 2, any set of vertices using $i < p$ colors induces a graph whose treedepth is at most $i$. These $p$-centered colorings give an alternative characterization of classes with bounded expansion.

**Proposition 4.** [71] A class of graphs $C$ has bounded expansion if and only if for some function $f(p)$, every graph $G \in C$ satisfies $\chi_p(G) \leq f(p)$ for all $p$.

Of central importance to computing $p$-centered colorings efficiently are transitive fraternal augmentations (TFAs) [74]. The idea of TFAs is to augment the graph with directed arcs in order to encode local properties; to encode information about subgraphs of radius $r$, we must perform $r$ iterations of the algorithm. Since TFAs work with directed graphs, we first transform $G$ into a directed acyclic graph (DAG) $\vec{G}_1$ by orienting the edges in a way that minimizes indegree (such as using Algorithm 3 in Section 2.2.5). We then create a sequence of DAGs $\vec{G}_1 \subseteq \vec{G}_2 \subseteq \cdots \subseteq \vec{G}_r$ in which $\vec{G}_{i+1}$ is formed by adding arcs between all pairs of transitive and fraternal vertices in $\vec{G}_i$. Vertices $u, v$ are said to be transitive if there is a vertex $x$ such that $\{ux, xv\} \subseteq \vec{G}_i$ and fraternal if $\{ux, vx\} \subseteq \vec{G}_i$. If $u, v$ are transitive, then we must add the arc $uv$;
if they are fraternal we add either $uv$ or $vu$ depending on whether which choice preserves acyclicity and minimizes indegree. To obtain $p$-centered colorings with a bounded number of colors, we use the following two results.

**Proposition 5.** [71] If $G$ belongs to a class of bounded expansion $C$, then for any $i$ the maximum indegree of the $i$th TFA depends only on $C$.

**Proposition 6.** [71] Let $\sigma$ be a topological ordering of the $p$th TFA, $\vec{G}_p$. Then a greedy coloring of $\vec{G}_p$ over $\sigma$ is a $(p + 1)$-centered coloring of $G$.

Proposition 5 also implies that the number of augmenting arcs in each round does not grow arbitrarily large in classes of bounded expansion, which means that the algorithm runs in time linear in the size of the graph.

Reidl introduced an alternative to TFAs called *distance-truncated transitive fraternal augmentations* (DTFAs) in which only the “most important” arcs are added in each round [83]. This algorithm assigns an integer label $w$ to each arc in $\vec{G}_i$. All arcs $uv \in \vec{G}_1$ receive label $w(uv) = 1$; arc $uv$ is added to $G_i$ with $w(uv) = i$ if transitive pair $u, v$ satisfies $w(ux) + w(xv) \leq i$ and likewise if $u, v$ are fraternal. In this way, DTFAs shortcut paths of length at most $i$. We note that every arc in the $i$th DTFA must also be in the $i$th TFA of $G$, which means that Propositions 5 and 4 have analogous versions for DTFAs.

**Proposition 7.** [83] If $G$ belongs to a class of bounded expansion $C$, then for any $i$ the maximum indegree of the $i$th DTFA depends only on $C$.

**Proposition 8.** [83] There exists a function $f$ such that a greedy coloring over a topological ordering of the $f(p)$th DTFA $\vec{G}_{f(p)}$ is a $p$-centered coloring of $G$.

### 2.2.4 Nowhere/Somewhere Density

Nešetřil and Ossona de Mendez used $r$-shallow minors to dichotomize between structurally sparse classes and structurally dense classes.

**Definition 11.** A class of graphs $C$ is said to be nowhere dense if there is a function $f(r)$ such that every graph $G \in C$ does not have an $r$-shallow clique minor of size $f(r)$.

It is clear that bounded expansion classes are also nowhere dense, since they cannot have arbitrarily dense $r$-shallow minors, let alone a clique. According to the sparse graph hierarchy, all classes that are considered to be structurally sparse must be nowhere dense. The classes that are not nowhere dense are considered to be somewhere dense. Somewhere dense classes are thus marked by having $r$-shallow clique minors of unbounded size (with respect to $r$).
2.2.5 Degeneracy and Core Decompositions

The structure of graphs can also be viewed through a lens other than that of graph minors. The degeneracy of a graph can identify the presence of a dense “core” in a graph.

**Definition 12.** A graph $G$ is $d$-degenerate if there exists an ordering of its vertices $\sigma$ such that every vertex has at most $d$ neighbors that precede it in $\sigma$. The smallest $d$ for which $G$ is $d$-degenerate is the degeneracy of $G$ and an ordering $\sigma$ that witnesses this property is a degeneracy ordering.

Unlike many other structural properties such as treedepth or grad, computing the degeneracy can be done in linear time using Algorithm 3. Reversing the order that the vertices were removed in Algorithm 3 gives a degeneracy ordering.

**Algorithm 3 degeneracy($G$)**

**Input:** Graph $G$

**Output:** Degeneracy of $G$

1: $i \leftarrow 0$
2: while $|G| > 0$ do
3: \hspace{1em} while there is a vertex $v$ of degree $\leq i$ do
4: \hspace{2em} $G \leftarrow G \setminus \{v\}$
5: \hspace{1em} end while
6: \hspace{1em} $i \leftarrow i + 1$
7: end while
8: return $i - 1$

Algorithm 3 also identifies structural information on the vertex level. The core number of a vertex $v$ is the value of $i$ when $v$ is removed from the graph, while the $k$-core of $G$ is the subgraph induced on all vertices of core number at least $k$. Thus, the degeneracy can equivalently be viewed as the largest value of $k$ for which $G$ has a non-empty $k$-core. The organization of a graph into its $k$-cores is known as a core decomposition.

Classes of bounded degeneracy do not fit nicely into the sparse graph hierarchy. While classes of bounded expansion (and those lower on the hierarchy) have bounded degeneracy, so do some families that are somewhere dense. For example, the family of 1-subdivisions of cliques are all 2-degenerate. On the other hand, a nowhere dense graph can have degeneracy $O(\log n)$. 
2.3 Random Graph Models

The taxonomy of the sparse graph hierarchy provides a unique lens from which to view graphs formed by randomized processes. A random graph model (RGM) defines a randomized process by which a graph can be generated. One of the most basic models is the Erdős-Rényi random graph [33] in which the edge between any pair of the \( n \) vertices appears independently with probability \( p \). More sophisticated processes have been devised in order to generate random data instances that match properties of real-world networks, such as their degree distributions [19] or the existence of communities [15].

2.3.1 Asymptotic Properties

While network scientists have often used random graph models to generate synthetic data on which to test algorithmic techniques, they are important to structural graph theory in a different way. Since the classification scheme on the sparse graph hierarchy relies on categorizing infinite families of graphs, there is not a way to place single networks into the hierarchy. On the other hand, random graph models can be used to define infinite families of graphs. Although many random graph models have a non-zero probability of generating any finite graph, as the size of the generated graphs grows large, certain properties may be very likely (or unlikely) to occur. Asymptotic behavior can be used to formally define “very likely”.

**Definition 13.** A random event \( X \) is said to happen asymptotically almost surely (a.a.s.) if

\[
\lim_{n \to \infty} \mathbb{P}[X] = 1.
\]

This allows us to say that a random graph model belongs to a particular structural class as long as it produces graphs that a.a.s. have some structural property. It has been proven, for example, that the Erdős-Rényi model a.a.s. has bounded degeneracy [80] but linear (and thus unbounded) treedepth [38] when the average degree is held constant, e.g., \( p \in O(1/n) \).

2.3.2 RGMs with Bounded Expansion

Though there has been significant research analyzing the asymptotic behaviors of RMGs, the results of most relevance to this paper pertain to classes of bounded expansion. We will use the number of graph models that a.a.s. have bounded expansion as justification for our focus on that class in Research Questions 2-5. Below are several examples of such models.
Perturbed Bounded Degree

Demaine et al. [27] proved that a significant generalization of the Erdős-Rényi model has bounded expansion a.a.s. This model, which they call perturbed bounded degree begins with a bounded degree graph and then allows arbitrary probabilities to be assigned to each other potential edge. As long as all such probabilities are bounded by $O(1/n)$, the model has bounded expansion.

Stochastic Block

The stochastic block model is similar to the Erdős-Rényi model but also tries to simulate the existences of communities. It takes as input a partition of the vertices into $r$ subsets $C_1, \ldots, C_r$ and an $r \times r$ probability matrix $P$. For vertices $u \in C_i$ and $v \in C_j$, the probability that edge $uv$ appears is $P[i, j]$. This allows the user to set higher probabilities to edges with both endpoints in a cluster $C_i$ than to edges that bridge two clusters. The proof for the perturbed bounded degree immediately implies bounded expansion in the stochastic block model as long as all such probabilities are bounded by $O(1/n)$.

Chung-Lu

It is also proven in [27] that the Chung-Lu model [19] can generate graphs that a.a.s. belong to a class of bounded expansion. The Chung-Lu model produces random graphs that in expectation match a specified degree sequence $D$. The structural sparsity of graphs generated by the Chung-Lu model is dependent on the proportion of vertices expected to have high degree. When this proportion decays supercubically, e.g., $O(1/d^{3+\epsilon})$, vertices have degree $d$, the model has bounded expansion a.a.s. If the decay is subcubic e.g. $\Omega(1/d^{3-\epsilon})$, the graphs generated are in a somewhere dense class a.a.s., while a cubic decay implies the graphs will a.a.s. belong to a nowhere dense class with unbounded expansion.

The Chung-Lu model can easily match heavy-tailed degree distributions commonly found in real-world networks but struggles to match other properties like clustering. To remedy this, a variant of the model adds households, meaning that it generates the prescribed degree distribution and then expands each vertex into a small subgraph with high clustering, such as a clique. Provided that the households have constant size with respect to $n$, the above results are still applicable to the household variant.

The authors of [27] also performed an experiment to give insight into whether the structural constraints of bounded expansion are realistic for real data. They generated multiple random instances of the Chung-Lu model matching the degree distributions of some real-world networks. Then a $p$-centered coloring was computed for each graph (originals and random
“copies”). For most of the networks, the number of colors used in the $p$-centered coloring in the real data was within or below the range of number of colors for the corresponding synthetic data sets. Since bounded $p$-centered coloring numbers imply bounded expansion, the authors concluded that these networks indeed had structure consistent with bounded expansion.

Random Intersection

In other research [34] the random intersection model has been shown to produce graphs that a.a.s. belong to classes of bounded expansion under certain conditions. The random intersection model begins with two sets of vertices $V$ and $A$ and creates a bipartite graph by adding edges between $V$ and $A$ uniformly at random with probability $p$. For each pair of vertices $u, v \in V$, the edge $uv$ is added if $u$ and $v$ have a common neighbor in $A$. At the conclusion of this process, $A$ is discarded and $V$ and its edges are returned. This process simulates a network of entities whose connectivities are based on mutual attributes e.g. students who have taken a common class. When the probability of each random edge is bounded by $O(1/|V|^{1+\epsilon})$, the random intersection model a.a.s. has bounded expansion.

2.4 Parameterized Complexity

Having introduced the graph theoretical concepts that form the foundation of this work, we now continue to the concepts from complexity theory. Though all NP-complete problems are “equally hard”, additional information can make some problems easier to solve. The field of parameterized complexity formalizes this notion with the definitions of additional subclasses of NP; in this work we consider the classes XP and FPT (fixed parameter tractable).

Definition 14. Given an input of size $n$ and a positive integer parameter $k$, an algorithm belongs to the class XP if its time complexity is $O(n^{f(k)})$ for some computable function $f$.

Definition 15. Given an input of size $n$ and a positive integer parameter $k$, an algorithm belongs to the class FPT if its time complexity is $O(f(k) \cdot n^c)$ for some computable function $f$ and constant $c$.

Algorithms in XP and FPT can run efficiently when the parameter $k$ is small, but devolve into super-polynomial algorithms when $k$ is not a constant.

The computational complexity of FPT algorithms is often reported using $O^*$ notation, which hides the polynomial factors in the complexity. For example, an algorithm with complexity $O(2^k n^2)$ would be written $O^*(2^k)$. This is because the superpolynomial component of the run time dominates in the limit and is thus potentially the best way to compare two parameterized algorithms.
2.4.1 Relationship to Graph Structure

Parameterized complexity is strongly intertwined with structural graph theory because the structural characteristics of sparse classes can be exploited to solve otherwise intractable problems more efficiently. For some classes, the connection between structure and parameterized algorithms is straightforward. Classes of bounded treedepth, for instance, are characterized as having treedepths that do not grow with the size of graphs in the class. An algorithm that uses the treedepth as the integer parameter $k$ would have the value of $f(k)$ bounded by a constant independent of $n$ for all graphs in the class and the asymptotic complexity will be simply $O(n^c)$ for some constant $c$. The bounded treedepth property is often exploited by such an algorithm by performing some brute force computation over some set of inputs exponential in the treedepth. For example, it might take paths from leaves to the root of the treedepth decomposition and find the optimum value of a function over all $2^{td(G)}$ subsets of vertices on each path. Since the treedepth decomposition restricts the ways these root paths overlap, function values from different root paths could be combined using dynamic programming, ultimately leading to computing that function for the entire graph.

Exploiting the structural properties of other classes is not always as obvious. The absence of certain ($r$-shallow) minors does not immediately translate into an integer parameter necessary for a parameterized algorithm and hence algorithms exploiting excluding a minor and bounded expansion require more clever tricks. As will be discussed at length in Chapter 4, some algorithms for classes of bounded expansion make extensive use of $p$-centered colorings. Since bounded expansion classes admit $p$-centered colorings with a number of colors independent of $n$, enumerating the subgraphs induced on all subsets of $p - 1$ colors can be done in $O^*(\chi_p(G)^p)$ time. Since each of these subgraphs has bounded treedepth, algorithms parameterized by treedepth can be run on each of the subgraphs and then their results can be combined to generate a solution for the entire graph.

2.4.2 Parameterized Lower Bounds

It can be useful to determine when a fixed-parameter tractable algorithm has reached “best possible” asymptotic dependence on the parameter. One method for establishing a parameterized algorithm is by showing faster algorithms violate the exponential time hypothesis (ETH), which is a complexity-theoretical conjecture relating to the hardness of $k$-SAT.

**Input:** Boolean formula $\Phi$ in $k$-CNF form

**Question:** Is there an assignment of variables such that $\Phi$ evaluates to true?
Conjecture 1 ([47]). There is no algorithm for \(k\)-SAT whose worst-case time complexity is \(2^{o(n)}\) for a formula with \(n\) variables.

The ETH is a stronger assumption than \(P \neq \text{NP}\) since the latter would not exclude, for example, an algorithm for \(k\)-SAT running in \(O(2^{\sqrt{n}})\) time. To use the ETH to derive a lower bound for a graph problem \(\Pi\) parameterized by a structural property, we perform an NP-hardness reduction from \(k\)-SAT in which the structural parameter \(k'\) in the resulting graph satisfies \(k' \leq f(n)\) for some function \(f\) of the number of variables \(n\). In this case, a \(2^{o(f^{-1}(k'))}\) algorithm for \(\Pi\) would imply a \(2^{o(n)}\) algorithm for \(k\)-SAT, which is not possible unless the ETH is false.
CHAPTER

3

LOCALLY ESTIMATING CORE NUMBERS

Summary

“Central” vertices in a graph tend to be more critical in the spread of information or disease and play an important role in clustering/community formation. Identifying such “core” vertices has recently received additional attention in the context of network experiments, which analyze the response when a random subset of vertices are exposed to a treatment (e.g. inoculation, free product samples, etc). This notion of centrality can be expressed by the core numbers of the vertices. Existing algorithms for computing the core number of a vertex require the entire graph as input, an unrealistic scenario in many real world applications. Moreover, in the context of network experiments, the subgraph induced by the treated vertices is only known in a probabilistic sense. We answer Research Question 1 by introducing a method for estimating the core number based only on the properties of the graph within a region of radius $\delta$ around the vertex, and prove an asymptotic error bound on random graphs. Further, we empirically validate the accuracy of our estimator for small values of $\delta$ on a representative corpus of real data sets. Finally, we evaluate the impact of improved local estimation on an open problem in network experimentation posed by Ugander et al [99]. The work in this chapter was published in the proceedings of the 2014 International Conference on Data Mining [76].
3.1 Introduction

In a graph modeling complex interactions between data instances, the connectivity of the vertices often yields useful insights about the properties of the represented entities. For example, in a social network, it is important to distinguish between a person who is a member of a relatively large, tight-knit community, and a person who exists on the periphery without much involvement with any cohesive communities. This type of connectivity is often not well-correlated with the vertex degree (raw number of connections), leading to the introduction of several more sophisticated metrics. Here, we focus on the core number [6]. To build intuition, consider the setting of a graph representing friendships in a social network – the vertices with large core numbers form a set where people tend to have many common friends, whereas the subset of vertices with small core numbers form a group where most people do not know one another. The applications of core numbers are numerous, well studied, and permeate a variety of domains, such as community detection [39, 64], virus propagation [52], data pruning [4], and graph visualization [2].

Existing algorithms for computing core numbers run in linear time with respect to the number of vertices and edges of the graph [6], but require the entire graph as input and simultaneously calculate the core number for every vertex. There are a number of scenarios in which current approaches are insufficient. First, one might only be concerned with the properties of a small subset of query vertices. If the query vertices constitute a small fraction of the entire graph, it would be much more time- and memory-efficient to locally estimate the core numbers of those specific vertices rather than using the global algorithm, regardless of the fact that it is linear. Second, the entire graph may be unknown and/or infeasible to obtain due to scale, privacy, or business strategy reasons. For example, suppose one wanted to investigate patterns of phone calls between cell phone users. It would be possible to contact a small set of people and ask them to voluntarily log their calls for one month. However, without access to the telecommunication companies’ private records, expanding the domain to the national or global level would not be possible. Finally, current methods for determining core numbers cannot be applied to solve problems in the domain of network experimentation, as we describe below.

A network treatment experiment is a randomized experiment in which subjects are divided into two groups: those receiving treatment and those receiving none (or a placebo). Network treatments differ from other randomized experiments in that the effects of the treatment (or lack thereof) on a given subject are assumed to be dependent on the experiences of other subjects in the experiment. Randomly assigning subjects into two groups (treated versus untreated) is

\[\text{For example, vertices which have some common metadata (e.g. age, physical location, etc.) of interest to the user that is not represented explicitly in the graph}\]
equivalent to randomly partitioning the vertices of a graph into two sets. Previous work has
given methods to calculate degree probabilities of a randomly partitioned graph [99], but an
analogous algorithm for the core numbers is an open problem. Given the results of Kitsak et
al. on the importance of core numbers in spreading information [52], an algorithm to predict
the likelihood of a given subject having a large core number would help researchers better
understand the impact of their experimental design. For example, researchers conducting
market testing on a product that relies on social interaction (such as a new social networking
site, online game, etc.) would have a greater ability to see whether early access to their product
will generate widespread excitement among the test subjects. Additionally, if certain groups
of vertices (say, females participating in the experiment) are more likely to be core exposed
than the others, we can reduce the bias of the estimate of the treatment effect by upweighting
the probability that underrepresented vertices (males) are treated.

All of the challenges mentioned above and Research Question 1 could be addressed by
estimating the core number of a vertex based on local graph properties. The existence of an
accurate non-global estimator is intuitively well-grounded, as it has been shown that addition
and deletion of edges can only affect the core numbers of a limited subset of vertices [87]. This
suggests that in spite of the fact that computing core numbers exactly requires knowledge
of the whole graph, the core number of a given vertex may often depend on a much smaller
subgraph. Moreover, even if a core number estimate has a small error, it may still be useful
in applications. In particular, it is often sufficient to delineate between vertices with a “large”
core number and those with a “small” core number. That is to say, while a vertex in the 50-core
is substantially more well-connected than one in the 2-core, it may be functionally identical to
a vertex in the 51-core in downstream analysis.

This work introduces a new local estimator of the core number at a specified vertex, which
allows a user to tune the balance between accuracy and computational complexity by varying
the size of the local region around the query vertex that it considers. We prove that in an
Erdős-Rényi random graph, the error of our approximation at each vertex asymptotically
almost surely grows arbitrarily slowly with the size of the graph. We also empirically evaluate
the estimates with respect to the actual core numbers on a representative corpus of real-world
graphs of varying sizes. The results on these graphs demonstrate that high accuracy can
be achieved even when considering only a small local region. Finally, we show how our
estimators can be applied to address the aforementioned open problem in network treatment
experiments. Specifically, we give an algorithm to tighten the upper bound on the core number
of a vertex given in [99], and evaluate the impact empirically on a sample experiment.
3.2 Related Work

The basic k-core decomposition algorithm has been tailored to meet additional constraints. Montressor et al. [67] and Jakma et al. [48] proposed methods by which the core decomposition could be computed in parallel. Li et al. [87] and, later, Sarıyüce et al. [57] described ways to update the core numbers of vertices in a dynamic graph without recomputing the full core decomposition each time a vertex or edge is added. Finally, Cheng et al. [17] gave an alternate implementation of the core decomposition for systems with insufficient memory to store the entire graph at once.

3.3 Local Estimation

In order to estimate core numbers efficiently without knowledge of the entire graph, we will restrict the domain of our algorithms to a localized subset around a vertex:

**Definition 16.** The δ-neighborhood of a vertex v, denoted N_δ(v), is the set of vertices at distance at most δ from v.

The estimation algorithms will vary the size of their input by allowing δ to range from zero to ∆, the diameter of G (the maximum distance among all pairs of vertices).

3.3.1 Neighborhood-based Estimation

A relatively naïve approach to local estimation would be to compute a core decomposition of the subgraph of G induced on the δ-neighborhood of v and use the resulting core number of v as the estimate:

**Definition 17.** Let the induced estimator, ˘kδ(v), be the core number of v in G[N_δ(v)].

By increasing δ, the induced subgraph captures a progressively larger fraction of the graph, improving the estimate until ˘kδ(v) = k(v) (which is guaranteed to happen at δ = ∆, but could happen for significantly smaller δ). Note that when δ = 1, the estimator will only be close to the core number if the neighbors of v are highly interconnected (so there is a subtle relationship to clustering coefficient).

**Lemma 1.** Let G = (V, E) be a graph. For all v ∈ V,

\[ 0 = ˘k_0(v) \leq ˘k_1(v) \leq \cdots \leq ˘k_\Delta(v) = k(v) \]

\[ ^2 \text{We use the typical shortest-path distance function throughout} \]
Proof. First we will establish the boundary conditions on our inequality. Because \( N_\Delta(v) = G, \) \( \tilde{k}_\Delta(v) = k(v). \) Since \( N_0(v) = \{v\}, \tilde{k}_0 = 0. \) Additionally, for all \( \delta \in [1, \Delta], N_{\delta-1}(v) \subseteq N_\delta(v). \) This implies that for all \( u \in V, \) the degree of \( u \) in the subgraph induced by the \((\delta - 1)\)-neighborhood of \( v \) can be no greater than the degree of \( u \) in the \( \delta \)-neighborhood of \( v. \) Thus \( v \) cannot participate in a deeper core with respect to the \((\delta - 1)\)-neighborhood than it does with respect to the \( \delta \)-neighborhood, which makes \( \tilde{k}_{\delta-1} \leq \tilde{k}_\delta. \)

In order to make a more sophisticated estimation, let us consider the information gained as \( \delta \) increases. At \( \delta = 0, \) we assume that \( d(v) \) is known. Because the \( k \)-core requires all vertices to have minimum degree \( k, k(v) \) can not be greater than \( d(v). \) Thus, \( d(v) \) itself can be an estimate of \( k(v). \) Expanding out to \( \delta = 1 \) allows information about \( v \)'s immediate neighbors to be utilized. Suppose the core numbers of the neighbors of \( v \) were known. It would then be possible to compute \( k(v) \) precisely using the following two lemmas (previously shown by Montresor et al):

**Lemma 2 ([67]).** A vertex \( v \) is in the \( j \)-core of a graph \( G \) if and only if \( v \) has at least \( j \) neighbors in the \( j \)-core.

We now give a closed-form algebraic expression for the largest \( j \) satisfying Lemma 2.

**Lemma 3 ([67]).** Let \( u_1, u_2, \ldots \) be the \( k \)-ordered neighbors of \( v. \) Then

\[
k(v) = \max_{1 \leq i \leq d(v)} \left( \min\left(k(u_i), d(v) - i + 1\right) \right).
\]

Proof. By the definition of core number and Lemma 2, \( k(v) \) is the largest \( j \) in \( 0 \leq j \leq d(v) \) so that \( v \) has at least \( j \) neighbors with core number at least \( j. \) For each \( u_i \) \((1 \leq i \leq d(v)), v \) has \( d(v) - i + 1 \) neighbors with core numbers at least \( k(u_i), \) since \( k(u_i) \leq k(u_{i+1}). \) If \( k(u_i) \leq d(v) - i + 1, v \) has at least \( k(u_i) \) neighbors in the \( k(u_i) \)-core. Otherwise, \( v \) has at least \( d(v) - i + 1 \) neighbors in the \((d(v) - i + 1)\)-core. This shows that the core number of \( v \) must be at least the minimum of \( k(u_i) \) and \( d(v) - i + 1 \) for every \( i \) (and thus the maximum over \( i). \) Equality follows easily by contradiction.

Note that \( k(v) \) is the maximum value of the minimum of two functions of \( i: k(u_i) \) and \( d(v) - i + 1. \) With respect to \( i, k(u_i) \) is monotonically non-decreasing and \( d(v) - i + 1 \) is monotonically decreasing. From a geometric perspective, then, the maximum of their minimums occurs at the intersection of the curves \( k(u_i) \) and \( d(v) - i + 1 \) (as stylized in Figure 3.1).

Although the core numbers of the neighbors of a vertex may not be known a priori, the reasoning behind Lemma 3 gives useful insight into the behavior of \( k(v). \) As shown by Cheng et al. [17], an upper bound on \( k(v) \) can be achieved if an upper bound on \( k(u) \) is known for all \( u \in N_1(v). \)
**Theorem 1** ([17]). Let $G = (V, E)$ be a graph, $v \in V$, and $\psi$ any function satisfying $\psi(u) \geq k(u) \forall u \in V$. Let $v$’s neighbors be $\psi$-ordered. Then

$$k(v) \leq \max_{1 \leq i \leq d(v)} \left( \min \left( \psi(u_i), d(v) - i + 1 \right) \right).$$

**Proof.** Substituting $\psi(u_i)$ for $k(u_i)$ in the expression from Lemma 3 can only increase the right hand side, giving an upper bound on $k(v)$. \qed

We base our second estimator on the idea of incorporating iterative upper bounds on the core numbers of a vertex’s neighbors:

**Definition 18.** Let the **propagating estimator**, $\hat{k}_\delta$, be the estimator of $k(v)$ given by the recurrence

$$\hat{k}_\delta(v) = \begin{cases} 
\max_{1 \leq i \leq d(v)} \left( \min(\hat{k}_{\delta-1}(u_i), d(v) - i + 1) \right) & \text{if } \delta > 0 \\
\hat{k}_{\delta-1}(v) & \text{if } \delta = 0
\end{cases}$$

where $u_1, u_2, \ldots$ are the $\hat{k}_{\delta-1}$-ordered neighbors of $v$.

Pseudocode for computing $\hat{k}_\delta(v)$ is given in Algorithm 4. Essentially, Algorithm 4 first computes the coarsest upper bound ($\hat{k}_0$) for those vertices at distance at most $\delta$ from $v$. Those estimates are used in conjunction with Theorem 1 to compute a slightly finer upper bound, $\hat{k}_1$, for those vertices at distance at most $\delta - 1$ from $v$. This process “propagates” inwards towards $v$ until its immediate neighbors have $\hat{k}_{\delta-1}$ values, which are used as the upper bounds in formulating $\hat{k}_\delta(v)$. The computational complexity of finding $\hat{k}_\delta$ is linear in the product of $\delta$ and the number of edges in $N_\delta(v)$ (see Theorem 2). Since Algorithm 3 is also linear with respect to the number of edges in the graph, the computational complexity of computing $\hat{k}_\delta$ is comparable to that of $\bar{k}_\delta$ for small $\delta$. 

---

**Figure 3.1** The core number $k(v)$ is the $y$-value at intersection of two functions.
Algorithm 4 prop_estimator($v, \delta$)

Input: Vertex $v$, distance $\delta$

Output: Propagating core number estimator of $v$ at distance $\delta$, i.e., $\hat{k}_\delta(v)$

1: if $\delta = 0$ then
2: return $d(v)$
3: else
4: for $u \in N_1(v)$ do
5: $\hat{k}_{\delta-1}(u) \leftarrow$ prop_estimator($u, \delta - 1$)
6: end for
7: $\hat{k}_\delta(v) \leftarrow d(v)$
8: for $i \in \{1, \ldots, |N_1(v)|\}$ do
9: $u_i \leftarrow$ vertex in $N_1(v)$ the $i$th smallest value of $\hat{k}_{\delta-1}$
10: $j \leftarrow \min(\hat{k}_{\delta-1}(u_i), d(v) - i + 1)$
11: if $j > \hat{k}_\delta(v)$ then
12: $\hat{k}_\delta(v) \leftarrow j$
13: end if
14: end for
15: return $\hat{k}_\delta(v)$
16: end if

Theorem 2. For a given vertex $u$, $\hat{k}_\delta(u)$ can be computed in $O(\delta \cdot |E_\delta(u)|)$ time using Algorithm 4, where $E_\delta(u)$ is the edge set of $G[N_\delta(u)]$.

Proof. Fix $\delta \in \{1, 2, \ldots\}$ and $u \in V$. Assume $\forall u' \in N_1(u)$, $\hat{k}_{\delta-1}(u')$ is known. Since $\hat{k}_\delta$ can only take integer values in the interval $[0, \max_{u \in V} d(v)]$, the sorting (line 9) can be done in $O(d(u))$ time using a bucket sort. Once sorted, each neighbor of $u$ is visited once (line 8) to find the minimum, which can also be done in $O(d(u))$ time. Thus computing $\hat{k}_\delta(u)$ from $\{\hat{k}_{\delta-1}(u')|u' \in N_1(u)\}$ has complexity $O(d(u))$.

Using dynamic programming, we can compute and store $\hat{k}_1(w) \forall w \in N_{\delta-1}(v)$, which in turn can be used to compute $\hat{k}_2(w) \forall w \in N_{\delta-2}(v)$, and so on (through $\delta - 1$ iterations) until we have computed $\hat{k}_\delta(v)$. The $j$th such iteration requires $\sum_{u \in N_{\delta-j}} O(d(u)) = O(E_{\delta-j}(v))$ operations. Since $N_{\delta-1} \subseteq N_\delta$, the total running time is $O(\delta \cdot |E_\delta|)$. \hfill \square

Unlike $\bar{k}_\delta(v)$, the estimate $\hat{k}_\delta(v)$ is a decreasing upper bound on $k(v)$ as $\delta$ increases:

Theorem 3. $\forall v \in V, k(v) \leq \hat{k}_\delta(v) \leq \hat{k}_{\delta-1}(v) \cdots \leq \hat{k}_1(v) \leq \hat{k}_0(v) = d(v)$ for any $\delta \geq 1$.

Proof. We first prove that $k(v) \leq \hat{k}_\delta(v)$ for all $\delta \geq 0$ by induction on $\delta$. The base case $k(v) \leq \hat{k}_0(v) = d(v)$ holds, since core number is always bounded by degree. Assume $k(v) \leq \hat{k}_\delta(v)$. Then $\hat{k}_\delta$ is an upper bound $\psi$ as in Theorem 1, and substituting the right hand side with Definition 18, we have $k(v) \leq \bar{k}_{\delta+1}(v)$. 

24
We now prove that $\hat{k}_{j+1}(v) \leq \hat{k}_j(v)$ for all $j \geq 0$ and $\forall v \in V$ by induction on $j$. Combining Definition 18 with Lemma 3, we see that $\hat{k}_\delta(v)$ is the maximum of the minimum of the functions $\hat{k}_{\delta-1}(u_i)$ and $d(v) - i + 1$ for $1 \leq i \leq d(v)$. Since the maximum of $d(v) - i + 1$ is $d(v)$, $\hat{k}_\delta(v) \leq d(v)$. Because $\hat{k}_0(v) = d(v)$ the base case $\hat{k}_1(v) \leq \hat{k}_0(v)$ is satisfied. Suppose that for some $j \geq 0$, $\hat{k}_j(v) \leq \hat{k}_{j-1}(v)$ for all $v \in V$. By Theorem 1, $v$ has at least $\hat{k}_j(v)$ neighbors that satisfy $\hat{k}_{j-1}(u) \geq \hat{k}_j(v)$ for $u \in N_1(v)$. Each such $u$ also satisfies $\hat{k}_j(u) \leq \hat{k}_{j-1}(u)$, meaning $v$ can have no more than $\hat{k}_j(v)$ neighbors that satisfy $\hat{k}_j(u) \geq \hat{k}_j(v)$. Thus $\hat{k}_{j+1}(v) \leq \hat{k}_j(v)$. \hfill \Box

### 3.3.2 Structures Leading to Error

One natural question is whether either $\hat{k}_\delta$ or $\tilde{k}_\delta$ has bounded error (is a constant-factor approximation of the core number). Unfortunately, there are extremal constructions forcing unbounded error for both estimators; both are based on $T_{j,\ell}$, the complete $j$-ary tree with $\ell$ levels (labeled so that level $i$ has $j^{i-1}$ vertices), rooted at a vertex $v$ (Figure 3.2).

**Lemma 4.** For all integers $\delta \geq 0$, $x \geq 1$, there exists a graph $G$ and vertex $v$ so that

$$\hat{k}_\delta(v) - k(v) = x - 1.$$  

**Proof.** First note that since $T_{j,\ell}$ is a tree, it is 1-degenerate. We show that the root vertex $v$ has $\hat{k}_\delta$ estimators with unbounded error. For any vertex $u$ with level number $i$ in $[2, \ell - \delta - 1]$, every vertex not equal to $v$ in $u$’s $\delta$-neighborhood has degree $j + 1$. As a result, $d(u) = \hat{k}_1(u) = \cdots = \hat{k}_{i-1} = j + 1$, and $\hat{k}_i = \cdots = \hat{k}_\delta = j$ (since $v$ has degree $j$ and will have propagated inwards). Thus, for any $\delta \leq \ell - 1$, we have $\hat{k}_\delta(v) - k(v) = j - 1$. \hfill \Box

![Figure 3.2](image_url)
Lemma 5. For all integers \( \delta \geq 0, x \geq 1 \), there exists a graph \( G \) and vertex \( v \) so that

\[
\hat{k}_\delta(v) - k(v) = x - 1.
\]

Proof. Consider the graph \( T_{j,\ell} \) created by adding vertices \( w_1, \ldots, w_j \) to \( T_{j,\ell} \) connected to the leaves of \( T_{j,\ell} \) as apices (see Figure 3.2). Then the root \( v \) is the vertex of minimum degree in \( T_{j,\ell} \) and has core number \( j \). Any induced subgraph of \( T_{j,\ell} \) that does not include at least one \( w_i \) is a tree, making it 1-degenerate. Thus \( k(v) - \hat{k}_\delta(v) = j - 1 \) whenever \( \ell \geq \delta + 1 \).

Note that in \( T_{j,\ell} \), \( \hat{k}_\delta(v) = k(v) \) for any \( \delta > 0 \). Likewise, in \( T'_{j,\ell} \), \( \hat{k}_\delta(v) = k(v) \) for any \( \delta \geq 0 \). Despite the fact that the errors of both estimators can theoretically be arbitrarily large, structures causing egregious errors (like \( T_{j,\ell} \) and \( T'_{j,\ell} \)) are unlikely to occur in real world networks; we provide evidence to support this claim in the next sections.

3.3.3 Expected Behavior on Random Graphs

In order to better understand the errors generated when approximating core number with \( \hat{k}_\delta \), we analyze its behavior on Erdős-Rényi random graphs. To avoid confusion, we use \( G(n, p) \) to denote the set of all Erdős-Rényi random graphs with \( n \) vertices and edge probability \( p \) and \( G(n, p) \) for a specific instance. Since all graphs on \( n \) vertices occur in \( G(n, p) \) with non-zero probability (when \( p \in (0, 1) \)), analysis typically focuses on whether a graph property is very likely (or unlikely) to occur as the size of an Erdős-Rényi random graph grows large. In keeping with prior work ([49, 62, 80]), we assume the average degree is fixed, letting \((n - 1)p = d\), a constant.

Specifically, we focus our attention on the growth of the error term \( \hat{k}_1(v) - k(v) \) as \( n \to \infty \) by deriving probabilistic expressions for \( \hat{k}_1(v) \) and \( k(v) \) for any \( v \in G(n, p) \), then demonstrating how each term grows with \( n \) compared to a function in \( \omega(1) \) (recall a function \( f(n) \) is \( \omega(1) \) if \( \lim_{n \to \infty} \frac{1}{f(n)} = 0 \).

Theorem 4. Suppose \( e(n) \) is \( \omega(1) \). Then for any \( v \in G(n, p) \), \( \hat{k}_1(v) - k(v) \) is \( O(e(n)) \) a.a.s.

Proof. Fix a vertex \( v \), and let \( S_{\geq \kappa}, S_{< \kappa}, \) and \( S_{= \kappa} \) be defined to be the subsets of \( N_1(v) \) with vertices of degree greater than, equal to, or less than \( \kappa \), respectively. We first evaluate \( \mathbb{P}[\hat{k}_1(v) = \kappa | d(v) = d] \). By Definition 18, if \( \hat{k}_1(v) = \kappa \), \( v \) has at least \( \kappa \) neighbors \( u \) with \( \hat{k}_0(u) \geq \kappa \) but less than \( \kappa + 1 \) with \( \hat{k}_0(u) \geq \kappa + 1 \) (or else \( \hat{k}_1(v) > \kappa \)). Therefore, \( \hat{k}_1(v) = \kappa \) implies \( |S_{\geq \kappa}| \leq \kappa \) and \( |S_{= \kappa}| + |S_{< \kappa}| \geq \kappa \), and

\[
\mathbb{P}[\hat{k}_1(v) = \kappa | d(v) = d] = \sum_{i=0}^{\kappa} \frac{d!}{i!} \mathbb{P}[\left(|S_{> \kappa}| = i \right) \wedge \left(|S_{< \kappa}| = j \right) \wedge \left(|S_{= \kappa}| = x \right)],
\]
where \( x = d - i - j \).

As \( n \) tends to infinity, the probability that any two neighbors of \( v \) have an edge between them approaches 0. Therefore, the degrees of \( v \)'s neighbors can be treated as independent, identical distributions in the limit. Since \( n \) is large and \( \bar{d} \) is fixed, this distribution is asymptotically Poisson with mean \( \bar{d} \). If \( \zeta_{\lambda}(k) \) and \( Z_{\lambda}(k) \) denote the Poisson probability mass function and cumulative distribution function, respectively, with mean \( \lambda \) (evaluated at \( k \)), then:

\[
\Pr[\hat{k}_1(v) = \kappa | d(v) = d] = \sum_{i=0}^{\kappa-1} \sum_{j=0}^{\kappa-1} \frac{d!}{i!j!x!} (1 - Z_{\bar{d}}(\kappa - 1))^i Z_{\bar{d}}(\kappa - 2)^j \zeta_{\bar{d}}(\kappa - 1)^x. \quad (3.1)
\]

By computing Equation 3.1 at each value of \( d \) for which \( \kappa \) is a possible value for the core number, we have:

\[
\Pr[\hat{k}_1(v) = \kappa] = \sum_{d=\kappa}^{\kappa-1} \zeta_{\bar{d}}(d) \cdot \Pr[\hat{k}_1(v) = \kappa | d(v) = d]. \quad (3.2)
\]

Pittel et al. [80] demonstrated that in \( G(n,p) \), the proportion of vertices in the \( k \)-core is a.a.s. a function of \( \bar{d} \) but not of \( n \). Moreover, for any vertex \( v \), \( k(v) \) is a.a.s. bounded by a constant (equivalently, in \( \Theta(1) \)). If \( \hat{k}_1(v) \) were also bounded by a constant, then the error term \( \hat{k}_1(v) - k(v) \) would be a.a.s. \( O(1) \). However, since the Poisson random variables in Equations 3.1 and 3.2 are only parameterized by \( \bar{d} \) and not by \( n \), the proportion of vertices in \( G(n,p) \) with \( \hat{k}_1 = \kappa \) is a.a.s. convergent to some non-zero constant. Thus, the probability of having an arbitrarily large value of \( \hat{k}_1 \) does not vanish as \( n \) grows large for a fixed (constant) \( \kappa \).

Let \( \kappa \) be a function of \( n \) in \( \omega(1) \). By Stirling’s approximation of the factorial,

\[
\zeta_{\bar{d}}(\kappa) \approx \frac{e^{-\bar{d}}}{\sqrt{2\pi\kappa}} \left( \frac{e\bar{d}}{\kappa} \right)^\kappa.
\]

Then as \( n \) grows large, \( \zeta_{\bar{d}}(\kappa) \to 0 \) and \( Z_{\bar{d}}(\kappa) \to 1 \). In Equation 3.1, the probability that a neighbor \( u \) of vertex \( v \) has degree at least \( \kappa \) (that is, \( u \in S_{>\kappa} \cup S_{=\kappa} \)) is asymptotically zero, and consequently \( \Pr[\hat{k}_1(v) = \kappa] \) also vanishes in the limit. This implies that a.a.s. \( \Pr[\hat{k}_1(v) = \kappa] \in O(\epsilon(n)) \) for any error function \( \epsilon(n) \in \omega(1) \). Using the result of [80] that \( k(v) \in \Theta(1) \), we have a.a.s. \( \hat{k}_1(v) - k(v) \in O(\epsilon(n)) \).

\[\square\]

### 3.4 Experimental Results

In the previous section, the behavior of the propagating estimator \( \hat{k}_\delta \) was analyzed from a theoretical perspective. In order to enhance this picture, we present computational results on a corpus of real data.
Table 3.1 Summary statistics for real-world graphs used in evaluating local core estimators.

| Graph                  | $|V|$  | $|E|$  | $\text{max } d(v)$ | $D$ | $\Delta$ |
|------------------------|------|------|------------------|----|--------|
| Amazon[94]             | 334863 | 925872 | 549              | 6  | 47     |
| Co-purchases           |      |      |                  |    |        |
| Autonomous systems     | 6214  | 12232 | 1397             | 12 | 9      |
| ca-AstroPh[94]         | 17903 | 196972 | 504              | 56 | 14     |
| Academic citations     |      |      |                  |    |        |
| DBLP[94]               | 317080 | 1049866 | 343             | 113| 23     |
| Academic citations     |      |      |                  |    |        |
| Enron[94]              | 33696  | 180811 | 1383             | 43 | 13     |
| Email correspondence   |      |      |                  |    |        |
| Facebook[98]           | 36371  | 1590655 | 6312             | 81 | 7      |
| Facebook friendship    |      |      |                  |    |        |
| Gnutella[94]           | 26498  | 65359  | 355              | 5  | 11     |
| Peer-to-peer fileshearing |      |      |                  |    |        |
| H. sapiens[7]          | 18625  | 146322 | 9777             | 47 | 10     |
| Protein-protein interation |      |      |                  |    |        |
| WPG[46]                | 4941  | 6594  | 19               | 5  | 46     |
| Western US power grid  |      |      |                  |    |        |

3.4.1 Methods

The estimators $\hat{k}_\delta$ and $\tilde{k}_\delta$ were evaluated on nine real-world graphs which cover a variety of domains and are structurally dissimilar. The graphs vary in size, density, core structure, and diameter (see Table 3.1 and Figure 3.3).

We computed the core number $k(v)$, as well as the values of $\hat{k}_\delta(v)$ and $\tilde{k}_\delta(v)$ for each vertex, letting $\delta$ vary from 0 to $\Delta$. To compare the accuracy of the estimators among vertices, we normalize by the true core number at each vertex. We refer to this metric as the core number estimate ratio. When the estimator ($\hat{k}_\delta$ or $\tilde{k}_\delta$) is exactly equal to the core number, the core number estimate ratio is 1, its optimal value. Since $\hat{k}_\delta$ is an upper bound on $k$, its core number estimate ratio is always at least one and becomes less optimal the larger it gets; the opposite is true for $\tilde{k}_\delta$, a lower bound.

3.4.2 Results

We first turn our attention to how often the estimators achieve optimal core number estimate ratios. Figure 3.4 shows how the proportion of vertices with ratio one grows as $\delta$ increases from zero to four. In all the graphs, the core number estimate ratio for $\hat{k}_\delta$ is optimal at least as often as that of $\tilde{k}_\delta$ at $\delta \leq 1$. Additionally, the proportion of vertices with optimal $\hat{k}_\delta$ estimate...
ratios is large in all the graphs (often upwards of 90%). While the propagating estimator does not have as pronounced of an advantage over the induced estimator when $\delta = 2$, the number of vertices with optimal $\hat{k}_\delta$ estimate ratios still grows noticeably.

We also examined the distribution of core number estimate ratios among those vertices where the estimate was not exact. The change in this distribution over the range $\delta = 1$ to $\delta = 4$ is shown in Figure 3.5, demonstrating that not only are the sub-optimal estimates closely centered around 1, but also that increasing $\delta$ can significantly decrease the size of the “tail” of the distribution (thereby improving the core number estimates of those vertices with the least optimal ratios).

Although Figures 3.4 and 3.5 suggest that $\hat{k}_\delta$ and $\tilde{k}_\delta$ can accurately estimate the core numbers in real world graphs using only knowledge of the $\delta$-neighborhood with a small value of $\delta$, it is important to understand how the size of the $\delta$-neighborhood impacts the behavior of the estimates. The purpose of having a localized estimate is to reduce the size of the input needed to compute the core number of a vertex. If the average $\delta$-neighborhood encompasses most of the graph, then not only is this purpose defeated, but we also may not be able to judge whether the accuracy of the localized estimates is only due to having knowledge of the entire graph (as opposed to any theoretical merits of the algorithms themselves). The mean and variance of the proportion of vertices in the $\delta$-neighborhood is shown in Table 3.2. The rate of growth of $\delta$-neighborhood sizes varies significantly among the nine graphs, which suggests that picking a value of $\delta$ to maintain appropriately small $\delta$-neighborhoods is highly dependent on the structure of the graph. Nonetheless, the average size of the $\delta$ neighborhood is below ten percent of the entire graph for all datasets at $\delta = 1$ and in all but one (namely Facebook, which we know to be significantly different from the other networks) at $\delta = 2$. 

**Figure 3.3 Core number distribution for the real world networks in Table 3.1.**
Figure 3.4 Proportion of vertices with optimal core number estimate ratios for the propagating (solid green) and induced (dashed blue) estimators as a function of $\delta$. 
Another natural way to measure the relative amount of information in the $\delta$-neighborhood is to normalize by the diameter $\Delta$. Figure 3.6 shows that the average proportion of vertices in the $\delta$-neighborhood is approximately uniform in all graphs when $\delta$ is expressed as a fraction of $\Delta$. In particular, there is a significant increase in the rate of growth of the $\delta$-neighborhood size occurring when $\delta$ is approximately 20% of the diameter. Thus, as one might expect,

![Graphs showing the relationship between $\delta$ and the proportion of vertices in the $\delta$-neighborhood.](image)

**Figure 3.5** Number of vertices with core number estimate ratios less optimal that a given threshold from $\delta = 1$ (lightest line) to $\delta = 4$ (darkest line). $\hat{k}_\delta$ is shown in green while $\bar{k}_\delta$ is shown in blue. Because the number of vertices with optimal ratios is frequently large (see Figure 3.4), the vertices with optimal ratios may not appear within the limits of the plot in order better capture the distribution of those vertices with suboptimal ratios.
Table 3.2 Proportion of vertices in $N_\delta$. Values less than .01 rounded to 0.

<table>
<thead>
<tr>
<th>Graph</th>
<th>$\delta = 1$ Avg. Var.</th>
<th>$\delta = 2$ Avg. Var.</th>
<th>$\delta = 3$ Avg. Var.</th>
<th>$\delta = 4$ Avg. Var.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AS</td>
<td>0</td>
<td>.09 .01</td>
<td>.44 .07</td>
<td>.82 .04</td>
</tr>
<tr>
<td>ca-AstroPh</td>
<td>0</td>
<td>.03 0</td>
<td>.25 .04</td>
<td>.66 .06</td>
</tr>
<tr>
<td>DBLP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.03 0</td>
</tr>
<tr>
<td>Enron</td>
<td>0</td>
<td>.03 0</td>
<td>.28 .04</td>
<td>.74 .06</td>
</tr>
<tr>
<td>Facebook</td>
<td>0</td>
<td>.21 0.03</td>
<td>.89 0.02</td>
<td>1 0</td>
</tr>
<tr>
<td>Gnutella</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.02 0</td>
</tr>
<tr>
<td>H. sapiens</td>
<td>0</td>
<td>.31 0.07</td>
<td>.81 0.04</td>
<td>.98 0</td>
</tr>
<tr>
<td>WPG</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.01 0</td>
</tr>
</tbody>
</table>

Neighborhood-based core number estimates seem most appropriate when $\delta$ is a small fraction of the diameter.

Figure 3.6 Average proportion of vertices in $N_\delta$ as a function of $\delta/\Delta$. 
To see the effect of this normalized setting on accuracy, consider Figure 3.7. After normalizing $\delta$ by the diameter, the variation between graphs is less pronounced. Even when $\delta$ is small compared to $\Delta$, the optimum core number estimate ratio can be achieved. It is worthy to note that Facebook is a stark exception in which a significant proportion of vertices cannot achieve an optimal core number estimate ratio even when $\delta = \Delta$. This graph has many vertices with $d(v) \gg k(v)$ and a small diameter. Thus, many vertices have very inaccurate $\hat{k}_0$-values, which propagate inwards and remain uncorrected due to the small number of refinements performed on the estimate. Ultimately, we conclude that $\hat{k}_\delta$ best achieves its goal of accurately estimating the core number using only a small local section of the graph when the graph has a large diameter and the ratio $\delta/\Delta$ is small (e.g. less than 0.2).

3.5 Network Treatment

We now turn to the domain of network experiments and use the $\hat{k}_\delta$ estimator to address an open problem given in [99].

3.5.1 Problem Statement

Recall from the introduction that a network treatment experiment is a random experiment in which some subjects are given a treatment and the rest are not. It differs from other experiments in that the effects of the treatment are assumed to be dependent on interactions between subjects, which can be modeled by a graph. The general goal is to measure the subjects’ experiences in a hypothetical universe where the entire graph is treated by observing the experience of the subject when only some of the graph is treated. Ugander et al. [99] focused on local properties of the vertices to compare these two scenarios. In particular, they identified two useful ways to concretely measure the experience of a subject via graph properties:

**Definition 19 ([99]).** A vertex $v$ experiences absolute $k$-degree exposure if $v$ and at least $k$ of $v$’s neighbors receive treatment.

**Definition 20 ([99]).** A vertex $v$ experiences absolute $k$-core exposure to a treatment condition if $v$ belongs to the $k$-core of the graph $G[V']$, where $V'$ is the set of treated vertices.

We will use $X_k^{(d)}(v)$ and $X_k^{(c)}(v)$ to denote the events that a vertex $v$ experiences absolute $k$-degree and absolute $k$-core exposure, respectively.

In order to reduce variance in later sections of their analysis, Ugander et al. first cluster the graph and then assign treatment randomly to the clusters (as opposed to individual subjects). If a cluster is chosen to be treated, all vertices in the cluster receive treatment; otherwise none of them do. Ugander et al. utilize a $3$-net clustering that is formed by growing balls of radius
Figure 3.7 Proportion of vertices with optimal core number estimate ratios for the propagating estimator (solid green) and the induced estimator (dashed blue) as a function of $\delta$. The x-axis has been normalized by the diameter.

two centered at randomly selected vertices until every vertex is covered by some ball. The procedures for computing the probabilities of $X_k^{(d)}(v)$ and $X_k^{(c)}(v)$ are independent of the method by which the graph was clustered, so we choose to omit further detail here and refer the reader to [99] for details.

Once the graph is clustered, a recursive function can be used to compute the probability that vertex $v$ experiences absolute $k$-degree exposure. We follow the notation of [99]. Let $s$ be the number of clusters that contain at least one vertex in $\{v\} \cup N_1(v)$, indexed $\{1, \ldots, s\}$ so
that $v$ resides in the highest numbered cluster. If $p$ is the probability that a cluster is treated and $\vec{w}_v = (w_{v,1}, \ldots, w_{v,s})$ is the number of edges from $v$ to the vertices in each cluster, then

$$\mathbb{P}[X_k^{(d)}(v)] = pf(s-1, \kappa - w_{v,s}; p, \vec{w}_v),$$  

(3.3)

where the function $f(j, T; p, \vec{w}_v)$ is defined as

$$f(1, 0; p, \vec{w}_v) = 1$$
$$f(1, T; p, \vec{w}_v) = p\mathbf{1}[T \leq w_{v,1}]$$
$$f(j, T; p, \vec{w}_v) = pf(j-1, T - w_{v,j}; p, \vec{w}_v)$$
$$+ (1-p)f(j-1, T; p, \vec{w}_v),$$

where $\mathbf{1}[B]$ denotes the indicator function (evaluates to 1 if the Boolean expression $B$ is true and 0 otherwise).

The function $f$ defined above recursively visits each cluster $j$ containing a neighbor of $v$ and considers the probability that $v$ is $T$-degree exposed in the first $j$ clusters conditioned on whether cluster $j$ receives treatment. If $j$ is treated, $v$ needs to have $T - w_{v,j}$ treated neighbors in the first $j-1$ clusters; otherwise, it needs $T$ such neighbors. It follows from Definition 19 that if $v$ is $k$-degree exposed, cluster $s$ is necessarily treated. This also implies that all of $v$’s neighbors in the same cluster are necessarily treated as well. Thus, we are ultimately concerned with finding $\kappa - w_{v,s}$ treated neighbors in the remaining $s-1$ clusters that contain a neighbor of $v$. Using dynamic programming, we can compute $\mathbb{P}[X_i^{(d)}(v)]$ for all $0 \leq i \leq \kappa$ in $O(s\kappa)$ time.

### 3.5.2 Estimating $k$-core Exposure Probabilities

In [99], Ugander et al. left computing the exact probability of absolute $k$-core exposure as an open problem, since the core decomposition requires knowledge of the entire graph. They instead defer to the fact that the absolute $k$-core exposure probability is bounded from above by the absolute $k$-degree exposure probability and use the latter in lieu of the former. This is problematic because there may not be a consistent relationship between $d(v)$ and $k(v)$. For example, if $v$ is a vertex with degree 100 and core number 10, $\mathbb{P}[X_1^{(c)}] = 0$ independent of $\mathbb{P}[X_0^{(d)}]$. Although this is an extreme case, there are more general cases where the two probabilities are not correlated. Specifically, vertices that require a large value of $\delta$ before $\hat{k}_\delta(v) = k(v)$ (as in Figure 3.8) can have many treated neighbors without having a large core number.

Recall that $\hat{k}_0(v)$ is the degree of $v$. As we have shown, even expanding the scope and computing $\hat{k}_1(v)$ can yield a considerably more accurate estimate of of the core number than the degree. Therefore, a tighter bound of the core exposure probability can be achieved by
Figure 3.8 $T'_{3,3}$ with 13 of 16 vertices treated. $k(v) = k(u_1) = k(u_2) = k(u_3) = 3$. Although $v$, $u_1$, $u_2$, and $u_3$ have all of their neighbors treated, they only have core number 1 with respect to the treated subgraph.

examining the degree exposure probability of $v$’s neighbors. To capture this, we introduce a $k_1$-related condition we call neighbor-degree exposure.

**Definition 21.** A vertex $v$ experiences **absolute $k$-neighbor-degree exposure** if at least $k$ of $v$’s neighbors experience absolute $k$-degree exposure.

We denote the event that vertex $v$ is absolute $k$-neighbor-degree exposed with $\hat{X}_k(v)$. Algorithm 5 gives a method for computing $P[\hat{X}_k(v)]$, which can then be used to estimate (specifically, find an upper bound on) $P[X_\kappa^c(v)]$.

**Algorithm 5** aknde-prob($G, v, C, p, \kappa$)

**Input:** Graph $G$, vertex $v$, clustering $C$, exposure probability $p$, desired exposure level $\kappa$

**Output:** Absolute $k$-neighbor-degree exposure probability of $v$, $P[\hat{X}_k(v)]$

1: Let $C(x)$ denote the cluster containing $x$
2: $C \leftarrow \left(\bigcup_{u \in N_2(v)} C(u)\right) \setminus C(v)$
3: $\hat{p}_\kappa \leftarrow 0$
4: **for** $S \subseteq C$ **do**
5: \[Y \leftarrow \{u \in N_1(v) : X_k^{(d)}(v) \text{ is true in } G[S \cup C(v)]\}\]
6: **if** $|Y| \geq k$ **then**
7: \[\hat{p}_\kappa \leftarrow \hat{p}_\kappa + p|S| + 1 (1 - p)^{|C| - |S|}\]
8: **end if**
9: **end for**
10: **return** $\hat{p}_\kappa$

The algorithm iterates through all subsets of clusters containing a vertex in $v$’s 2-neighborhood.
and determines whether treating them yields a scenario where \( v \) has \( k \) neighbors that are absolute \( k \)-degree exposed. If so, line 7 adds the probability of that configuration occurring to the final probability. Because it enumerates all possible subsets of \( C \), Algorithm 5 will run in \( O(sK \cdot d(v) \cdot 2^s) \) time in the worst case. While this algorithm is exponential in the number of clusters, Ugander et al. assume that the graph satisfies some restricted growth conditions\(^3\). In this case, the number of clusters that contain vertices from \( N_2(v) \) does not grow with respect to the size of the graph [99] which bounds the running time at \( O(K \cdot d(v)) \).

In graphs failing the restricted growth requirements, the running time can still be improved. Note that if treating a specific subset of clusters \( S \) on line 4 does not yield \( K \) vertices in \( N_1(v) \) that are \( K \)-degree exposed, then treating any \( S' \subseteq S \) also cannot yield at least \( K \) vertices in \( N_1(v) \) that are \( K \)-degree exposed. Thus, if the subsets of \( C \) are enumerated in decreasing order of their sizes, we can prune the search space to avoid needless computation. Moreover, the clustering algorithm can be biased towards selecting 3-net clusterings that minimize \( |C| \).

For example, one possible bias would be to select the centers of the balls with probability proportional to their degrees.

We applied Algorithm 5 and Equation 3.3 to the WPG data set (Table 3.1) and binned the data based on the difference \( P[\hat{X}_K] - P[X_k^{(d)}] \) as shown in Figure 3.9. It is particularly noteworthy that multiple vertices have a neighbor-degree exposure probability of zero but a non-zero probability of degree exposure. Moreover, many of those vertices have their degree exposure probability maximized (equal to 0.25). Thus, the empirical data confirms that absolute degree exposure probability may be a misleading estimate of absolute core exposure probability.

Finally, we consider a second approach for improving the approximation of \( P[X_k^{(c)}(v)] \) that, like \( \hat{k}_d \), tightens an upper bound on \( P[X_k^{(c)}(v)] \) by using the bounds on \( P[X_k^{(c)}(u)] \) for \( u \) in \( N_1(v) \). We examine those vertices \( u' \) in \( N_1(v) \) that satisfy \( P[X_k^{(d)}(u')] = 0 \). These vertices cannot contribute to \( P[\hat{X}_K(x)] \), so we can disregard them when computing \( \tilde{w}_v \). Thus, we can use Equation 3.3 with a modified \( \tilde{w}_v \) to get a tighter upper bound on \( P[X_k^{(c)}] \). Figure 3.10 shows that pruning can decrease the probability of a majority of the vertices (in fact, many probabilities decrease from \( p \) to 0). This further bolsters our argument that \( X_k^{(c)}(v) \) is only weakly correlated with \( X_k^{(d)}(v) \), but using information from \( v \)'s neighbors can yield a much tighter upper bound at minimal additional cost.

### 3.6 Conclusion

We introduced \( \hat{k}_d \), a novel method of estimating the core number of a vertex that uses only the data available in the \( \delta \)-neighborhood of the vertex. We formally proved that in an Erdős-Rényi

---

\(^3\)Namely, \( \exists c \) such that \( |N_{\delta+1}(v)| \leq c \cdot |N_{\delta}(v)| \) \( \forall v \in V \)
graph, the error of $\hat{k}_1$ grows arbitrarily slowly with respect to the size of the graph. After computing $\hat{k}_2$ on a representative corpus of real-world networks, we demonstrated that a high-accuracy estimate of the core number can be achieved using a limited subset of the graph. Finally, we described two ways in which the estimators could be used to improve calculations and accuracy in network treatment experiments.
Figure 3.10 Histogram of differences between $P[X_{k}^{(d)}]$ before and after pruning for $\kappa = 7$ and $p = 0.25$. The $x$-axis gives the difference in probability, while the $y$-axis gives the proportion of vertices occurring in that bin. For clarity, only those vertices which are not pruned are considered in the plot.
Summary

We answer Research Question 2 by evaluating existing algorithms for counting subgraph isomorphisms in bounded expansion graph classes. The specific graph class was chosen for its “realistic” properties, while the choice of subgraph isomorphism counting reflected its common use in data analytics. The structural algorithm of interest follows an intricate four-stage pipeline that colors the graph, decomposes it into well-structured subgraphs, counts isomorphisms in each of these subgraphs, and then combines the numerous counts appropriately. In order to evaluate such an algorithm, we developed its first known implementation, in an open-source package called CONCUSS, and ran four experiments. The first determined the relative speed of various configurations of subroutines in CONCUSS. The second pitted CONCUSS against an existing algorithm for subgraph isomorphism counting that does not explicitly exploit structural sparsity. The third experiment tested how different \( p \)-centered colorings affect the runtime of CONCUSS. In the final experiment, we determined how the runtime varies when performing the dynamic programming on subgraphs of different treedepths. The work in this chapter is under submission to the 26th Annual European Symposium on Algorithms [77].
4.1 Introduction

Based on the results on random graph models presented in Section 2.3 and their intuitive characterization as sparse with localized density, bounded expansion seems to be a promising candidate for a graph class that gives useful algorithmic advantages without imposing constraints unrealistic for many data sets. For this reason, Research Question 2 focuses on the evaluating the practicality of algorithms for classes of bounded expansion.

Our main contribution in this chapter is a set of computational experiments using CONCuss [75], an open source software tool written in pure Python that implements the best known algorithm for subgraph isomorphism counting (Section 4.3) in bounded expansion classes [27]. Our experimental design is oriented towards two primary goals: (1) evaluating the extent to which CONCuss is currently competitive with existing algorithms for subgraph isomorphism counting and (2) identifying bottlenecks in the algorithmic pipeline of CONCuss to guide further theoretical research. Theoretical advances to this pipeline would not only enable faster subgraph isomorphism counting, but also provide an improved framework for solving other combinatorial problems in graph classes of bounded expansion (see Section 4.2). To these ends, our experiments answer four questions:

Q1. What choices of subroutines give the best performance in CONCuss?
Q2. How does CONCuss compare to other algorithms for subgraph isomorphism counting?
Q3. How does the distribution of color class sizes influence downstream performance?
Q4. Would alternative coloring types incur large penalties in the downstream subroutines?

4.2 Algorithmic Landscape for Classes of Bounded Expansion

A significant portion of the algorithmic research for classes of bounded expansion focuses around a meta-theorem about decidability of first-order (FO) logic on graphs.

Proposition 9 ([32]). Let \( \Phi \) be a first order formula and \( C \) a class of graphs with bounded expansion. There exists a linear-time algorithm for deciding \( \Phi \) for any graph \( G \in C \).

FO logic allows quantification over vertices in the graph, but not over sets of vertices. Some examples of graph properties that can be expressed in FO logic are the existences of fixed-size vertex covers, independent sets, dominating sets, or subgraph isomorphisms. Though Proposition 9 gives a linear time algorithm to solve a class of common problems, it hides a large constant factor in the big-O notation, which renders the theorem unusable in practical applications.

A few other algorithms have been developed for specific problems on bounded expansion classes that do not rely on the “heavy machinery” of meta-theorems. Dawar and Kreutzer
introduced an algorithm that decides the distance-$d$ dominating set problem in linear time [25]. This problem asks whether there is a vertex set $S$ of size $k$ such that the minimum distance from each vertex $v$ to a member of $S$ is at most $d$.\footnote{For $d = 1$, this is the traditional dominating set problem.} Dvořák et al. have shown that if the expansion function $f$ of class $C$ satisfies $\log f(x) \in o(x)$, then every graph in $C$ has a sublinear separator which can be found in $o(n^2)$ time [31]. Multiple algorithms [27, 74] exist for counting the number of subgraph isomorphisms in linear time. Finally, localized centrality measures can also be computed in linear time [27].

### 4.3 Subgraph Isomorphism Counting

We have chosen to study algorithms for counting subgraph isomorphisms due to previous research suggesting that isomorphism counting is a powerful tool in network science.

**Definition 22.** Graph $H$ is isomorphic to a subgraph of graph $G$ if there exists a mapping $\psi : V(H) \to V(G)$ such that $uv \in E(H)$ if and only if $\psi(u)\psi(v) \in E(G)$.

The subgraph isomorphism counting problem is to count the number of distinct mappings from $V(H)$ to $V(G)$; solving this problem is #W[1]-hard\footnote{This means there is unlikely to be a FPT subgraph isomorphism counting algorithm parameterized by the size of $H$.} [50].

Each isomorphism from $H$ into a subgraph of $G$ must preserve both edges and non-edges, meaning that the image of $H$ is an induced subgraph of $G$. However, subgraph isomorphism counting is not equivalent to counting the number of occurrences of $H$ as an induced subgraph of $G$. If $H$ has multiple automorphisms, isomorphisms from $H$ to itself, each distinct induced subgraph of $G$ will be counted multiple times, as shown in Figure 4.1. This can be remedied by dividing the number of isomorphisms by the number of automorphisms of $H$, the latter of which can be counted in time $O(2^{\sqrt{k}\log k})$ where $k = |V(H)|$ [63].

Subgraph isomorphism counting is important as a building block for additional graph analysis. In the next section we give examples of two such analytical techniques.

#### 4.3.1 Motif Counting

Originally used in the study of cellular biology [92], motif counting involves comparing the number of isomorphic subgraphs in a graph of interest to what one might expect to occur “by chance”. First, a random graph model is used to construct a graph $G_R$ that matches $G$ in its size and other parameters allowed by the model e.g. degree distribution, clustering coefficient, etc. Then any subgraph isomorphism algorithm can be used to count the number of occurrences of $H$ as a subgraph in $G_R$ and $G$. Usually $H$ represents some functional unit
in the domain of the data; for example, a bifan in a regulatory network representing logical controls for dependent processes [60]. As such, a relative abundance of $H$ in $G$ compared to $G_R$ suggests that specific mechanisms that drive interactions between data instances inherently cause these patterns to occur.

4.3.2 Graphlet Degree Distribution

Subgraph isomorphism is also a building block in the graphlet degree distribution [82]. In the standard degree distribution, we compute the number of vertices incident to exactly one edge, then two edges, and so on. The graphlet degree distribution generalizes this by looking at the number of times a vertex occurs in an automorphism orbit of a particular small subgraph. The automorphism orbits of a graph are the labels corresponding to labeling the vertices of a graph such that any two vertices that could be mapped to one another in an automorphism receive the same label. For example, the two endpoints of a $P_3$ occupy the same orbit, which is distinct from the orbit of the central vertex. The authors [82] suggest measuring the distribution of vertex occupation of all 73 orbits of graphs with 5 or fewer vertices. This, of course, requires finding isomorphisms for each $H$ such that $|H| \leq 5$. Graphlet degree distributions give a more robust “fingerprint” of a graph’s structure which has in turn been used to uncover underlying domain knowledge [66]. They also have been used in network alignment [65], using the knowledge that two vertices are more likely to be mapped to one another if they occupy the orbits at similar frequencies.

4.3.3 Existing Algorithms

Below is a survey of existing algorithms for subgraph isomorphism counting. We note that none of them have better worst-case time complexity than the brute force $O(|G|^{|H|})$ algorithm.
4.3.3.1 Ullmann

One of the first papers written about the subgraph isomorphism problem was by Ullmann in 1976 [100]. To test whether two graphs \( G' \subseteq G \) and \( H \) are isomorphic, this algorithm uses a \( |V(G) \times V(H)| \) matrix that tracks whether a particular mapping from \( H \) to \( G \) is possible. Originally, the matrix only evaluates the “possibility” of an isomorphism by checking whether each vertex of \( H \) maps to a vertex in \( G \) with larger degree, but then uses matrix operations to expand this check to match two vertices’ neighborhoods. Ullmann’s algorithm employs recursive backtracking to narrow down the space of candidate mappings without enumerating all possibilities. This algorithm can be easily modified to find non-induced subgraphs; however, counting the number of isomorphic subgraphs could require traversing a much larger search tree since we cannot simply stop once one isomorphism has been found.

4.3.3.2 VF2

The VF [20] and VF2 [21] algorithms utilize a different approach. They iteratively expand partial mappings of a subgraph of \( G \) to \( H \) until a full match is found. To trim the search space, these algorithms ignore partial mappings that lead to “dead ends”. That is to say, before adding a vertex to the partial mapping, we make sure that the new matching would conform to a set of feasibility rules. The feasibility rules check all neighbor relationships are preserved in the partial matching, and that each vertex from \( G \) in the partial matching has sufficiently many neighbors outside the mapping to complete a final isomorphism. Like Ullmann’s algorithm, it is possible to count subgraphs as well as finding non-induced subgraphs. The VF2 algorithm was tested in [21] and found to be orders of magnitude faster than Ullmann’s algorithm. It is currently implemented in many graph software packages such as NetworkX [42], igraph [22], and Boost [93].

4.3.3.3 QuickSI/Swift-Index

Some algorithms focus on finding isomorphisms in an indexed graph database. One paradigm for such algorithms is filter and verify, which essentially selects candidate subgraphs and then checks in greater detail whether the subgraph is isomorphic [91]. The QuickSI algorithm describes \( H \) using a spanning tree and an auxiliary data structure for the additional edges not found in the spanning tree. Computing such data structures for the subgraphs of \( G \) allows for easy comparison to determining the existence of an isomorphism. The Swift-Index part of the algorithm is used for the filtering portion and uses feature-based methods to determine which graph indices of \( G \) could be part of a spanning tree that matches that of \( H \). The indices of the graph are constructed as to allow fast execution of the filter algorithms. Since the algorithm
depends on spanning trees to sketch a subgraph, this algorithm does not give an easy way to search for non-induced subgraphs.

4.3.3.4 GraphQL

Another isomorphism matcher in a graph database is the GraphQL [43]. It defines a query language akin to SQL that lets users search for vertices based on specified edges and vertex labels. Like SQL, this query language allows multiple records to be returned, which means isomorphism counting can be done. The underlying portion of the matching algorithm is based on growing the graph using a simple DFS of candidates that could match the next vertex in the query. The “meat” of the algorithm is in the way it trims the search space. One such trick is through a bipartite matching algorithm. The image of vertex \( u \in H, u' \), must have a subtree in the DFS compatible with that of the DFS subtree of \( u \). Thus, they construct a bipartite graph between the unprocessed vertices of \( H \) and their candidate images and look for a bipartite matching that saturates the side corresponding to the vertices of \( H \). If no such matching exists, the search space can be trimmed. It also prioritizes “more likely” candidates in \( G \) with a cost function derived from the way the joins were written in a particular query.

4.3.3.5 STwig

A final approach looks to scale to large graphs online without indexing on a parallel architecture. The STwig algorithm [97] is called such because it decomposes \( H \) into a set of overlapping two-level trees called STwigs that in total encompass all the edges and vertices of \( H \). The individual STwigs can be found in \( G \) in parallel or serially and then joined together to make an entire graph. The STwigs are intermediately labeled in order to ensure that the join process only pieces together compatible STwig matches. The algorithm incurs some initial overhead to compute a better decomposition of \( H \) into STwigs. This in turn leads to better load balancing and faster joins. However, the STwig decomposition does not allow for non-induced subgraphs to be found.

Based on its versatility, empirical performance, and independence from specific architectures, we choose to focus on the VF2 algorithm later in this chapter.

4.4 CONCUSS

Broadly speaking, the algorithmic pipeline in CONCUSS executes four modules, illustrated in Figure 4.2, to count the number of isomorphisms of graph \( H \) in \( G \).

1. **Color**: Find a \((|H| + 1)\)-centered coloring of \( G \).
2. **DECOMPOSE**: Compute a treedepth decomposition of the subgraph induced by each set of $|H|$ colors.

3. **COMPUTE**: Using dynamic programming, count the isomorphisms in the treedepth decompositions.

4. **COMBINE**: Combine the counts from **COMPUTE** to get the total number of isomorphisms in $G$.

We briefly describe these modules below. Many of the subroutines in the modules can achieve the same high-level functionality with different implementation details, e.g., heuristics, data structures, etc. **CONCUSS** allows users to specify their choices for some of these subroutines in a configuration file; for more details, see Appendix A. These alternatives are designated with teletype.

---

**Figure 4.2** Algorithmic pipeline in **CONCUSS**.
4.4.1 Color

The workflow in the Color module is outlined in Algorithm 6. The objective is to produce a \( p \)-centered coloring by greedily coloring the vertices of the graph. Because a simple greedy coloring may not immediately be \( p \)-centered, additional edges are added to the graph to impose further constraints. After a bounded number of iterations of the loop in Algorithm 6, a greedy coloring is guaranteed to be \( p \)-centered [71, 83]. Since each substep runs in linear time with respect to \(|G|\), the entire Color module also runs in linear time.

**Algorithm 6** \( p \)-cc\((G)\) (sketch)

**Input:** Graph \( G \), integer \( p \)

**Output:** \( p \)-centered coloring of \( G \)

1. \( G' \leftarrow G \)
2. repeat
   3. Orient the edges of \( G' \) to make it a directed acyclic graph of low indegree
   4. Augment \( G' \) with additional edges based on the orientation
   5. Color the vertices greedily
   6. Check whether each set of vertices with \( p - 1 \) colors has low treedepth in \( G \)
3. until the coloring is \( p \)-centered

**Orient:** The later stages of the Color module operate on directed acyclic graphs. In essence, directed edges capture the ancestor relationships in treedepth decompositions of small sets of colors by having edges point from an ancestor to its descendants. In this representation, vertices with large indegree have many ancestors, indicating subgraphs with large treedepth. A low-indegree orientation can be found either with a Degeneracy ordering [64] (Algorithm 3) or by Sandpiling [5].

**Augment:** After the edges are oriented, additional directed edges are augmented to the graph to impose more constraints. These are exactly the transitive-fraternal augmentations (TFA) and distance-truncated transitive fraternal augmentations (DTFA) described in Section 2.2.3.

**Color:** The acyclic orientation of the edges defines an ordering over which to greedily color the vertices. To minimize the number of colors, CONCUSS uses one of three different heuristics: prioritizing high-degree vertices (High-degree), prioritizing low-degree vertices (Low-degree), or prioritizing vertices with many colors already represented among their neighbors (DSATUR) [11].

**Check:** Finally, we check whether the coloring is \( p \)-centered by testing whether each connected subgraph that uses \( p - 1 \) or fewer colors has a unique color. This can be done by keeping track of components and their respective color multiplicities using union-find...
structures; when two components merge, we ensure that there is still a unique color. We perform this check at every iteration in order to prevent the algorithm from adding any more colors than necessary.

**Other Optimizations:** We include additional pre- and post-processing steps to obtain smaller colorings. Vertices of high degree can cause many additional edges to be augmented at each iteration of the loop, which in turn causes more colors to be used. For this reason, removing all high-degree vertices, e.g., degree at least $\sqrt{n}$, finding a coloring of the smaller graph, and then giving each removed vertex a new unique color can reduce the total number of colors. Likewise, in any component of size larger than two, vertices of degree one can be removed and all given the same color in post-processing. After obtaining a $p$-centered coloring, we can also potentially reduce the number of colors by randomly adding transitive and fraternal edges, computing a new greedy coloring, and seeing if this greedy coloring is $p$-centered. The final post-processing step is to check whether pairs of color classes can be merged without violating the $p$-centered property.

### 4.4.2 Decompose

After the coloring is computed, CONCUSS finds the components induced by each set of $|H|$ colors. To save computation, CONCUSS processes overlapping color sets sequentially in a depth-first search manner. For example, if $|H| = 4$ and 9 colors, the sets would be processed as such: \{1\}, \{1, 2\}, \{1, 2, 3\}, \{1, 2, 3, 4\}, \{1, 2, 3, 5\}, \ldots, \{1, 2, 3, 9\}, \{1, 2, 4\}, \{1, 2, 4, 5\}, etc.

Given a component induced by a particular color set, a treedepth decomposition is created using Algorithm 2. In order to save memory, these treedepth decompositions are fed into the subsequent **Count** and **Compute** stages before a new color set is processed.

### 4.4.3 Compute

The **Compute** module uses the dynamic programming algorithm of Demaine et al. [27] to count the isomorphisms of $H$ in a treedepth decomposition. It has exponential time dependence on the small treedepth, but runs in linear time with respect to $|G|$. Briefly, this algorithm exploits the fact that no edges in the treedepth decomposition join vertices in different branches. Therefore, partial pieces of $H$ in one subtree have limited ways of interacting with those in other subtrees. By tracking which pieces occur in each subtree, these counts of partial pieces can be combined to count all isomorphisms in the decomposition.

Integral to the dynamic programming are $k$-patterns, also called boundaried graphs. These (sub)graphs have a subset of vertices labeled uniquely with integers on the interval $[1,k]$, which in turn determine how partial pieces of $H$ can “glue” together.
4.4.4 Combine

The goal in the COMBINE module is to sum the counts from each treedepth decomposition to get the count for the whole graph. However, because the color sets contain overlapping subgraphs, a simple summation of counts may count some isomorphisms more than once. For example, if \( H = P_4 \), the subgraph in Figure 4.3 will be counted in the color sets \{blue, orange, purple, yellow\}, \{blue, orange, purple, green\}, etc. There are multiple ways to rectify this issue.

Figure 4.3 The highlighted \( P_4 \) only uses three colors and thus will be counted in multiple color sets of size four.

Inclusion-exclusion simply enumerates all color sets with fewer than \( |H| \) colors, counts the isomorphisms in those subgraphs, and then applies the inclusion-exclusion principle to correct the counts appropriately. We can also incorporate the combinations into the dynamic programming itself by ignoring counts of color sets that have already been seen (Hybrid).

4.4.5 Extensions to other problems

We note that the four-stage algorithmic workflow (COLOR, DECOMPOSE, COMPUTE, COMBINE) can be useful for solving problems other than subgraph isomorphism counting. In particular, this approach is amenable to problems in which local subsolutions can be efficiently computed on graphs of low treedepth and then combined to create a global solution. For such a problem, it is possible to reuse the COLOR and DECOMPOSE modules and simply replace the COMPUTE and COMBINE modules with the appropriate alternatives.

4.5 Experimental Design

We conducted four experiments, one to answer each question posed in Section 4.1. The first two (Section 4.6) identify an “optimal” configuration for CONCUSS and test its scalability against an
existing subgraph isomorphism counting algorithm. The other two experiments (Section 4.7) investigate ways in which properties of the coloring affect downstream computation to guide future theoretical research.

To compare the effects of varying multiple options, we normalize two measures $a, b$ using the difference to sum ratio, i.e., $\frac{a - b}{a + b}$. This ratio will be close to 1 when $a \gg b$ and close to $-1$ when $b \gg a$. When comparing times, a positive ratio indicates $b$ is faster than $a$.

### 4.5.1 Data

Since the theoretical advantages of the algorithms implemented in CONCUSS rely on exploiting structure, it was necessary to ensure our input graphs belonged to classes of bounded expansion. Consequently, we chose three random graph models described in Section 2.3 which asymptotically almost surely produce classes of bounded expansion [27]: the stochastic block model (SB) [44] and the Chung-Lu model [19], with households (CLH) and without (CL). The first two models both exhibit clustering commonly found in real-world networks; the CL model was included as a baseline. For each model, we selected configuration parameters known to produce bounded expansion classes. In the CL and CLH models, we used a degree distribution with exponential decay and household size four; for SB we used

\[
\begin{bmatrix}
.40 & .30 & .20 & .10 \\
- & .50 & .13 & .05 \\
- & - & .35 & .11 \\
- & - & - & .45
\end{bmatrix}
\]

as the probability matrix. For each model, three random instances with 1024 vertices and average degree 6 were generated.

### 4.5.2 Hardware

The experiments were run on identical machines with four-core, 3.0 GHz Intel Xeon E5 v3 processors with a 10 MB cache and 64 GB of memory ($4 \times 16$ GB). Each run of CONCUSS received dedicated resources to avoid interference from other processes.

### 4.6 Competitive Evaluation

We evaluated the practicality of CONCUSS by first testing different configurations against one another before comparing them to an existing implementation of a different subgraph isomorphism counting algorithm.
4.6.1 Configuration Testing

To measure the run times of various configurations, we created one configuration file for each combination of options described in Sections 4.4.1 and 4.4.4. For all possible pairings of random graph instances and configuration options we ran CONCUSS three times. In each run we counted the number of isomorphisms of $P_4$; the particular motif did not vary because the dynamic programming algorithm has theoretically comparable run time for all motifs of the same size. The metrics of interest in each run were the number of colors used, the total elapsed time, and the time specifically spent in the Color module. To compare the performance between two implementations of a subroutine, we did a pairwise comparison between configurations in which all other subroutines were held constant. That is to say, to differentiate between $A_1$ and $A_2$, we would compare $(A_1, B_1, C_1)$ against $(A_2, B_1, C_1)$, $(A_1, B_2, C_1)$ against $(A_2, B_2, C_1)$, etc.

![Figure 4.4](image.png)

**Figure 4.4** Average difference/sum ratio in coloring size (top), Color time (middle), and Decompose, Compute, and Combine time (bottom) between paired configuration options.

Though there were minor variations across graph models, the effects of the different configuration options (Figure 4.4) were relatively consistent. Orienting the edges using Sandpiling
and Degeneracy led to comparable time and coloring sizes. Surprisingly, using DTFA, which is more prudent in augmenting edges, did not outperform TFA in either time or coloring size. We attribute this result to the fact that the average number of augmentation steps needed to reach a $p$-centered coloring was smaller while using TFA (2.9 vs. 4.8), indicating that it may be worth making “suboptimal” choices for the augmentations as long as it leads to a $p$-centered coloring sooner. The Low-degree coloring prioritization generally yielded fewer colors and a shorter run time than High-degree, while the more complicated DSATUR prioritization was somewhere in-between. Taken as a whole, the best coloring routine used Degeneracy orientation, TFA, and Low-degree prioritization.

The general distribution of time spent within the Color module also did not vary much with the coloring configuration (Figure 4.5). Computing the augmentations and coloring the vertices took a very small fraction of the total run time; upwards of 80% of the time was spent ensuring merging color classes kept the coloring $p$-centered, with the pre-merge
optimization consuming most of the rest of the time. Nearly all of the computation in these two post-processing routines is in deciding whether a smaller coloring is indeed \( p \)-centered. As such, small improvements to the efficiency of checking the treedepth of all color sets would pay large dividends in the overall time.

Although the post-processing reduced the coloring size by at least 50% in 90% of the colorings and the total execution time correlated positively with coloring size (Figure 4.6), we cannot assess whether those optimizations were a net benefit without extrapolating beyond our observed data. Because the time spent merging is dependent on the coloring size, we believe there is a coloring size threshold below which the post-processing is worthwhile; it would be useful to empirically identify this threshold in future work.

The effects of the configurations on the computation downstream from the coloring (Figure 4.4) were weaker and more varied. In particular, we note that Low-degree prioritization often led to slower downstream computation, but this was offset by the faster coloring times.

The only configuration option varied outside of the Color module was the method of correcting double counting in the Combine module. We were once again surprised to see that Hybrid, the “intelligent” method of circumventing enumerating many additional color classes, was not successful at improving the run time over the naïve Inclusion-exclusion. To the contrary, Hybrid increased the total time of the Decompose, Compute, and Combine modules by at least a factor of two, getting up to a factor of ten when the number of colors was very large. We ultimately concluded the preferred configuration is Degeneracy, TFA, Low-degree, and Inclusion-exclusion.

### 4.6.2 Comparison with NXVF2

After determining the behavior of various configurations of CONCUSS, we wanted to assess its performance compared to the NetworkX [42] implementation of the popular VF2 algorithm [21], which we will refer to as NXVF2.

#### 4.6.2.1 Additional Data

Because of the large \( p \)-centered coloring sizes of the Chung-Lu and stochastic block model graphs generated for the previous experiment, CONCUSS requires around 45 minutes to count in graphs where NXVF2 terminates in 1 second. We hypothesized, however, that in large graphs with small coloring sizes and many isomorphisms of \( H \) into \( G \), CONCUSS would outperform NXVF2. We included the restriction on number of isomorphisms because the run time of CONCUSS does not depend on the count, while NXVF2 requires more time for each additional isomorphism. To test this hypothesis we generated graphs that met the criteria with the following procedure. First, we generated a complete binary tree of depth \( d \).
we selected a tree vertex uniformly at random and attached to it the endpoint of a $P_\ell$. We continued randomly selecting tree vertices with replacement and adding a $P_\ell$ for a total of $s \cdot 2^d$ additions. That is, we made sure that each tree vertex had an average of $s$ $P_\ell$s as neighbors. We denote this graph $T_{d,s,\ell}$ (Figure 4.7). In this way, we could effectively vary the size of the graph and the number of isomorphisms of small stars and paths.

![Figure 4.7 $T_{d,s,\ell}$](image)

### 4.6.2.2 Results

We observed (Figure 4.8) that our hypothesis holds when counting stars in $T_{d,s,\ell}$ for sufficiently large $s$. In particular, CONCUSS has more than a fourteen-fold speed advantage over NXVF2 in counting $S_5$s in $T_{12,16,1}$. We believe the difference in time would diverge even more for counting $S_5$s in $T_{14,16,1}$, but NXVF2 already takes nearly 80 hours in $T_{12,16,1}$.

It is also important to note that when $\ell = 1$, the number of paths does not increase in the same way that the number of stars increases. This is because every path of length at least four must contain multiple tree vertices, of which each tree vertex is adjacent to at most three. Moreover, $P_n$ has exactly two automorphisms (“forwards” and “backwards”) for every $n$, while $S_n$ has $(n - 1)!$ automorphisms (any ordering of the leaves). We observed that there were insufficiently many $P_4$s and $P_5$s in $T_{d,s,1}$ to see the same performance benefits, and NXVF2 was consistently faster than CONCUSS.

As predicted, CONCUSS can outperform NXVF2 when counting in $T_{d,s,4}$, which has many more $P_4$s and $P_5$s (Figure 4.9). Similar to the results shown in Figure 4.8, CONCUSS counts $P_3$s faster in all graph sizes, while the advantage in counting $P_5$s is only realized in the largest graphs.
4.7 Bottleneck Identification

Having identified configurations and graph instances in which CONCUSS outperforms NXVF2, we proceeded to identify bottlenecks in CONCUSS that ought to receive further theoretical consideration. We focused on how changing properties of the output of the Color module would influence performance in the other three modules since the different coloring subroutines had very different run times.

4.7.1 Color Class Distribution

Because a graph may admit many distinct \( p \)-centered colorings of small size, it is important to know whether two colorings of the same size lead to observable differences in the downstream computation. Though the effects the coloring has on the rest of the pipeline almost certainly depend on complex and subtle interactions, we hypothesized that the distribution of sizes
Figure 4.9 Average difference/sum ratio between CONCUSS and NXVF2 on $T_{d,s,4}$ as a function of $s$, the average number of $P_4$s per tree vertex. Each small plot shows a fixed motif and value of $d$. Negative ratios indicate CONCUSS is faster.

of color classes is an important factor. The intuition behind this hypothesis is that different distributions in sizes will lead to different “shapes” of treedepth decompositions, i.e., one large, wide decomposition vs. many small, thin decompositions. To test it, we created colorings of equal size that have unequal distributions of color class sizes.

The stochastic block and Chung-Lu with households graphs described previously were used again as inputs for this experiment; we excluded the Chung-Lu graphs without households due to their long run times. For each graph instance, we found the largest coloring from the configuration experiment, then iteratively split in half the existing color classes (Figure 4.10) in colorings generated by the other configurations until all colorings had the same size. To vary the distributions of color class sizes, we used three splitting heuristics: max, med, and min, which target the maximum-, median-, and minimum-sized color classes. When splitting color classes, we chose a random subset of vertices to move into the new color class; to account for

3Color classes with exactly one vertex were ignored for calculating the minimum and median sizes.
Figure 4.10 Creating a new color class (black) using each of the three splitting heuristics (min, med, max).

Table 4.1 Average run time and dynamic programming operations used for each color splitting heuristic. The statistics for med and min are reported as percent increases over max.

<table>
<thead>
<tr>
<th>Model</th>
<th>Method</th>
<th>Avg time (s)</th>
<th>Joins</th>
<th>Forgets</th>
</tr>
</thead>
<tbody>
<tr>
<td>SB</td>
<td>max</td>
<td>856</td>
<td>$5.12 \times 10^5$</td>
<td>$2.00 \times 10^8$</td>
</tr>
<tr>
<td>SB</td>
<td>med</td>
<td>+19%</td>
<td>+48%</td>
<td>+13%</td>
</tr>
<tr>
<td>SB</td>
<td>min</td>
<td>+20%</td>
<td>+49%</td>
<td>+13%</td>
</tr>
<tr>
<td>CLH</td>
<td>max</td>
<td>1327</td>
<td>$7.63 \times 10^5$</td>
<td>$3.01 \times 10^8$</td>
</tr>
<tr>
<td>CLH</td>
<td>med</td>
<td>+21%</td>
<td>+48%</td>
<td>+4%</td>
</tr>
<tr>
<td>CLH</td>
<td>min</td>
<td>+21%</td>
<td>+50%</td>
<td>+5%</td>
</tr>
</tbody>
</table>

This randomness, we repeated each splitting heuristic three times for each coloring. We ran the Decompose, Compute, and Combine modules of CONCUSS three times for each graph instance and associated new coloring, using Inclusion-Exclusion in the Combine module.

We observed (Table 4.1) that max led to the fastest performance, while med and min had very similar run times. The dynamic programming algorithm counts isomorphisms from the leaves of the treedepth decomposition upwards using two operations: forgetting, which moves from children to their parent, and joining which combines results from “siblings” (vertices with the same parent). Since siblings often belong to the same color class, ensuring that no color class has too many vertices limits the number of children, which in turn limits the number of joins. In Table 4.1, we report that when using max the average number of joins was 33% lower than when splitting the median or minimum-sized classes. This experiment revealed that future coloring algorithms should attempt to make the sizes of the color classes more uniform.
4.7.2 Color Set Treedepth

The dynamic programming (DP) in the Compute module counts isomorphisms of a subgraph of size $h$ in an arbitrary treedepth decomposition of depth $t$. While this algorithm runs in linear time with respect to the size of the graph, it takes $O(t^h)$ time with respect to $t$. One potential direction for further theoretical improvements to the bounded expansion pipeline is to find alternatives to $p$-centered colorings in which subgraphs with at most $i < p$ colors have treedepth that is small but greater than $i$. Ideally, relaxing treedepth requirement would result in a significant reduction in the size of the coloring without dramatically increasing the time spent in DP in the Compute module. Though we cannot measure the size of a hypothetical coloring, we can evaluate how the time per DP operation increases as we count the same subgraph in larger treedepth decompositions. To do this, we computed a 6-centered coloring, enumerated sets of 3, 4, and 5 colors, and counted the number of isomorphisms of $P_3$ in each set.

The run time of the DP algorithm is only dependent on the treedepth of the decompositions insofar as it increases the number of labelings $^4$ of the ancestors of each vertex. There are $t^3 + 3t^2 + t$ such labelings for a $P_3$, each of which requires a constant number of DP operations, so the amount of time spent counting in one color set should grow proportionally to this function.

To compare our observations to the predicted growth of $t^3 + 3t^2 + t$, we assumed that the average time per operation for $t = 3$ aligned correctly with this theoretical time and extrapolated outwards for $t \in \{4, 5\}$. As reported in Figure 4.11, the observed times grow significantly slower than predicted for all operations. The time spent on each operation is dependent on the depth of the vertex in the treedepth decomposition, but the average depth of vertices does not grow at a

---

$^4$These are the $k$-patterns mentioned in Section 4.4.3; see [27].
Table 4.2 Average execution time per operation and average depth of vertices in the treedepth decomposition. Rows for treedepth $t$ only include those operations incurred in counting in color sets of exactly $t$ colors.

<table>
<thead>
<tr>
<th>Model</th>
<th>$t$</th>
<th>Time per leaf</th>
<th>Time per join</th>
<th>Time per forget</th>
<th>Avg. depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL</td>
<td>3</td>
<td>$7.5 \times 10^{-7}$</td>
<td>$7.3 \times 10^{-7}$</td>
<td>$1.4 \times 10^{-6}$</td>
<td>2.0</td>
</tr>
<tr>
<td>CL</td>
<td>4</td>
<td>$7.7 \times 10^{-7}$</td>
<td>$8.1 \times 10^{-7}$</td>
<td>$1.7 \times 10^{-6}$</td>
<td>2.4</td>
</tr>
<tr>
<td>CL</td>
<td>5</td>
<td>$7.9 \times 10^{-7}$</td>
<td>$9.0 \times 10^{-7}$</td>
<td>$2.0 \times 10^{-6}$</td>
<td>2.8</td>
</tr>
<tr>
<td>CLH</td>
<td>3</td>
<td>$8.0 \times 10^{-7}$</td>
<td>$7.4 \times 10^{-7}$</td>
<td>$1.4 \times 10^{-6}$</td>
<td>2.0</td>
</tr>
<tr>
<td>CLH</td>
<td>4</td>
<td>$8.0 \times 10^{-7}$</td>
<td>$8.1 \times 10^{-7}$</td>
<td>$1.7 \times 10^{-6}$</td>
<td>2.4</td>
</tr>
<tr>
<td>CLH</td>
<td>5</td>
<td>$8.1 \times 10^{-7}$</td>
<td>$9.3 \times 10^{-7}$</td>
<td>$2.1 \times 10^{-6}$</td>
<td>2.8</td>
</tr>
<tr>
<td>SB</td>
<td>3</td>
<td>$8.1 \times 10^{-7}$</td>
<td>$7.3 \times 10^{-7}$</td>
<td>$1.4 \times 10^{-6}$</td>
<td>1.9</td>
</tr>
<tr>
<td>SB</td>
<td>4</td>
<td>$7.9 \times 10^{-7}$</td>
<td>$7.9 \times 10^{-7}$</td>
<td>$1.5 \times 10^{-6}$</td>
<td>2.3</td>
</tr>
<tr>
<td>SB</td>
<td>5</td>
<td>$8.0 \times 10^{-7}$</td>
<td>$8.5 \times 10^{-7}$</td>
<td>$1.8 \times 10^{-6}$</td>
<td>2.7</td>
</tr>
</tbody>
</table>

one-to-one ratio with the treedepth (Table 4.2). This means that the additional color classes add vertices at many depths of the decomposition, rather that simply increasing the depth of every branch uniformly, and thus we seldom “pay” for the cost of the larger depths. Consequently, we conclude that colorings producing deeper decompositions with fewer colors ought to be a viable strategy for improving the overall speed of any implementation based on this pipeline.

### 4.8 Conclusion

We demonstrated that exploiting bounded expansion structure is a promising methodology for scaling subgraph isomorphism counting to larger classes of sparse graphs. We identified through testing various configurations of CONCUSS that orienting with a degeneracy ordering, augmenting with TFA, coloring low-degree vertices first, and combining counts using the inclusion-exclusion principle led to fast run times and small colorings. On sufficiently sparse graphs with high isomorphism counts, CONCUSS was able to significantly outperform NXVF2.

In order to get the bounded expansion pipeline to outperform other methods in a broader set of graphs, it is important to understand in future work whether the large coloring sizes of the random graph models were an artifact of the heuristics used in generating the $p$-centered colorings, or whether their asymptotically bounded values are actually large.

Our experiments also identified three areas for future theoretical research. The first is to reduce the time needed to verify a coloring is $p$-centered, which would yield large dividends in the Color module. The second is to design $p$-centered coloring algorithms that balance the sizes of the color classes, consequently reducing the number of join operations. Finally, our results suggest using colorings that trade fewer colors for larger treedepth per color set could be faster than using $p$-centered colorings.
Summary

Low-treedepth colorings are an important tool for algorithms that exploit structure in classes of bounded expansion; they guarantee subgraphs that use few colors are guaranteed to have bounded treedepth. These colorings have an implicit tradeoff between the total number of colors used and the treedepth bound, and prior empirical work suggests that the former dominates the run time of existing algorithms in practice. We answer Research Question 3 by introducing \textit{p-linear colorings} as an alternative to the commonly used \textit{p-centered colorings}. They can be efficiently computed in bounded expansion classes and use at most as many colors as \textit{p-centered colorings}. Although a set of \( k < p \) colors from a \textit{p-centered} coloring induces a subgraph of treedepth at most \( k \), the same number of colors from a \textit{p-linear} coloring may induce subgraphs of larger treedepth. A simple analysis of this treedepth bound shows it cannot exceed \( 2^k \), but no graph class is known to have treedepth more than \( 2k \). We establish polynomial upper bounds via constructive coloring algorithms in trees and interval graphs, and conjecture that a polynomial relationship is in fact the worst case in general graphs. We also give a co-NP-completeness reduction for recognizing \textit{p-linear} colorings and discuss ways to overcome this limitation in practice. The results in this chapter are under submission to the 44th International Workshop on Graph-Theoretic Concepts in Computer Science [53].
5.1 Introduction

Although many algorithms for classes of bounded expansion—including those implemented in CONCUSS—traditionally rely on \( p \)-centered colorings, they are generally compatible with any low-treedepth coloring, i.e., a vertex coloring in which every small set of colors induces a subgraph of bounded treedepth. The experimental results in the previous chapter suggest that replacing \( p \)-centered colorings with a low-treedepth coloring that uses fewer colors but has weaker guarantees on the treedepth of subgraphs induced on small color sets may lead to a faster run time without significant high-level modifications to the pipeline. To answer Research Question 3, we thus choose to focus on developing such a coloring.

In this chapter, we introduce \( p \)-linear and linear colorings as analogs to \( p \)-centered and centered colorings, respectively. The primary difference between these colorings is that (\( p \)-)centered colorings guarantee unique colors in subgraphs, while (\( p \)-)linear colorings only extend that guarantee to paths. We identify that \( p \)-linear colorings share three important properties with \( p \)-centered colorings that allow them to be used in the bounded expansion algorithmic pipeline.

1. The minimum coloring size is constant in graphs of bounded expansion.
2. A coloring of bounded size can be computed in polynomial time.
3. Small sets of colors induce graphs of small treedepth.

The third of these properties is of particular interest, since understanding the tradeoffs between coloring size and treedepth in switching between \( p \)-centered and \( p \)-linear colorings fundamentally depends on bounding the maximum treedepth of a graph that admits a linear coloring with \( k \) colors. Equivalently, we frame this problem as determining the gap between the minimum number of colors needed for a linear versus a centered coloring in any given graph. A naïve analysis does not exclude the possibility that this gap is exponentially large, despite the fact that the largest known difference is a constant multiplicative factor. We conjecture that our proven constant factor lower bound is also the upper bound; as evidence, we prove that in trees and interval graphs the difference is polynomially bounded (in the coloring size) and give polynomial time algorithms (with respect to the graph size) to certify this difference. Surprisingly, we also prove that some \( p \)-linear colorings cannot be verified in polynomial time unless \( P = \text{co-NP} \) and discuss the practical implications of these findings.

5.2 \( p \)-Linear and Linear Colorings

We introduce \( p \)-linear colorings as an alternative to \( p \)-centered colorings.
**Definition 23.** A \(p\)-linear coloring is a coloring \(\psi\) of a graph \(G\) such that for every path\(^1\) \(P\), either \(P\) has a center or \(\psi|_P\) uses at least \(p\) colors.

It is proven in [83] that after performing \(2^p\) DTFA iterations, any proper coloring of the augmented graph is a \(p\)-linear coloring. This implies that \(p\)-linear colorings indeed have constant size in bounded expansion and can be constructed in polynomial time (like \(p\)-centered colorings).

In the interest of maintaining consistency with prior terminology, we define linear colorings analogously to centered colorings.

**Definition 24.** A linear coloring is a coloring of a graph \(G\) such that every path has a center. The linear coloring number is the minimum size of a linear coloring and is denoted \(\chi_{\text{lin}}(G)\).

Note that linear colorings must also be proper. A simple recursive argument indicates that every path of length \(d\) requires at least \(\log_2(d+1)\) colors in a linear coloring; thus a graph of linear coloring number \(k\) has no path of length \(2^k\). Because every depth-first search tree is a treedepth decomposition, \(\text{td}(G) \leq 2^{\chi_{\text{lin}}(G)}\), proving that small numbers of colors in a \(p\)-centered coloring induce graphs of bounded treedepth\(^2\).

Our study of the divergence between linear and centered coloring numbers will naturally focus on linear colorings that are not also centered colorings. We say \(\psi\) is a non-centered linear coloring (NCLC) of graph \(G\) if \(G\) contains a connected induced subgraph with no center. For NCLC \(\psi\), we say a connected induced subgraph \(H\) is a witness to \(\psi\) if \(H\) is non-centered but every proper connected subgraph of \(H\) has a center. For the sake of completeness, we prove in Lemma 6 that many simple graph classes do not admit NCLCs.

**Lemma 6.** If \(G\) is a path, star, cycle, complete graph, or complete bipartite graph, any linear coloring of \(G\) is also a centered coloring.

**Proof.** We analyze each graph class separately below.

**Path:** Definitions 3 and 24 are identical when \(G\) is a path.

**Star:** Every non-trivial star has centered coloring number 2. Since these graphs cannot be properly colored with a single color and linear colorings are proper, \(\chi_{\text{lin}}(G)\) must also be 2.

**Cycle:** Since every connected proper subgraph of a cycle is a path, the entire graph \(G\) must be the witness to any NCLC. But if \(G\) does not have a unique color, the subgraph formed by deleting any single edge would be a non-centered path. Thus every connected subgraph has a center, making any linear coloring centered.

---

\(^1\)This includes non-induced paths.

\(^2\)This tightens a bound in [83] from double to single exponential.
**Complete graph:** Linear and centered colorings are both proper, so they cannot differ in a complete graph.

**Complete bipartite graph:** Let \( \psi \) be an NCLC of \( G \) and \( H \) be a witness to \( \psi \). If \( \psi|_H \) only contains one color, \( H \) is an isolated vertex and the coloring is centered. Since every color in \( \psi|_H \) appears at least twice, there are vertices \( v, v', u, u' \) such that \( \psi(v) = \psi(v') \) and \( \psi(u) = \psi(u') \) but \( \psi(v) \neq \psi(u) \). Because \( H \) must also be a complete bipartite graph, \( v \) and \( v' \) must belong to the same bipartition and likewise for \( u \) and \( u' \). But then \( v, u, v'u' \) form a path with no center and thus \( \psi \) is not a linear coloring.

\[ \square \]

### 5.3 Lower Bounds

To understand the tradeoff between the number of colors and treedepth of small color sets when using \( p \)-linear colorings in lieu of \( p \)-centered colorings, it is important to know the maximum treedepth of a graph of fixed linear coloring number \( k \), \( t_{\text{max}}(k) \). In Lemmas 8 and 9, we prove lower bounds on \( t_{\text{max}}(k) \) through explicit constructions of graph families. In order to show that these graphs have large treedepth, we first establish assumptions about the structure of treedepth decompositions that can be made without loss of generality.

**Lemma 7.** Let \( G \) be a graph and \( S \subset V(G) \) such that \( G[S] \) is connected and with respect to some component \( C \in G \setminus S \), every vertex in \( S \) is an apex of \( C \). Then for any treedepth decomposition \( T \) of depth \( k \), we can construct a treedepth decomposition \( T' \) such that:

1. \( \text{depth}(T') \leq k \)
2. Each vertex in \( S \) is an ancestor of every vertex in \( C \) in \( T' \)
3. For each pair of vertices \( u, w \) such that \( u \in C \iff w \in C, u \) is an ancestor of \( w \) in \( T' \) iff it is an ancestor of \( w \) in \( T \).

**Proof.** Let \( \phi \) be a canonical centered coloring of \( G \) with respect to \( T \). Let \( T' \) be a canonical treedepth decomposition with respect to \( \phi \); if there are multiple vertices of unique color, prioritize removing those outside \( C \) before members of \( C \), and then small colors over large colors, i.e., remove color 2 before color 5. Since \( T' \) is derived from a centered coloring with \( k \) colors, its depth is at most \( k \), satisfying condition 1.

Condition 2 is satisfied as long each member of \( S \) is removed in the construction of \( T' \) before any member of \( C \). Note that since \( S \) contains apex vertices with respect to \( C \) and the only neighbors of \( C \) outside \( C \), the removal of any vertex from \( C \) cannot disconnect a previously connected component if \( S \) has not been removed. Thus at any point in the algorithm before
the removal of $S$, if a vertex in $C$ has a unique color in its remaining component $H$, there must be another vertex in $H \setminus C$ of unique color as well. Consequently, we will never be forced to remove any vertex of $C$ before $S$.

To prove condition 3 is satisfied, observe that $u$ is an ancestor of $w$ in $T'$ iff there is a connected subgraph $H$ containing $u$ and $w$ and no vertex with color smaller than $\psi(u)$. As stated previously, $G \setminus C$ is a connected subgraph, which means that there is a subgraph $H$ witnessing this ancestor-descendant relationship between $u$ and $w$ such that $H \cap C = \emptyset$ if $u \notin C$ and $H \cap (G \setminus C) = \emptyset$ if $u \in C$. Thus the relationships in $T$ are preserved in $T'$.

Using Lemma 7, we now show that $t_{\max}(k) \geq 2k$.

Lemma 8. There exists an infinite sequence of graphs $R_1, R_2, \ldots$ such that

$$\lim_{i \to \infty} \frac{\chi_{\text{cen}}(R_i)}{\chi_{\text{lin}}(R_i)} = 2.$$

Proof. Define $R_i$ recursively such that $R_1$ is a single isolated vertex and $R_i$ is a complete graph on vertices $v_1, \ldots, v_i$ along with $i$ copies of $R_p$ for $p = \lfloor \frac{i-1}{2} \rfloor$, call them $H_1, \ldots, H_i$, such that $v_j$ is an apex with respect to $H_j$. We prove that $\chi_{\text{lin}}(R_i) = i$ and $\lim_{i \to \infty} \chi_{\text{cen}}(R_i) = 2i$.

With respect to the linear coloring number, note that $\chi_{\text{lin}}(R_i) \geq i$ since the clique of size $i$ requires $i$ colors by Lemma 6. We prove the upper bound $\chi_{\text{lin}}(R_i) \leq i$ by induction on $i$. The case of $i = 1$ is trivial; assume it is true for $1, \ldots, i - 1$. From the inductive hypothesis, we can assume each $H_j$ only requires $p$ colors for a linear coloring. Consider the coloring $\psi$ of $R_i$ such that $\psi(v_j) = j$ and $\psi|_{H_j}$ is a linear coloring of $H_j$ using colors $\{(j + p) \mod i, (j + p + 1) \mod i, \ldots, (j + 2p - 1) \mod i\}$. If $\psi$ is not a linear coloring, there is some path $Q$ without a center. Since $\psi(v_j) \notin \psi|_{H_j}$, $Q$ must contain vertices from at least two $H_j$s; each $v_j$ is a cut vertex, so $Q$ cannot contain vertices from more than two $H_j$s. However, $\psi^{-1}(1) \subseteq \{v_1\} \cup V(H_{p+2}) \cup \ldots V(H_{2p})$, but $\{p+2, \ldots, 2p\} \notin \psi|_{H_1}$, which means $Q \cap H_1 = \emptyset$. Based on the symmetry of $\psi$ we can apply the same argument to the remaining colors, which means that no such non-centered path $Q$ exists and $\psi$ is indeed a linear coloring of size $i$.

With respect to the centered coloring number, by Lemma 7 there is an minimum-depth treedepth decomposition in which $v_j$ is an ancestor of $H_j$. This implies there is a $j$ such that no vertex in $H_j$ shares a color in the canonical coloring with any of the vertices in the clique. Thus $\chi_{\text{cen}}(R_i) = i + \chi_{\text{cen}}(R_p)$; in the limit this recursion approaches $2i$.

The graphs in Lemma 8 contain large cliques. We now show that this is not a necessary condition for the linear and centered coloring numbers to diverge.
Lemma 9. Let $B_\ell$ be the complete binary tree with $\ell$ levels. Then

$$
\lim_{\ell \to \infty} \frac{\chi_{\text{cen}}(B_\ell)}{\chi_{\text{lin}}(B_\ell)} \geq \frac{3}{2}
$$

Proof. A straightforward inductive argument shows that $\chi_{\text{cen}}(B_\ell) = \ell$. For any integer $d$ we can color $B_{3d}$ with $2d + 4$ colors recursively as follows. We divide $B_{3d}$ into $d$ "stripes" of three levels each and color each one with the $w, x, y, z$ pattern shown in Figure 5.1b. If $w_i, x_i, y_i, z_i$ are the colors used in stripe $i$, we let $x_i = x_j$ and $y_i = y_j$ if and only if $i = j$, while $w_i = w_j$ and $z_i = z_j$ exactly when $i$ and $j$ have the same parity. In this way, the colors $w_i, z_i$ are reused across different stripes.

To prove this is a linear coloring of $B_{3d}$, observe that every path within a single stripe has a center. Since every path in stripe $i$ that includes a leaf has $x_i$ or $y_i$ as a center and those colors are not reused in other stripes, every path with vertices in multiple stripes must also have a center. In the limit this coloring has size $\frac{2}{3} \ell$, proving the lemma.

We conjecture that the construction in Lemma 8 is optimal.

Conjecture 2. For any graph $G$, $\chi_{\text{cen}}(G) \leq 2\chi_{\text{lin}}(G)$.

While the exclusion of a path of length $2^k$ indicates $t_{\text{max}}(k) \leq 2^k$, this nonetheless leaves a large gap between the upper and lower bounds on $t_{\text{max}}(k)$. To move towards a proof of Conjecture 2, we consider two restricted graph classes—namely, trees and interval graphs—in the next two sections and establish polynomial upper bounds on $t_{\text{max}}(k)$ for graphs in these classes.
5.4 Upper Bounds on Trees

Schäffer proved that there is a linear time algorithm for finding a minimum-sized centered coloring of a tree $T$ \cite{schaffer1990}. In this section we prove the following theorem by showing a correspondence between the centered coloring from Schäffer’s algorithm and colors on paths in any linear coloring of $T$.

**Theorem 5.** There exists a polynomial time algorithm that takes as input a tree $T$ and a linear coloring $\psi$ of $T$ with size $k$ and outputs a centered coloring of $T$ whose size is at most $O(k^3)$.

Schäffer’s algorithm finds a particular centered coloring whose colors are ordered in a way that reflects their roles as centers. For this reason, the coloring is called a *vertex ranking* and the colors are referred to as *ranks*; it guarantees that in each subgraph, the vertex of maximum rank is also a center. We will use this terminology in this section to clearly distinguish between the ranks in the vertex ranking and colors in the linear coloring. Note that the canonical centered coloring of a treedepth decomposition is a vertex ranking if the colors are ranked decreasing from the root downwards, which implies that every centered coloring can be converted to a vertex ranking of the same size. Of central importance to Schäffer’s algorithm are what we will refer to as *rank lists*.

**Definition 25.** For vertex ranking $r$ of tree $T$, the rank list of $T$, denoted $L(T)$, can be defined recursively as $L(T) = L(T \setminus T_v) \cup \{r(v)\}$ where $v$ is the vertex of maximum rank in $T$.

Schäffer’s algorithm arbitrarily roots $T$ and builds the ranking from the leaves to the root of $T$, computing the rank of each vertex from the rank lists of each of its children.

**Proposition 10** (\cite{schaffer1990}). Let $r$ be a vertex ranking of $T$ produced by Schäffer’s algorithm and let $v \in T$ be a vertex with children $u_1, \ldots, u_l$. If $x$ is the largest integer appearing on rank lists of at least two children of $v$ (or 0 if all such rank lists are pairwise disjoint) then $r(v)$ is the smallest integer satisfying $r(v) > x$ and $r(v) \notin \bigcup_{i=1}^l L(u_i)$.

Our proof of Theorem 5 is based on tracking sets of colors of $\psi$ on leaf paths as Schäffer’s algorithm moves up the rooted tree. Define the *color vector* of a path $P$ with respect to linear coloring $\psi$ to be the set of colors from $\psi$ appearing on $P$. Let $S(v)$ be the set of all color vectors of all leaf paths in $T_v$. Let $\kappa(v)$ be the maximum cardinality of any color vector in $S(v)$ and $S_\kappa(v) = \{C \in S(v) : |C| = \kappa(v)\}$. We show below that every vertex $v$ has a corresponding vertex $u$ that is “similar” in rank but “dissimilar” in values of $\kappa$ and/or $S_\kappa$.

**Lemma 10.** Let $\psi$ be a linear coloring of tree $T$ such that $|\psi| = k$ and $v$ a vertex of rank $i > k$. There exists a vertex $u \in T_v$ such that
• \(i - \kappa(v) - 1 < r(u) < i\) and

• Either \(\kappa(u) < \kappa(v)\) or \(|S_\kappa(u)| \leq \left\lfloor \frac{1}{2} |S_\kappa(v)| \right\rfloor\).

**Proof.** We assume that every vertex \(u\) such that \(i - \kappa(v) - 1 < r(u) < i\) satisfies \(\kappa(u) = \kappa(v)\), as the lemma is trivially satisfied otherwise. For any vertex \(w \in T_v\) such that \(\kappa(w) = i\), \(S_\kappa(w) \subseteq S_\kappa(v)\) since the leaf paths to \(w\) are subpaths of leaf paths to \(v\). Consequently, if \(T\) contains two unrelated vertices \(u, u'\) such that \(\kappa(u) = \kappa(u') = i\), then \(S_\kappa(u) \cap S_\kappa(u') = \emptyset\) or else there is a path through the least common ancestor of \(u, u'\) that has no center. Assuming WLOG that \(|S_\kappa(u)| \leq |S_\kappa(u')|\), the pigeonhole principle implies that \(|S_\kappa(u)| \leq \left\lfloor \frac{1}{2} |S_\kappa(v)| \right\rfloor\).

If no such unrelated vertices exist, then every vertex in \(P\) with rank between \(i - \kappa(v) - 1\) and \(i\) must lie on some common leaf path \(P\). Consider ordering the vertices of \(P\) sequentially such that the leaf is the first vertex and the root is the last vertex. Let \(w_j\) be the \(j\)th vertex on \(P\) of rank at least \(i - \kappa(v)\) and \(w'_j\) the child of \(w_j\) on \(P\) for \(j > 1\). By this definition and Proposition 10, \((i - \kappa(v)) \in L(w'_2)\) and \(r(w_2) = i - \kappa(v) + 1\). However, \((i - \kappa(v)) \notin L(w'_4)\) so \(r(w_4) = i - \kappa(v) + 2\). Continuing this pattern, we observe that if \(j = 2^x\), \(w_j\) is the first vertex on \(P\) with rank \(i - \kappa(v) + x\). Thus the path from \(w_1\) to \(v\) contains at least \(2^{\kappa(v)}\) vertices, which means it must have at least \(\kappa(v) + 1\) colors in \(\psi\). But this is impossible since the size of \(\psi|_P\) is at most \(\kappa(v)\), and thus the lemma is proven. \(\square\)

Using Lemma 10 as a recursive step, we prove Theorem 5 by tracing the values of functions \(\kappa\) and \(S\) down towards the leaves.

**Proof of Theorem 5.** Let \(u_i\) be the vertex of maximum rank in \(T\). There is a maximal sequence of vertices \(u_2, \ldots, u_q\) such that \(u_{i+1}\) satisfies the properties of Lemma 10 with respect to \(T_{u_i}\). Note that the ranks of \(u_1, \ldots, u_q\) are monotonically decreasing and \(r(u_i) - r(u_{i+1}) \leq \kappa(u_i)\). Moreover, every vertex in \(T\) satisfies

\[
1 \leq |S_\kappa(v)| \leq \binom{k}{\kappa(v)}
\]

and \(1 \leq \kappa(v) \leq k\). Since the only vertices with \(\kappa(v) = 1\) are the leaves,

\[
r(u_i) \leq \sum_{i=1}^{k} i \left( \log_2 \binom{k}{i} + 1 \right) \leq O(k^3).
\]

Consequently, \(r\) is a vertex ranking of size at most \(O(k^3)\) that can be computed in linear time. \(\square\)
5.5 Upper Bounds on Interval Graphs

Because linear colorings are equivalent to centered colorings when restricted to paths, we turn our attention to the linear coloring numbers of “pathlike” graphs. We investigate a particular class of “pathlike” graphs in this section and prove a polynomial relationship between their centered and linear coloring numbers.

**Definition 26.** A graph $G$ is an interval graph if there is an injective mapping $f$ from $V(G)$ to intervals on the real line such that $uv \in E(G)$ iff $f(u)$ and $f(v)$ overlap.

We refer to the mapping $f$ as the interval representation of $G$. Since the overlap between intervals $f(u)$ and $f(v)$ is independent of the interval representations of the other vertices, every subgraph of an interval graph is also an interval graph. The interval representation of $G$ implies a natural “left-to-right” layout that gives it “pathlike” qualities, which are manifested in restrictions on the length of induced cycles (chordal) and paths between vertex triples (AT-free).

**Definition 27.** A graph is chordal if it contains no induced cycles of length $\geq 4$.

**Definition 28.** Vertices $u, v, w$ are an asteroidal triple (AT) if there exist $uv$-, $vw$-, and $wu$-paths $P_{uv}, P_{vw},$ and $P_{wu}$, respectively, such that $N[w] \cap P_{uv} = N[u] \cap P_{vw} = N[v] \cap P_{wu} = \emptyset$. A graph with no AT is called AT-free.

**Proposition 11** ([56]). Graph $G$ is an interval graph iff $G$ is chordal and AT-free.

Intuitively, Definition 28 is a set of three vertices such that every pair has a path that avoids the neighbors of the third. Roughly speaking, in the context of linear colorings Proposition 11 indicates that if $w$ is a center of a “long” $uv$-path $P$ in $G$, any vertex $w'$ such that $\psi(w) = \psi(w')$ must have a neighbor on $P$. We devote the rest of this section proving Theorem 6.

**Theorem 6.** There exists a polynomial time algorithm that takes as input an interval graph $G$ and a linear coloring of $\psi$ of $G$ with size $k$ and outputs a centered coloring of $G$ with size at most $k^2$.

Our algorithm makes extensive use of the following well-known property of maximal cliques in interval graphs.

**Proposition 12** ([56]). If $G$ is an interval graph, its maximal cliques can be linearly ordered in polynomial time such that for each vertex $v$, the cliques containing $v$ appear consecutively.

In particular, we identify a prevailing path whose vertices “span” the maximal cliques and a prevailing subgraph that consists of the prevailing path as well as vertices in maximal cliques “between” consecutive vertices on the prevailing path. We will show that any linear coloring is
a centered coloring when restricted to the prevailing subgraph and that after removing the prevailing subgraph, the remaining components each use fewer colors.

Let \( C_1, \ldots, C_m \) be an ordering of the maximal cliques of \( G \) that satisfies Proposition 12. We say vertex \( v \) is introduced in \( C_i \) if \( v \in C_i \) but \( v \notin C_{i-1} \), and denote this as \( I(v) = i \). Likewise, \( v \) is forgotten in \( C_j \) if \( v \in C_j \) but \( v \notin C_{j+1} \), and denote this as \( F(v) = j \). The procedure for constructing a prevailing subgraph and prevailing path is described in Algorithm 7. This algorithm selects the vertex \( v \) from the current maximal clique that is forgotten “last” and adds \( v \) to the prevailing path and \( C_{F(v)} \) to the prevailing subgraph. We prove in Lemma 11 that if \( P, Q \) are a prevailing path and subgraph, the vertices in \( Q \setminus P \) can be inserted between vertices of \( P \) to form a Hamiltonian path of \( Q \).

Algorithm 7 prevailing\((G)\)

**Input:** Interval graph \( G \)

**Output:** Prevailing path \( P \), prevailing subgraph \( Q \)

1: \( C_1, \ldots, C_m \leftarrow \) maximal cliques of \( G \) labeled in accordance with Proposition 12
2: \( P \leftarrow \emptyset \)
3: \( V_Q \leftarrow \emptyset \)
4: \( i \leftarrow 1 \)
5: \( j \leftarrow 1 \)
6: while \( i < m \) do
7: \( v_j \leftarrow \arg \max_{u \in C_i} F(u) \)
8: \( P \leftarrow P \cup \{v_j\} \)
9: \( i \leftarrow F(v) \)
10: \( V_Q \leftarrow V_Q \cup V(C_i) \)
11: \( j \leftarrow j + 1 \)
12: end while
13: \( Q \leftarrow G[V_Q] \)
14: return \( P, Q \)

**Lemma 11.** Every prevailing subgraph has a Hamiltonian path.

**Proof.** Let \( P, Q \) be the prevailing path and subgraph constructed in Algorithm 7. We prove the lemma by constructing a Hamiltonian path of \( Q \). Let \( M_j \) be the set of all \( u \in Q \setminus P \), for which \( j \) is the smallest integer for which \( u \in C_{F(v_j)} \). In other words \( M_j \) contains the vertices in \( C_{F(v_j)} \) that do not appear in \( C_{F(v_{j-1})} \). If \( M = \bigcup_{1 \leq j \leq p} M_j \) then by construction \( P \cup M = Q \). Moreover, for each \( u \in M_j \), \( M_j \cup \{v_j, v_{j+1}\} \subseteq N[u] \). For each \( M_j \), let \( \mu_j^1, \mu_j^2, \ldots, \mu_j^{M_j} \) be a ordering of \( M_j \).
such that $F(\mu_i^j) \leq F(\mu_i^{j+1})$. Then

$$v_1, \mu_1^1, \ldots, \mu_1^{M_1}, v_2, \mu_2^1, \ldots, \mu_2^{M_2}, \ldots, v_p, \mu_p^1, \ldots, \mu_p^{M_p}$$

is a Hamiltonian path of $Q$.

Although the fact that the prevailing subgraph $Q$ has a Hamiltonian path implies $Q$ has a center with respect to $\psi$, we must ensure that the proper subgraphs of $Q$ also have a center. In Lemma 12, we prove $\psi|_Q$ is centered by showing every proper connected subgraph of $Q$ also has a Hamiltonian path.

**Lemma 12.** If $Q$ is a prevailing subgraph of an interval graph $G$ and $\psi$ a linear coloring of $G$, $\psi|_Q$ is a centered coloring.

**Proof.** It suffices to show that every proper, connected induced subgraph of $Q$ has a Hamiltonian path, since the existence of a Hamiltonian path implies the subgraph has a center. Assume $H \subseteq Q$ has a Hamiltonian path. Let $w$ be a center and $w_p, w_s$ be its predecessor and successor in the Hamiltonian path. It is clear that the subpath from the start of the Hamiltonian path to $w_p$ remains a path in $H \setminus \{w\}$; this is also true for the subpath from $w_s$ to the end. Therefore if $H \setminus \{w\}$ is disconnected, there are two components and both have Hamiltonian paths.

Otherwise suppose $H \setminus \{w\}$ is connected. Note that if $P = \{v_1, \ldots, v_p\}$ is the prevailing path generated by Algorithm 7, $C_{F(v_j)} \cap C_{F(v_{j+1})}$ = $\emptyset$ or else $v_{j+2}$ would be forgotten later than $v_{j+1}$ and would have been chosen to be $v_{j+1}$ instead. Thus, there is some $1 \leq \ell \leq q \leq p$ such that $H \setminus \{w\} \subseteq C_{F(v_j)} \cup C_{F(v_{j+1})} \cup \ldots \cup C_{F(v_q)}$ and since $H \setminus \{w\}$ is connected, for each $\ell \leq j < q$ the intersection of cliques $C_{F(v_j)}$ and $C_{F(v_{j+1})}$ is non-empty. Consequently, the ordering of the vertices in the Hamiltonian path of $Q$ also defines a Hamiltonian path of $H \setminus \{w\}$. 

Since any linear coloring $\psi$ of the prevailing subgraph $Q$ must also be a centered coloring, $\text{td}(Q) \leq |\psi|$. To get a bound on the treedepth of $G$, we focus on the relationship between $Q$ and $G \setminus Q$. In particular, we show that the components of $G \setminus Q$ use fewer than $|\psi|$ colors by proving that each such component has an apex in the prevailing path.

**Lemma 13.** Let $P, Q$ be a prevailing path and subgraph of an interval graph $G$. For each component $X$ of $G \setminus Q$, there is a vertex $a \in P$ such that $X \subseteq N(a)$.

**Proof.** For $1 \leq j \leq p$, let $X_j$ be the set of components of $G[\bigcup_{i=F(v_j)}^{F(v_{j+1})-1} C_i] \setminus Q$, defining $F(v_0) = 0$. By this definition and the fact that $v_j$ is a member of both $C_{F(v_{j-1})}$ and $C_{F(v_j)}$, $v_j$ is a neighbor of all vertices in $X$ for each $X \in X_j$. Thus it suffices to show that $\bigcup_{j=1}^p X_j$ are the components of $G \setminus Q$.
Since \( V(Q) = \bigcup_{j=1}^{p} C_{F(v_j)} \), \( V(G) = V(Q) \cup V(\mathcal{X}_1) \cup \cdots \cup V(\mathcal{X}_p) \) and \( V(Q) \cap \bigcup_{j=1}^{p} \mathcal{X}_j = \emptyset \). Hence, if \( X \in \mathcal{X}_j \) is not a component of \( G \setminus Q \), then there must be some component \( X' \in \mathcal{X}_i \) for which \( i \neq j \) and there exists \( u \in X \) and \( u' \in X' \) and \( uu' \in E(G) \). But \( C_{F(v_j)} \cup C_{F(v_j+1)} \) has no common vertices with \( X \) and separates it from any vertices in \( \mathcal{X}_j \). An analogous statement for \( X' \) is true as well, so no such edges \( uu' \) exist. Therefore we conclude that \( \bigcup_{1 \leq j \leq p} \mathcal{X}_j \) are the components of \( G \setminus Q \) and the lemma is proven.

We can now establish a polynomial upper bound on the treedepth of interval graphs, proving Theorem 6.

**Proof of Theorem 6.** Let \( \mathcal{A} \) be the algorithm that constructs a treedepth decomposition \( T \) of \( G \) by finding a prevailing subgraph \( Q \) (Algorithm 7), using \( \psi|_Q \) to create a treedepth decomposition of \( Q \), and recursively constructing treedepth decompositions of \( G \setminus Q \). If \( \text{depth}(T) \leq k^2 \) and \( \mathcal{A} \) runs in polynomial time, then the canonical centered coloring of \( T \) is a centered coloring of \( G \) of size at most \( k^2 \). We prove \( \mathcal{A} \) satisfies these requirements by induction on \( k = |\psi| \). At \( k = 1 \), the graph consists of isolated vertices and \( \mathcal{A} \) trivially constructs a treedepth decomposition of \( G \) of depth 1 in polynomial time.

Assume \( \mathcal{A} \) has the desired properties for linear colorings of size at most \( k - 1 \). Because the maximal cliques of an interval graph can be enumerated and ordered in polynomial time (Proposition 12), identifying \( Q \) via Algorithm 7 can be done in polynomial time. By Lemma 12, the canonical treedepth decomposition of \( Q \) has depth at most \( k \). Since every component \( X \) of \( G \setminus Q \) has an apex \( a \) in \( P \) (Lemma 13), we can assume \( a \) is an ancestor in \( T \) of each vertex in \( X \) (Lemma 7). Because \( \psi \) is proper, \( \psi(a) \) does not appear in \( \psi|_X \) and since induced subgraphs of interval graphs are themselves interval graphs, \( \mathcal{A} \) finds a treedepth decomposition of \( X \) whose depth is at most \( (k - 1)^2 \). Thus \( T \) has depth \( k + (k - 1)^2 \leq k^2 \). The recursion only lasts \( k \leq n \) steps, so \( \mathcal{A} \) runs in polynomial time.

**5.6 Hardness of Recognizing Linear Colorings**

Based on the similarity in definition between linear and centered colorings, one might assume that computing them should be roughly equally difficult. Finding a centered coloring of a fixed size is NP-hard [9], but given a coloring of a graph, we can recognize whether it is centered in polynomial time by attempting to create the canonical treedepth decomposition; this procedure will identify a non-centered subgraph if the coloring is not centered. To the contrary, we will prove that LINEAR COLORING RECOGNITION, the problem of recognizing whether a coloring is linear, is co-NP-complete. In order to prove the hardness of LINEAR COLORING RECOGNITION, we first define a dual problem. The NON-CENTERED PATH problem takes a graph \( G \) and coloring \( \psi \) as input and decides whether \( G \) has a non-centered path \( P \).
We focus on proving the hardness of Non-centered Path because a certificate to that problem is easily definable: a path where every color appears at least twice.

**Theorem 7.** Non-centered Path is NP-complete.

**Proof.** A certificate to Non-centered Path can be verified in linear time by iterating over all vertices in the path and counting color occurrences. Thus, Non-centered Path is in NP.

We prove NP-hardness by reducing from CNF-SAT. Given a CNF-SAT formula $\Phi$ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$, we construct a graph $G$ and coloring $\psi$ that will have a non-centered path if and only if $\Phi$ is satisfiable. We assume that $\Phi$ satisfies the following properties:

1. Every variable appears at most once in each clause.
2. No clause contains both a variable and its negation.
3. Every variable appears as a positive literal and negative literal.

We can assume (1) since the disjunction operation is idempotent. Every clause for which (2) does not hold is satisfied by any truth assignment of the variables and thus can be removed without changing the satisfiability of $\Phi$. If variable $x_i$ appears only positively then assigning $x_i$ to be false does not cause any clauses to be satisfied. Therefore, it is sufficient to set $x_i$ to true and only consider the clauses of $\Phi$ that do not contain $x_i$; since the analogous statement is true when $x_i$ does not appear positively, we can assume (3).

The variables of $\Phi$ are represented by a set of vertices $U = \{u_0, \ldots, u_n\}$. For each $x_i$, we connect $u_{i-1}$ and $u_i$ with two paths $P_T^i$ and $P_F^i$; we will force the non-centered path to contain vertices from exactly one of $P_T^i$ and $P_F^i$, which will correspond to whether $x_i$ was set to true or false.
or false. The path \( P_i^T \) contains one vertex for each \( C_j \) in which \( x_i \) appears positively while \( P_i^F \) contains one vertex for each \( C_j \) in which the negation of \( x_i \) appears. By assumption (2), we can label this vertex uniquely with \( w_{ij} \) and the order of the vertices on \( P_i^T \) and \( P_i^F \) can be chosen arbitrarily. To complete the construction of \( G \), we add path \( P_0 = u_1', u_2', \ldots, u_n', w_1', w_2', \ldots, w_m' \) such that \( w_m' \) is adjacent to \( u_0 \) and all other vertices on \( P_0 \) have no additional edges. Finally, we attach a pendant vertex \( u_0' \) to \( u_n \). Since each vertex \( w_{ij} \) corresponds to a unique literal in \( \Phi \) and \( |U| + |P_0| = 2n + m + 1 \), \( G \) has size linear in the size of \( \Phi \).

To encode satisfaction of clauses, we color \( G \) with coloring \( \psi : V(G) \rightarrow \{0, \ldots, n + m\} \) such that \( \psi(u_i) = \psi(u_i') = i \) and \( \psi(w_j) = \psi(w_{ij}) = n + j \). In this way, we force any non-centered path to contain all colors and color \( j + n \) appears twice if and only if \( C_j \) is satisfied. An example can be found in Figure 5.2.

We now prove that \( \Phi \) is satisfiable if \( G \) contains a path \( Q \) with no center. Given a satisfying assignment of \( \Phi \), let \( P_i^* \) be \( P_i^T \) if \( x_i \) is set to true and \( P_i^T \) if \( x_i \) is set to false. Then \( Q = P_0 \cdot u_0 \cdot P_1^* \cdot u_1 \cdot \cdots \cdot P_n^* \cdot u_n \cdot u_0' \) is a non-centered path since it contains all pairs \( u_i, u_i' \) and \( \bigcup_{1 \leq i \leq n} P_i^* \) contains a vertex with the same color as each vertex in \( \bigcup_{1 \leq i \leq m} w_i' \).

To prove the reverse direction suppose \( G \) contains a non-centered path \( Q \). Let \( U' = \{u_0', \ldots, u_n'\} \). Since each vertex in \( U' \) shares a color with exactly one other vertex and that vertex is a member of \( U \), \( Q \) contains a vertex from \( U \) iff \( Q \) contains a vertex from \( U' \). By our construction of \( P_0 \) and assumptions about \( \Phi \), no component of \( G \setminus (U \cup U') \) contains two vertices of the same color. Thus, \( Q \) must contain vertices from \( U \), \( U' \), and \( G \setminus (\cup \cup U') \). For any \( 0 \leq i \neq j \leq n \), every \( u_i u_j \) path contains \( u_i \) or \( u_j \), which implies that \( (U \cup U') \subset Q \) and \( Q \) is a \( u_i' u_j' \) path. In order for \( Q \) to be connected, \( P_0 \subseteq Q \) and in order for it to be a path, exactly one of \( P_i^T \) and \( P_i^F \) (denote it \( P_i^* \)) is a subpath of \( Q \) for each \( 1 \leq i \leq n \). Since the colors in \( w_1', \ldots, w_m' \) are unique, \( \bigcup_{1 \leq i \leq n} P_i^* \) contains at least one vertex of each color on \([n + 1, n + m] \), which corresponds to a selection of truth assignments to the variables of \( \Phi \) such that every clause is satisfied.

\[ \square \]

**Corollary 1.** **Linear Coloring Recognition is co-NP-complete.**

The co-NP-hardness of recognizing linear colorings is compounded by two stronger hardness implications. First, the coloring \( \psi \) given in Theorem 7 has size \( m + n + 1 \), which means that unless ETH (Conjecture 1) fails, there is no \( \mathcal{O}(k) \) algorithm to recognize a linear coloring of size \( k \). Second, the graph \( G \) constructed in Theorem 7 is outerplanar with pathwidth two, which implies that neither treewidth-style dynamic programming nor a Baker-style layering approach is likely to solve this problem efficiently. Nonetheless, the fact that \( \chi_{cen}(G) = O(\log m + \log n) \) while \( |\psi| = m + n + 1 \) leaves open the possibility that Linear Coloring Recognition becomes easier for colorings of minimum size.
5.7 Conclusion

We have introduced $p$-linear and linear colorings as an alternative to $p$-centered and centered colorings for use in algorithms for classes of bounded expansion. The $p$-linear colorings are computable in polynomial time and require a constant number of colors in classes of bounded expansion, while inducing graphs of bounded treedepth for all small sets of colors, allowing direct substitution in existing algorithmic pipelines. A major direction for future work is to bring the upper bound on $t_{\text{max}}(k)$ of $2^k$ closer to the lower bound of $2k$. In particular, it appears our current toolkit for analyzing linear colorings must be expanded in order to prove (or disprove) Conjecture 2. We also believe it is worth studying whether recognizing linear colorings can be done in polynomial time if we assume the coloring is of size $\chi_{\text{lin}}(G)$. Finally, using $p$-linear colorings in practice will require an efficient method for translating a linear coloring into a treedepth decomposition. Although there exist general-purpose algorithms to find treedepth decompositions efficiently in graphs of bounded linear coloring number (e.g. [84]), a more specialized algorithm that avoids “heavy machinery” is likely necessary to be practically useful.
Summary

We study the computational complexity of identifying dense substructures, namely $r/2$-shallow topological minors and $r$-subdivisions. Of particular interest is the case $r = 1$, when these substructures correspond to very localized relaxations of subgraphs. Since Densest Subgraph can be solved in polynomial time, we ask whether these slight relaxations also admit efficient algorithms.

In the following, we provide a negative answer: Dense $r/2$-Shallow Topological Minor and Dense $r$-Subdivision are already NP-hard for $r = 1$ in very sparse graphs. Further, they do not admit algorithms with running time $2^{o(tw^2)}n^{O(1)}$ when parameterized by the treewidth of the input graph for $r \geq 2$ unless the exponential time hypothesis fails. The work in this chapter was published in the 42nd International Symposium on Mathematical Foundations of Computer Science [69].
6.1 Introduction

Many structural graph theory results stem from the principle that excluding certain (dense) substructures affords algorithmic tractability [26, 28, 29, 32, 35, 36, 41, 83]. It is therefore natural to ask for each of these substructures whether computing its densest occurrence is efficiently possible. In this chapter, we characterize the complexity of this problem for substructures formed by contracting short disjoint paths.

Shallow minors (Definition 6) form a gradient between the locality of subgraphs (0-shallow minors) and the global nature of (∞-shallow) minor containment. Because finding \( r \)-shallow minors of density/degeneracy at least \( d \) is polynomial time solvable at \( r = 0 \) [37, 40] but NP-complete at \( r = \infty \) [10], one might expect the problem to be fixed-parameter tractable with respect to \( r \). However, Dvořák proved that for any fixed \( r > 0 \) both variations are NP-complete already in graphs of maximum degree four and \( d \geq 4 \) (\( d \geq 2 \) if degeneracy is the measure) [30]. Accordingly, a parameterization by \( d \) also cannot possibly yield an fpt-algorithm—a sharp contrast to unrestricted minors [10]. Dvořák further showed that the problem is in FPT if parameterized by the treewidth \( tw \) of the input graph and designed an \( O(4^{tw^2}n) \) dynamic programming algorithm.

We consider whether more conservative relaxations of subgraph containment remain efficiently solvable, specifically \( r/2 \)-shallow topological minors and \( r \)-subdivisions. A graph \( H \) is an \( r/2 \)-shallow topological minor of a graph \( G \) if a \((\leq r)\)-subdivision of \( H \) is isomorphic to some subgraph of \( G \). The case of \( r = 1 \) is of particular interest because it generalizes subgraphs to a proper subset of 1-shallow minors.

We show that Dense \( r/2 \)-Shallow Topological Minor (Dense \( r/2 \)-STM) and Dense \( r \)-Subdivision (Dense \( r \)-SD) are NP-complete already in subcubic apex-graphs\(^1\) for \( r \geq 1 \) via a reduction from Positive 1-in-3SAT. Accordingly, a parameterization by the target density \( d \) does not make these problems fixed-parameter tractable. The same reduction also implies that neither problem can be solved in time \( O(2^d(n)) \) unless the ETH (Conjecture 1) fails. In other words, finding dense substructures which are just slightly “less local” than subgraphs seems to be intrinsically difficult. Following the results of Dvořák, we then consider a parameterization by treewidth and ask whether an algorithm with running time better than \( O(2^{tw^2}n) \) is possible. Surprisingly, we can rule out such an algorithm already for Dense 1-Shallow Topological Minor: unless the ETH fails, no algorithm with running time \( O(2^{(tw^2)}n) \) can exist.

\(^1\)That is, a graph in which the removal of a single vertex results in a subcubic planar graph.
The graph $H$ is a 1-shallow topological minor of $G$, as witnessed by the model marked with blue nails and golden paths.

### 6.2 Background

A graph $H$ appears as an $r$-subdivision in a graph $G$ if the graph obtained from $H$ by subdividing every edge $r$ times is isomorphic to some subgraph of $G$. In this way, $H$ is a $r/2$-shallow topological minor of $G$ if a graph obtained from $H$ by subdividing every edge up to $r$ times is isomorphic to a subgraph of $G$.

The following two problems are the focus of this chapter:

**Dense $r/2$-Shallow Topological Minor (Dense $r/2$-STM)**

*Input:* A graph $G$ and a rational number $d$.

*Question:* Is there an $r/2$-shallow topological minor $H$ of $G$ with density $\|H\| / |H| \geq d$?

**Dense $r$-Subdivision (Dense $r$-SD)**

*Input:* A graph $G$ and a rational number $d$.

*Question:* Is there a graph $H$ that is contained in $G$ as an $r$-subdivision with density $\|H\| / |H| \geq d$?

The following variant, which we prove to be NP-complete in Section 6.4, might be of independent interest:

**Dense Bipartite Subdivision**

*Input:* A bipartite graph $(X, Y, E)$ and a rational number $d$.

*Question:* Are there subsets $X' \subseteq X, Y' \subseteq Y$ such that all vertices in $X'$ can be smoothed into unique edges in $Y'$ and $|X'| / |Y'| \geq d$?
Our main tool will be linear reductions from the following SAT-variant:

**Positive 1-in-3SAT**

*Input:* A CNF boolean formula \( \psi \) with only positive literals.

*Question:* Does \( \phi \) have a satisfying assignment such that each clause contains exactly one true variable?

Mulzer and Rote showed [68] that Positive 1-in-3SAT remains NP-hard when restricted to planar formulas. A formula \( \phi \) is planar if the graph obtained from \( \phi \) by creating one vertex for each clause and variable and connecting a variable-vertex to a clause-vertex if the clause contains said variable is planar.

Schaefer [88] provided a linear reduction from 3SAT to 1-in-3SAT. We can further easily transform a formula \( \phi \) with negative literals into one with only positive literals as follows: for each variable \( x \), introduce the variables \( x^+, x^-, a_x, b_x, c_x \). Replace every occurrence of \( x \) with \( x^+ \) and every occurrence of \( \bar{x} \) with \( x^- \) and add the clauses

\[
\{x^+, x^-, a_x\}, \{x^+, x^-, b_x\}, \{a_x, b_x, c_x\},
\]

to the formula. It is easy to verify that exactly one of \( x^+, x^- \) must be true in a 1-in-3 satisfying assignment and that the resulting formula \( \phi' \) has size linear in \(|\phi|\). In conclusion, there exists a linear reduction from 3SAT to Positive 1-in-3SAT which implies that under ETH, Positive 1-in-3SAT cannot be solved in time \( 2^{o(n)}(n + m)^{O(1)} \), where \( n \) is the number of variables and \( m \) is the number of clauses. Using sparsification one can further show that the ETH excludes algorithms for 3SAT with running time \( 2^{o(m)}(n + m)^{O(1)} \) (see, e.g., the survey by Cygan et al. [23]). The above reduction implies the following lower bound:

**Proposition 13.** Assuming ETH, Positive 1-in-3SAT cannot be solved in time \( 2^{o(m)}(n + m)^{O(1)} \).

### 6.3 Algorithmic Considerations

We start with a basic observation about the problems in question with the smallest sensible depths of \( r = 1 \):

**Lemma 14.** The densest \( \frac{1}{2} \)-shallow minor or 1-subdivision on a given set of nails can be computed in polynomial time.

**Proof.** Assume we are to find the densest 1-subdivision with nail set \( X \) in a graph \( G \). We construct an auxiliary bipartite graph \( \hat{G} \) with vertex set \( V(G) \setminus X \cup \{ \frac{X}{2} \} \) where the vertex \( v \in
$V(G) \setminus X$ is connected to $xy \in (X^2)$ iff $\{x, y\} \subseteq N(v)$ in $G$, that is, if $v$ can be contracted into the edge $xy$. Now simply note that a matching of cardinality $\ell$ in $\hat{G}$ corresponds to a 1-subdivision in $G$ with $\ell$ subdivisions. Finding a maximal matching in $\hat{G}$ therefore provides us with the densest 1-subdivision in $G$ with nail set $X$. The same proof works for $\frac{1}{2}$-shallow minors if we subdivide all edges existing inside $X$ and then construct $\hat{G}$.

Consequently, Dense 1-SD and Dense $\frac{1}{2}$-STM both admit a simple $2^nn^{O(1)}$-algorithm: we guess the nail set $X$ and apply the matching construction from Lemma 14. For the same reasons, both problems are in XP when parameterized by the number of nails. We cannot hope for much better since for $r = 0$ and $d \sim k^2$ we simply recover the problem of finding a $k$-clique. Besides being W[1]-hard and thus probably not in FPT, $k$-CLIQUE further does not admit algorithms with running time $f(k)n^{o(k)}$ unless the ETH fails [16].

The approach of guessing the nail sets also fails for larger depths: knowing the nails of $a$, say, 1-shallow minor leaves us with the problem of contracting paths of length two into $X$, which cannot be represented as a simple matching problem. The reduction presented in Section 6.5 proves as a corollary that Dense 1-STM remains NP-hard when the nail set of the densest minor is known.

Finally, as we will see in Section 6.4, both problems are already NP-complete for very small densities $d$, making them paraNP-complete under this parameterization. Therefore none of the input variables will work well as a parameterization, and it is sensible to consider structural parameters, meaning parameters derived from the input graph. A good contender for such parameters are width measures like tree-, path-, or cliquewidth. Indeed, we can express the problem of finding a dense shallow minor or a dense subdivision in MSO$_2$ and apply variants of Courcelle’s theorem to obtain the following:

**Proposition 14.** Dense $\frac{r}{2}$-STM and Dense $r$-SD are in FPT when parameterized by the treewidth of the input graph.

**Proof.** We can express a model for an $r$-shallow minor in MSO$_2$ as follows: it consists of a vertex-set $W$ and an edge set $F$, where $F$ induces a set of paths. We can further easily express that the paths formed by $F$ are a) of length at most $r$, b) disjoint, and c) have endpoints in $W$. Lastly, we demand that for every pair $x, y \in W$ there exists at most one path in $F$ that has $x$ and $y$ as endpoints.

From an optimization perspective, we can therefore express the feasible solutions to Dense $\frac{r}{2}$-STM (and Dense $r$-SD with small modifications). In order to express our optimization goal, let us introduce one more set of vertices $C$ with the property that every path induced by $F$ contains at most one vertex from $C$—for example, we can express in MSO$_2$ that vertices of $C$ are not pairwise reachable via the graph induced by $F$. With this auxiliary set, the density of
the resulting minor is at least \(|C|/|W|\) and exactly the density if \(C\) is maximal with respect to our choice of \(F\). Accordingly, we find that there exists an \(r\)-shallow topological minor of density at least \(d\) if \(|C| - d|W| \geq 0\). This constraint and the aforementioned MSO₂-description of a minor fall within the expressive power of the EMSO-framework introduced by Arnborg, Lagergren, and Seese [3] and we conclude that both \(\text{Dense } r/2\text{-STM}\) and \(\text{Dense } r\text{-SD}\) are fpt when parameterized by treewidth.

Furthermore, it is not difficult (albeit tedious) to design a dynamic programming algorithm that solves \(\text{Dense } r/2\text{-STM}\) and \(\text{Dense } r\text{-SD}\) in time \(2^{O(tw^2)} n\). The quadratic dependence on the treewidth stems from the fact that we have to keep track of which edges we have contracted so far and there is no obvious way to circumvent this. The important question was whether any of the known techniques to reduce the complexity of connectivity-problems [8, 24, 79] could be applied here. The answer is, to our surprise, negative as we will discuss in Section 6.5.

6.4 NP-Hardness

This section will be dedicated to the proof of the following theorems which both follow directly via a linear reduction from \(\text{Positive } 1\text{-in-3SAT}\).

**Theorem 8.** \(\text{Dense } r/2\text{-STM}\) and \(\text{Dense } r\text{-SD}\) are NP-hard for \(r \geq 1\), even when restricted to graphs that can be turned into subcubic planar graphs by deleting a single vertex.

**Theorem 9.** \(\text{Dense } r/2\text{-STM}\) and \(\text{Dense } r\text{-SD}\) cannot be solved in time \(2^{o(n)} n^{O(1)}\) on bipartite graphs unless the ETH fails.

A special case of our result might be of independent interest:

**Theorem 10.** \(\text{Dense Bipartite Subdivision}\) is NP-hard even on instances \(((X,Y,E),d)\) where vertices in \(X\) have degree at most 3 and \(d \geq 3\).

In the following, we present two reductions from \(\text{Positive } 1\text{-in-3SAT}\) that depend on the parity of \(r\). We describe the reduction for \(r \in \{1, 2\}\) and then argue how to modify the construction for arbitrary values of \(r\). Note that the resulting instances are such that the densest graph \(H\) that appears as a \(r/2\)-shallow topological minor appears, in fact, as an \(r\)-subdivision and thus the reductions work for both problems.

**Reduction for \(r\) odd**

Let \(\psi\) be a \(\text{Positive } 1\text{-in-3SAT}\) instance with clauses \(C_1, \ldots, C_m\) and variables \(x_1, \ldots, x_p\). We assume that every variable in \(\psi\) appears in at least 3 clauses; if not, we can duplicate clauses
to achieve this without changing the satisfiability of $\psi$. We construct a graph $G$ from $\psi$ in the following manner (see Figure 6.2):

1. For each variable $x_i$, create a cycle $D_i$ with as many vertices as the frequency of $x_i$ in $\psi$.
2. Create an apex-vertex $a$ that is connected to every vertex of the cycles $D_1, \ldots, D_p$.
3. For each clause $\{x_i, x_j, x_k\}$, add a vertex $u_{ijk}$ to the graph and connect it to one vertex in $D_i, D_j, D_k$ each that has not yet been connected to any clause-vertex.
4. Subdivide every edge appearing in the cycles $D_1, \ldots, D_p$ and all edges incident to the apex $a$.

For easier presentation, let us color the vertices of $G$ as follows: the vertices introduced in the first step are *white*, the vertices introduced in the third step *gray*, and the subdivision vertices created in the last step *black* (the apex vertex $a$ remains uncolored). Note that the graph $G$ is bipartite, where one side of the partition contains exactly the white vertices and $a$.

Note that if the input formula $\psi$ is planar, then the constructed graph is planar and subcubic after removing the apex vertex $a$.

**Lemma 15.** If $\psi$ is satisfiable then $G$ has a topological minor at depth $\frac{1}{2}$ of density $\frac{5m}{2m+1}$.
Proof. We construct the minor $H$ by first smoothing each black vertex. Then, for each variable set to true, we delete the corresponding cycle $D_i$. Since $\psi$ is satisfiable and each clause has exactly one variable set to true, this step deletes exactly one neighbor from each gray vertex.

We complete the construction of $H$ by smoothing out each gray vertex. $V(H)$ consists of exactly two vertices corresponding to each clause plus the apex $a$, for a total of $2m + 1$ vertices. Since all vertices of $H$ were colored white in $G$, $a$ has degree $2m$. Aside from the edges incident to $a$, there are $m$ edges from smoothing gray vertices and $2m$ edges from smoothing black vertices, which yield a total of $5m$ edges. Thus, we have found a minor at depth $1/2$ of density $\frac{5m}{2m+1}$.

Lemma 16. If $G$ has a topological minor at depth $1/2$ of density at least $\frac{5m}{2m+1}$, then the formula $\psi$ is satisfiable.

Proof. Let $H$ be the densest shallow topological minor at depth $1/2$ and fix some model of $H$ in $G$. We first argue that the nails of $H$ consist only of white vertices and potentially the apex vertex $a$.

Claim. The nails of $H$ consist of the apex vertex $a$ and some subset of white vertices.

First, since the density of $H$ is greater than two, its minimum degree is at least three (the removal of a degree-two vertex would increase the density). Since black vertices have degree two in $G$, the nails of the model forming $H$ therefore cannot be black. Accordingly, every black vertex either does not participate in the formation of $H$ or it is smoothed into an edge.

Let us define $G_b$ to be the graph obtained from $G$ by smoothing all black vertices. Since black vertices have degree exactly two, this operation is uniquely defined. By the previous observation, $H$ can be obtained from $G_b$ by only smoothing gray vertices and taking a subgraph. This of course implies that the nails of $H$ are all either gray, white, or the apex vertex $a$. Let us now exclude the first of these three cases: assume $y$ is a gray nail of $H$ in $G_b$. Again, the degree of $y$ in $H$ must be at least three to ensure maximal density of $H$, and since $y$ has degree three in $G$ it must also have degree exactly three in $H$. Note that the three neighbors of $y$ are necessarily white and independent in $G_b$, thus we can smooth $y$ into an (arbitrary) edge between two of its neighbors. The newly obtained graph $H'$ is again a half-shallow topological minor of $G$ and it contains one vertex and two edges less than $H$. Since the density of $H$ is greater than two, this implies that the density of $H'$ is greater than that of $H$, a contradiction. We conclude that the nails of $H$ cannot be gray and therefore only consist of white vertices and, potentially, the apex vertex $a$. To see that $a$ must be contained in $H$, simply note that otherwise the maximum degree of $H$ would be three and as thus $H$'s density would lie strictly below the assumed $\frac{5m}{2m+1}$. In summary: $H$'s nails consist of the apex vertex $a$ and some subset of white vertices of $G$, proving the claim.
Since the white vertices in $G$ are independent, the above claim further implies that the construction of $H$ can be accomplished without smoothing white vertices. We can therefore divide said construction into two steps: first we smooth all gray and black vertices to construct a graph $G_{gb}$ from $G$ and then we take the subgraph $H \subseteq G_{gb}$. In the following, we will refer to edges in $G_{gb}$ or $H$ as gray if they originated from smoothing a gray vertex and black if they originated from smoothing a black vertex. Note that the set of black and gray edges partition $E(G_{gb})$ and hence also $E(H)$.

We now denote by $v_4$ the number of degree-four vertices in $H$ and by $v_3$ the number of degree-three vertices (as observed above, no vertex with degree lower than three can exist in $H$ and $a$ is the only vertex of degree greater than four). Since the number of gray edges is at most $m$ and a degree-four vertex must be incident to a gray edge, we have that $v_4 \leq 2m$. Let $w = v_3 + v_4$ be the number of white vertices in $H$ and $\alpha = v_4/w$ the ratio of degree-four vertices among them. Using these quantities, we can express $H$’s density as

$$\frac{2v_3 + 5/2v_4}{v_3 + v_4 + 1} = 2 \frac{w}{w+1} + \frac{\alpha}{2} \frac{w}{w+1} = \left(2 + \frac{\alpha}{2}\right) \frac{w}{w+1}$$

which we combine with the density-requirement on $H$ to obtain

$$\left(2 + \frac{\alpha}{2}\right) \frac{w}{w+1} \geq \frac{5m}{2m+1} \iff \alpha \geq 2 \left(\frac{5m}{2m+1} \frac{w+1}{w}\right) - 4.$$ 

Note that the right-hand side is equal to one for $w = 2m$, smaller than one for $w > 2m$, and larger than one for $w < 2m$. This last regime would imply the impossible $\alpha > 1$ and we conclude $2m \leq w \leq 3m$, where the upper bound $3m$ is simply the total number of white vertices in $G$. Rewriting $w$ as $\beta m$ for $2 \leq \beta \leq 3$, we revisit the density-constraint on $H$:

$$\left(2 + \frac{\alpha}{2}\right) \frac{\beta m}{\beta m+1} \geq \frac{5m}{2m+1} \iff \left(2\beta + \frac{\alpha\beta}{2}\right)(\beta m + 1) \geq 5(\beta m + 1)$$

$$\iff (\alpha - 1)m + \frac{5}{2} \geq \frac{5}{\beta}.$$  

(\star)

We will now show that $\alpha$, the fraction of degree-four vertices among all $w$ white vertices, needs to be one in order for (\star) to hold. To that end, we distinguish the following two cases:

**Case 1:** $\beta = 2$. Assuming $\alpha \neq 1$, the largest possible value for $\alpha$ is achieved when $v_4 = 2m - 2$ (the case of exactly one gray edge missing from $H$), resulting in $\alpha = (2m - 2)/2m = 1 - \frac{1}{m}$. Plugging this value of $\alpha$ and $\beta = 2$ into (\star), we obtain that

$$(1 - 1/m - 1)m + \frac{5}{2} = -1 + \frac{5}{2} \geq \frac{5}{2}.$$
a contradiction. Smaller values of $\alpha$ lead to the same contradiction, and we conclude that necessarily $\alpha = 1$.

**Case 2:** $2 < \beta \leq 3$. Assuming $\alpha \neq 1$, the largest possible value for $\alpha$ is achieved when $v_4 = 2m$, resulting in $\alpha = 2m/\beta m = \frac{2}{\beta}$. Now $(\ast)$ becomes

$$
\left( \frac{2}{\beta} - 1 \right) m + \frac{5}{2} \geq \frac{5}{\beta} \iff m \leq \frac{10 - 5\beta}{2\beta} \cdot \frac{\beta}{2 - \beta} = \frac{5}{2}.
$$

Thus for formulas $\psi$ with at least three clauses, we arrive at a contradiction and conclude that $\alpha = 1$.

We have now shown that a) $H$ contains only vertices of degree four and b) that $|H| \geq 2m$. Since there cannot be more than $2m$ vertices of degree four, we conclude that $H$ has exactly $2m$ vertices. Note that therefore $H$ must consist of a collection of black-edge cycles with a total of $2m$ vertices, each of which is incident to exactly one of the $m$ gray edges. Note that each black cycle $B_i$ in $H$ corresponds to a cycle $D_i$ (associated with variable $x_i$) in $G$, where $B_i$ was constructed from $D_i$ by smoothing black vertices. Thus we can associate every black cycle $B_i$ in $H$ with a variable $x_i$ in $\psi$. We claim that setting all such variables $x_i$ that have a black cycle $C_i$ in $H$ to false and all other variables to true is a 1-in-3 satisfying assignment of $\psi$.

Consider any clause $\{x_i, x_j, x_k\}$ in $\psi$. The corresponding gray vertex $u_{ijk}$ in $G$ was smoothed into a gray edge $e_{ijk}$ in $H$, since all $m$ gray vertices are present in $H$. Accordingly, exactly two of the three black cycles $B_i, B_j, B_k$ are contained in $H$. Thus the assignment constructed above will set exactly two of the variables $x_i, x_j, x_k$ to false and one variable to true. This argument holds for every clause in $\psi$ and we conclude that the constructed assignment is 1-in-3 satisfying, proving the lemma.

This concludes the reduction from **Positive 1-IN-3SAT**. Note that an optimal solution in the reduction necessarily does not use any edges from the original graph, but only edges resulting from contractions. Therefore the reduction works for both **Dense $\frac{1}{2}$-STM** and **Dense $\frac{1}{2}$-SD**. As noted above, for $r = 1$ the constructed graph is bipartite. Since all gray and black vertices have degree at most three, Theorem 10 follows.

In order for the reduction to work for arbitrary odd $r$, we need to modify the construction in two places: first, we subdivide every edge in the clause gadget $(r - 1)/2$ times. Second, instead of subdividing all edges appearing in the cycles $D_1, \ldots, D_p$ and edges incident to the apex $a$ once, we subdivide them $r$ times. The correctness of this reduction follows from easy modifications to Lemma 15 and 16, concluding our proof of Theorem 8 for odd values of $r$. Finally, to see that the above reduction also proves Theorem 9 for odd $r$, simply note that the reduction results in a graph of size $\Theta(m)$ and the ETH lower bound follows from Proposition 13.
Reduction for $r$ even

Let $\psi$ be a positive 1-in-3SAT instance as described above. Construct graph $G$ in the following manner. We once again create a cycle $D_i$ for each variable $x_i$, connect an apex vertex $a$ to each vertex on the cycles, and color these vertices white. For this construction, however, we subdivide all edges between white vertices twice i.e. each white-white edge is replaced by a three-edge path. As with our previous construction, the subdivision vertices are all colored black. For each clause $C_i = \{x_j, x_k, x_\ell\}$, we add a triangle $u_{ij}, u_{ij}, u_{ik}$ and connect it to the vertices from $D_j, D_k, D_\ell$ corresponding to $C_i$ such that $u_{ij}$ is incident to the vertex from $D_j$ etc. We color each of these vertices gray.

**Lemma 17.** If $\psi$ is satisfiable then $G$ has a topological minor at depth 1 of density $\frac{5m}{2m+1}$.

**Proof.** We construct the minor $H$ by first smoothing each black vertex. Then, for each variable set to true, we delete the corresponding cycle $D_i$. Since $\psi$ is satisfiable and each clause has exactly one variable set to true, each gray triangle has two vertices of degree three and one of degree two. The degree two gray vertices are deleted, leaving the remaining gray vertices to lie on three-edge paths between white vertices. These paths are subsequently smoothed to create white-white edges.

$V(H)$ consists of exactly two vertices corresponding to each clause plus $a$, for a total of $2m + 1$ vertices. Since all vertices of $H$ were colored white in $G$, $a$ has degree $2m$. Aside from the edges incident to $a$, there are $m$ edges from smoothing gray vertices and $2m$ edges from smoothing black vertices, which yield a total of $5m$ edges. Thus, we have found a minor at depth 1 of density $\frac{5m}{2m+1}$. □

**Lemma 18.** If $G$ has a topological minor at depth 1 of density $\frac{5m}{2m+1}$ then $\psi$ is satisfiable.

**Proof.** Let $H'$ be the densest topological minor at depth 1. For the same reasons presented in Lemma 16, $H'$ has no black nails, and thus we can smooth all black vertices into white-white edges. This lack of black nails also implies that no white vertices can be smoothed to form a new edge incident to a gray vertex.

If $H'$ contains all the gray and white vertices, it has $3m$ degree 4 white vertices, $3m$ degree 3 gray vertices, and one apex $a$ with degree $3m$ for a total of $12m$ edges and $6m + 1$ vertices. This implies a density below $\frac{5m}{2m+1}$, and thus not all white and gray vertices are nails.

Since the gray vertices induce triangles, there is no way to smooth gray vertices to create a new gray-gray edge. Consider one such triangle $T_i$. If we smooth two vertices in $T_i$ to create a single gray-white edge, the gray nail has degree 2 and should be deleted instead to increase the density. On the other hand, smoothing exactly one gray vertex to create a gray-white edge cause the remaining gray vertex to have degree two. Thus, any gray nail in $H'$ is adjacent.
to three white vertices. Note that instead of having a gray nail, we could delete one gray
vertex and smooth the other two into a white-white edge. The proof in Lemma 16 already
demonstrated that forming the white-white edges is necessary to yield a density of \( \frac{5m}{2m+1} \), and
thus \( H' \) has no gray nails.

Since the gray vertices must be smoothed and deleted to create two degree 4 vertices and
one degree 3 vertex per clause, the arguments in Lemma 16 imply that for \( H' \) to have density
\( \frac{5m}{2m+1} \), \( \psi \) must be satisfiable.

In order for the reduction to work for arbitrary even \( r \), we again modify the construction in
two places: first, we subdivide every edge of the triangle making up the clause gadget \( r/2 - 1 \)
times. Second, instead of subdividing all edges appearing in the cycles \( D_1, \ldots, D_p \) and edges
incident to the apex \( a \) twice, we subdivide them \( r \) times. With both cases of \( r \) even or odd
covered, we conclude that Theorem 8 and Theorem 9 hold true.

### 6.5 ETH Lower Bounds

We show in this section that the ETH implies that we cannot get a single-exponential algorithm
parameterized by treewidth for Dense \( r/2 \)-STM for \( r \geq 2 \).

**Theorem 11.** Unless the ETH fails, there is no algorithm that decides Dense 1-Shallow Topological Minor on a graph with treewidth \( t \) in time \( 2^{o(t^2)}n^{O(1)} \).

Our proof proceeds via a reduction from CNF-SAT. Assume that the CNF formula \( \Phi \) with
variables \( x_1, \ldots, x_n \) and clauses \( C_1, \ldots, C_m \) is such that \( \sqrt{n} \) is an even integer; if not, we pad
\( \Phi \) with dummy variables that appear in no clauses, which does not affect the answer to \( \Phi \). Figure 6.3 contains a sketch of the construction outlined in the following.

**Decision gadget:** The reduction will use sequences of vertices connected by decision gadgets.
The decision gadget is a path of three vertices \( d_L, d_C, d_R \) which we will always connect to a
sequence of three vertices. For a sequence of vertices \( v_1, v_2, v_3 \), connecting the decision gadget
to the sequence involves adding the edges \( \{d_L, v_1\}, \{d_C, v_2\}, \) and \( \{d_R, v_3\} \).

**Variable gadgets:** We construct a grid of vertices \( R \) with \( \sqrt{n} \) rows and \( m \) columns, denoting with \( R[i,j] \) the vertex in the \( i \)th row and \( j \)th column. Each variable \( x_i \) will be represented by a
sequence of \( m \) vertices \( X_i \), one from each column. We will denote with \( X_i[j] \) the \( j \)th vertex in
the sequence \( X_i \), for any \( 1 \leq j \leq m \). In order to represent each of the \( n \) variables with \( m \) vertices
using a \( \sqrt{n} \times m \) grid, the sequences \( X_i \) must overlap with each vertex in \( R \) being part of the
representation of \( \sqrt{n} \) variables. Specifically, let \( X_i[j] = R[(i + j i/\sqrt{n}) \) mod \( \sqrt{n}, j] \). In other
words, two vertices in successive columns will be in the same row in sequences \( X_1, \ldots, X_{\sqrt{n}} \).
they will be one row apart ("wrapping around" to the top from the bottom) in sequences $X_1, X_2, \ldots, X_k$ and so on.

For each sequence $X_i$, we connect $X_i[j-1], X_i[j], X_i[j+1]$ to a decision gadget. Denote such a decision gadget as $D_{i,j}$. We also "wrap around" $X_i$ by connecting $X_i[m-1], X_i[m], X_i[1]$ and $X_i[1], X_i[2]$ to their own decision gadgets.

Clause gadgets: Each clause $C_i$ will be represented by a bipartition of vertices $A_i, B_i$ where $|A_i| = |B_i| = \sqrt{n}$. Let $A_i[j]$ be the $j$th vertex in $A_i$ and $B_i[j]$ likewise. Let $\sigma$ be an ordering of the vertices in $A_i \cup B_i$ corresponding to an Eulerian tour of a biclique with bipartition $A_i, B_i$. Assume without loss of generality that $\sigma_1 = A_i[1], B_i[1], \ldots, B_i[\sqrt{n}], A_i[\sqrt{n}]$ and note that every pair of vertices $a \in A_i$ and $b \in B_i$ appears consecutively exactly once in $\sigma$. For each consecutive triple of vertices in $\sigma$, attach a decision gadget (but do not "wrap around").

Connecting variables and clauses: For each pair of vertices $(A_i[j], B_i[k])$ for $1 \leq j, k \leq \sqrt{n}$ associate the pair with a unique variable $x_{ij}$. Connect $x_{ij}$ to $A_i[j]$ and $B_i[k]$ via 3-edge paths. If $x_{ij}$ appears in clause $C_i$ positively, connect $d_L$ of the decision gadget $D_{ij}$ to $B_i[k]$ via an edge and to $A_i[j]$ via a 2-edge path. If it appears negatively, add the same connections to $d_R$ of $D_{ij}$ instead.
With the description of the reduction completed, let us now prove its correctness, i.e., we prove that $G$ has a 1-STM of density $\rho = \frac{4\sqrt{n}}{3}$ if and only if $\Phi$ is satisfiable.

**Forward direction:** To prove the forward direction, we show how the satisfying assignment yields a topological minor of the desired density. We note that a cyclical sequence of vertices joined by decision gadgets can form a cycle in one of two ways: by smoothing each $d_C$ and $d_L$ or each $d_C$ and $d_R$. Let the former be known as the left configuration of those sequenced gadgets and the latter the right configuration. Create a cycle on the vertices in $X_i$ by choosing the right configuration if $x_i$ is true and the left configuration if $x_i$ is false.

For each clause, pick an arbitrary variable $x_\ell$ that satisfies it and let $a$ and $b$ be the pair of vertices from $A_i$ and $B_i$ assigned to that variable. If $a <_\sigma b$, set all decision gadgets preceding $a$ in $\sigma$ to the left configuration and all the decision gadgets succeeding $b$ to the right configuration; do the reverse if $b <_\sigma a$. Thus $A_i$ and $B_i$ form a biclique missing the $ab$ edge. Since there is a 3-edge path from $a$ to $b$ in $G$ through a vertex in $D_{i,\ell}$ that has not been smoothed, we can use that path to form the $ab$ edge. Smooth the remaining 3-edge induced paths in $G$ and delete the vertices from the decision gadgets that were not contracted.

The nails of the resulting minor are exactly $A \cup B \cup R$. There are $m\sqrt{n}$ vertices in $R$ and each one participates in $\sqrt{n}$ variable gadgets. Since each variable gadget becomes a cycle there are $mn$ edges within $R$. The $m$ clause gadgets become bicliques on $2\sqrt{n}$ vertices each and thus contain $mn$ edges in total. Each variable gadget ends up with two edges into each clause gadget, for a total of $2mn$ edges connecting them. In total, this makes $4mn$ edges and $3m\sqrt{n}$ vertices, exactly $\rho$.

**Reverse direction:** We now prove the reverse direction by assuming $\Phi$ is unsatisfiable. Let $H'$ be a 1-STM with density $\rho$. Since $\rho$ is $\Theta(\sqrt{n})$ and the vertices in the induced paths and decision gadgets have degree at most 4, we can assume that none of those vertices appears as a nail in $H'$. Thus, the nail set of $H'$ is a subset of $A \cup B \cup R$. The only paths between these nail candidates that use at most three edges do not contain nail candidates as interior vertices, meaning nail candidates are never smoothed. Let $H(\Phi)$ be the minor constructed by the process described in the forward direction proof for an arbitrary satisfying assignment of $\Phi$. Observe that for fixed $n$ and $m$, the minor $H(\psi)$ is identical for every satisfiable formula $\psi$; let $\mathcal{H}(n, m)$ be that minor. Moreover, every pair of nail candidates that has a 3-edge path between them in $G$ is adjacent in $\mathcal{H}(n, m)$, meaning that $H'$ is a either a subgraph of or identical to $\mathcal{H}(n, m)$.

We now show that no proper induced subgraph of $\mathcal{H}(n, m)$ has density $\rho$. If a graph is $d$-regular and connected, it has edge density $d/2$; deleting part of a degree-regular graph leaves vertices with degree less than $d$, so no proper subgraph reaches that density. Therefore $R$ and each $A_i \cup B_i$ achieve their maximum densities of $\sqrt{n}$ and $\sqrt{n}/2$ only when including the entire subgraph, implying a dense subgraph must contain portions of both vertex and
clause gadgets. The only vehicle for increasing density is to use edges between vertex and clause gadgets, which means we should only include a vertex in $R$ in a subgraph if it also contains its neighbors from $A$ and $B$. Let $G'$ be the subgraph of $G$ induced on an $i \times j$ subgrid of $R$ and all of its neighbors in $A \cup B$. The density within the subset of $A \cup B$ is greatest when those vertices induce $j \times j$ bicliques, so we assume they do. Under this assumption, the density of $G'$ is $4j$ if $j = m$ and $(4j-1)i/3j$ if $j < m$ since the edges that wrap around $R$ cannot be realized. In either case, the density is strictly less than $\rho$ unless $j = m$ and $i = \sqrt{n}$ i.e. exactly $\mathcal{H}(n,m)$.

A decision gadget connected to sequential vertices $v_1, v_2, v_3$ can only create the edge $v_1, v_2$ or the edge $v_2, v_3$, since $d_C$ needs to be smoothed to construct either edge. Consequently, choosing to set some decision gadgets in the same variable gadget to opposite configurations (or neither configuration) creates at least one fewer edge than if they were all set to the same decision. Thus, the configurations of the variable gadgets correspond to some truth assignment to $\Phi$ in the way intended in $H(\Phi)$. This indicates that there is a clause gadget that has no neighbors in a variable gadget that can be used to be smoothed into an edge in the clause gadget. However, because there is one fewer decision gadget than the number of biclique edges in the clause gadgets of $\mathcal{H}(n,m)$, $H'$ cannot realize all possible edges in the biclique and thus there is no 1-STM of density $\rho$.

It stands to prove that the above reduction has the proper implications for a parameterization by treewidth. Using cops and robbers, we can show that $G$ has treewidth $O(\sqrt{n})$ as follows: We permanently station $\sqrt{n}$ cops on the first column of $R$. We use $2\sqrt{n}$ cops to walk through the columns of $R$ sequentially; when a column is completely covered with cops we can explore its corresponding clause gadget with a separate unit of $2\sqrt{n}$ cops. In total, this requires $O(\sqrt{n})$ cops. Although the formula $\Phi$ may have been padded with additional variables, it would only have been enough to increase $\sqrt{n}$ by 2. This means the number of variables in the unpadded instance is still $O(n)$. Thus, if an algorithm parameterized by treewidth $t$ could find a dense 1-STM in time $2^{o(t^2)}n^{O(1)}$, then we could use our reduction to solve CNF-SAT in time $2^{o(n)}n^{O(1)}$, violating ETH. This concludes the proof of Theorem 11.

An immediate consequence is that, unlike Dense $\frac{1}{2}$-STM, Dense 1-STM is still NP-hard when the exact nail set is known.

### 6.6 Conclusion

We showed that finding dense substructures that are just slightly less local than subgraphs is computationally hard, and even a parameterization by treewidth cannot provide very efficient algorithms. This indicates that recognizing classes of bounded expansion is an inherently difficult problem. While our first reduction excludes a subexponential exact algorithm assuming the ETH, we could not exclude an algorithm with a running time of $(2 - \varepsilon)^n n^{O(1)}$. Is such an
algorithm possible for \( r = 1 \), or can one find a tighter reduction that provides a corresponding SETH lower bound? Our second reduction rules out a \( 2^{o(tw^2)}n^{O(1)} \)-algorithm for \( r = 2 \). Is a faster algorithm for \( r = 1 \) possible?

Finally, we ask whether there is a sensible notion of substructures that fit in between \( \frac{1}{2} \)-shallow topological minors and subgraphs for which we can find the densest occurrence in polynomial time.
CHAPTER

7

LOCAL SEARCH

Summary

To further the existing algorithmic research for bounded expansion graph classes and answer Research Question 4, we focus on FPT algorithms to be used in local search, a common metaheuristic for optimization problems. Specifically, the algorithms find valid $k$-exchanges for feasible solutions to various optimization problems. In this context, a $k$-exchange is a replacement of vertices or edges from inside a feasible solution $S$ with vertices or edges from outside $S$ such that creates a new feasible solution of better quality, i.e., size or weight, and the number of elements removed from or added to $S$ is no more than $k$. Using dynamic programming techniques similar to those in CONCUSS, we create algorithms for Minimum Vertex Cover $k$-Exchange, Maximum Independent Set $k$-Exchange, and Minimum Maximal Matching $k$-Exchange that are both efficient and able to be implemented for use in practice. Furthermore, we show how each algorithm can be adapted to accommodate weighted graphs as well. This work was done in collaboration with Brian Lavallee [55].

7.1 Introduction

While the algorithmic pipeline implemented in CONCUSS counts subgraph isomorphisms (see Section 4.4), not all of its components are specifically tailored for that problem. Finding
a \( p \)-centered coloring and creating treedepth decompositions of low depth in the \textsc{Color} and \textsc{Decompose} modules, respectively, are generic tools that allow the structural sparsity of bounded expansion classes to be algorithmically exploited. Thus, the workflow of \textsc{Color}, \textsc{Decompose}, \textsc{Compute}, and \textsc{Combine} can be seen as a collection of independent modules in which subroutines can be interchanged in one module without necessarily affecting the others. To answer Research Question 4, we utilized this modularity and developed FPT algorithms for the \textsc{Compute} and \textsc{Combine} stages that solve different hard problems in polynomial time. Our algorithms are designed with practicality and implementability in mind and thus avoid the use of the “heavy machinery” found in Proposition 9 and other meta-theorems.

Our focus is on algorithms for \textit{first-order} (FO) properties, meaning the property can be expressed in a logical sentence in which quantification is restricted to vertices (as opposed to sets of vertices). Proposition 9 guarantees that all such problems are FPT in classes of bounded expansion; historically, the existence of an FPT algorithm for problem \( \Pi \) via a meta-theorem often indicates there is an FPT algorithm for \( \Pi \) that does not require the meta-theorem. Because the complexity of the meta-theorem depends exponentially on the number of variables in the FO sentence, a number of FO properties are of the form “having a vertex cover of size \( k \)”, “having a maximal matching of size \( k \)”, etc. However, many such problems already have practical FPT algorithms for general graphs [1, 23] and/or describe graph parameters that are unlikely to be of constant size in large sparse graphs.

Instead, we focus our attention on \textit{local search}, a metaheuristic for solving generic optimization problems that takes a current feasible solution and makes many small perturbations to it to form a “neighborhood”. If any solution in the neighborhood is better than the current one, the best solution from the neighborhood is taken and the process repeats, terminating once a neighborhood contains no improvement. As a metaheuristic, local search can be applied to many optimization problems and therefore has received considerable attention in the literature [51, 61, 95, 96].

For a typical graph optimization problem, neighboring solutions can be formed by replacing vertices or edges in the solution with vertices or edges outside the solution. One method of finding potential replacements is using a \textit{k-exchange} [59]. Given a graph optimization problem \( \Pi \) and integer \( k \), we define the problem \( \Pi k\text{-Exchange} \) as follows.

<table>
<thead>
<tr>
<th>( \Pi k\text{-Exchange} )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> A graph ( G ) and a feasible solution ( S \subseteq G ) to ( \Pi ).</td>
</tr>
<tr>
<td><strong>Output:</strong> Sets ( I \subseteq G \setminus S ) and ( R \subseteq S ) such that ((S \setminus R) \cup I ) is a feasible solution to ( \Pi ) and (</td>
</tr>
</tbody>
</table>
It is proven in [83] that if verifying whether a set $S$ is a feasible solution to $\Pi$ can be expressed in a FO sentence, then the problem $\Pi$ k-Exchange is expressible in a FO sentence whose size is bounded by the size of the neighborhood (as opposed to the size of the overall solution). While larger values of $k$ may help the algorithm avoid local optima, k-exchanges naively require enumerating $O(n^{f(k)})$ sets in order to check the neighborhood of a single solution. Our goal is use the bounded expansion structure to reduce the complexity of certain k-exchange problems from XP down to FPT. Specifically, we devise a dynamic programming scheme for practical FPT k-exchange algorithms for the following problems.

**Theorem 12** (Lavallee, O’Brien, Sullivan [55]). The problems

- **Minimum Vertex Cover k-Exchange**
- **Maximum Independent Set k-Exchange**
- **Minimum Maximal Matching k-Exchange**

can be solved in $O^*(n^2)$ time on graphs belonging to classes of bounded expansion.

Our algorithms also extend to weighted variants of these problems in which each vertex or edge is assigned a real-valued weight and the optimization problem seeks to minimize/maximize the sum of weights in the solution rather than the size of the solution.

**Weighted $\Pi$ k-Exchange**

**Input:** A graph $G$, weight function $w$ and a feasible solution $S \subseteq G$ to $\Pi$.

**Output:** Sets $I \subseteq G \setminus S$ and $R \subseteq S$ such that $S' = (S \setminus R) \cup I$ is a feasible solution to $\Pi$, $|I|, |R| \leq k$, and

$$\sum_{y \in S'} w(y) \begin{cases} < \sum_{x \in S} w(x) & \text{if $\Pi$ is a minimization problem} \\ > \sum_{x \in S} w(x) & \text{if $\Pi$ is a maximization problem.} \end{cases}$$

**Corollary 2.** The problems **Minimum Weighted Vertex Cover k-Exchange**, **Maximum Weighted Independent Set k-Exchange**, and **Minimum Weighted Maximal Matching k-Exchange** can be solved in $O^*(n^2)$ time on graphs belonging to classes of bounded expansion.
7.2 Background

7.2.1 Notation

For a vertex \( v \) in treedepth decomposition \( T \), let \( v_1, \ldots, v_\ell \) be the children of \( v \). Recalling that \( T_v \) denotes the subtree of \( T \) rooted at \( v \), we will use the notation \( T_{\leq v_j} = T_{v_1} \cup \ldots \cup T_{v_j} \). Let \( A(v) = \{ a_1, \ldots, a_{d-1} \} \) be the set of ancestors of \( v \) with respect to \( T \) such that \( a_1 = \text{root}(T) \) and \( a_{d-1} = \text{parent}(v) \), and \( A[v] = A(v) \cup \{ v \} \). We note that \( A[v] = A(v_1) = \cdots = A(v_\ell) \). We say \( \phi : U \rightarrow \Gamma \) is a labeling function of vertex set \( U \) with respect to some alphabet \( \Gamma \). With a slight abuse of notation, we let \( \phi^{-1}(\gamma) = \{ u : \phi(u) = \gamma \} \) for all \( \gamma \in \Gamma \). Let \( \epsilon : \emptyset \rightarrow \emptyset \) denote the null function.

7.2.2 Problem Definitions

A vertex cover is a set of vertices \( S \) such that for each edge \( uv \in E(G) \), \( \{ u, v \} \cap S \neq \emptyset \). An independent set is a set of vertices \( S \) such that \( \{ u, v \} \subseteq S \) implies \( uv \notin E(G) \). A maximal matching is a set of edges \( S \) such that each vertex belongs to at most one edge in \( S \) and the graph \( G[V(G) \setminus S] \) is an independent set. If \( uv \) is an edge in a maximal matching \( S \), we say \( u \) is matched with \( v \) in \( S \); if \( u \) is not the endpoint of any edge in \( S \), \( u \) is unmatched in \( S \).

7.3 Algorithmic Strategy

Our algorithms will follow the same general strategy as the SUBGRAPH ISOMORPHISM COUNTING algorithm [27] used in CONCUSS (Section 4.4). We compute a \( p \)-centered coloring of \( G \) for a value of \( p \) that guarantees that every \( k \)-exchange will induce a subgraph of low treedepth. From there we can enumerate subsets of \( x \) colors, where \( x \) is the maximum number of vertices that can participate in a \( k \)-exchange, and create a treedepth decomposition of each component of the subgraph induced on \( x \) colors.

Within a specific treedepth decomposition \( T \), we store dynamic programming tables at each vertex \( v \) that tracks partial solutions “above” and “below” \( v \), i.e., in \( A(v) \) and \( T_v \), respectively. The partial solutions for the ancestors of \( v \) are represented as ancestor labeling functions that describe the role each ancestor plays in the partial solution, while in the subtree rooted at \( v \) we simply count the number of exchange vertices without tracking which specific vertices participate. The dynamic programming tables are initialized at the leaves of the treedepth decomposition and then used to derive additional tables using two operations: joins and forgets. The join operation combines tables of “siblings” (vertices with the same parent) until there is a single table for all children of a vertex \( v \). Because siblings have the same ancestors and their respective subtrees are disjoint, we can safely join table entries whose ancestor labeling
functions prescribe compatible roles for the ancestors. Given the unified table for \( v \)'s children, the forget operation creates a table for \( v \) and removes it from the ancestor labeling. Pseudocode for this dynamic programming strategy is detailed in Algorithm 8. In the event that \( \mathcal{T} \) is disconnected, we can simulate a connected graph by placing a dummy root\(^1\) as the parent of the roots of all components of \( \mathcal{T} \); this is equivalent to performing the join operation on all tables at the roots of components of \( \mathcal{T} \).

### Algorithm 8 \( k \)-exchange(\( \mathcal{T} \))

**Input:** Treedepth decomposition \( \mathcal{T} \)

**Output:** Entry at root table corresponding to (non)-existence of a \( k \)-exchange

1. \( T[X] \leftarrow \) dynamic programming table at vertex set \( X \)
2. \( \sigma \leftarrow \) post-order traversal of \( \mathcal{T} \)
3. **for all** \( v \in \sigma \) **do**
4.   *if* \( v \) is a leaf *then*
5.     initialize a dummy table for the "children" of a leaf
6.     **end if**
7.   **for all** \( j \in \{2, \ldots, \ell\} \) **do**
8.     \( T[v_1, \ldots, v_j] \leftarrow \text{join}(T[v_1, \ldots, v_{j-1}], T[v_j]) \)
9.   **end for**
10. \( T[v] \leftarrow \text{forget}(T[v_1, \ldots, v_{\ell}]) \)
11. **end for**
12. **return** appropriate entry from \( T[\text{root}(\mathcal{T})] \)

By the definition of \( x \), any \( k \)-exchange involves vertices that use at most \( x \) different colors. Thus, there is a \( k \)-exchange if and only if there is a set of \( x \) colors \( C \) such that the dynamic programming algorithm identifies a \( k \)-exchange in the treedepth decomposition of the subgraph induced on vertices with colors \( C \). In other words, there is no work to be done in combining subsolutions from separate treedepth decompositions; in fact, depending on whether we would like to examine all \( k \)-exchanges or just accept any one, we could potentially stop checking additional color sets once the first valid \( k \)-exchange is found.

### 7.4 Vertex Cover

We note that an algorithm for Minimum Vertex Cover \( k \)-Exchange is also an algorithm for Maximum Independent Set \( k \)-Exchange since \( S \) is an independent set of \( G \) iff \( V(G) \setminus S \) is a vertex cover of \( G \). Since the vertex cover \( k \)-exchange involves removing at most \( k \) vertices from

\(^1\)This vertex receives no label by the labeling functions and is not considered to be part of the graph when determining the validity of a (partial) solution.
the solution and including at most $k - 1$ vertices from outside the solution, it is sufficient to compute a $2k$-centered coloring of $G$ and enumerate color sets of size $2k - 1$.

### 7.4.1 Algorithm Description

For **Minimum Vertex Cover k-Exchange**, we define the ancestor labeling function $\phi : A(v) \rightarrow \{I, R, *\}$ such that $\phi^{-1}(I)$ is the set of vertices to be added to $S$, $\phi^{-1}(R)$ is the set of vertices removed from $S$, and $\phi^{-1}(*)$ is the set of vertices whose (non-)membership in $S$ is unchanged. We say $\phi$ is *valid* if $\phi^{-1}(R) \subseteq S$ and $\phi^{-1}(I) \cap S = \emptyset$; we only consider valid labelings for the rest of this section. Let $\phi \ominus v$ be the labeling function $\phi'$ with the property that $\phi'(v) = \emptyset$ and $\phi'(w) = \phi(w)$ for $w \neq v$. In other words, $\phi \ominus v$ removes $v$ from the domain of $\phi$.

To initialize the tables for each leaf $v$, we create a dummy table $T[v_1, \ldots, v_j][\phi, 0, 0]$ for the “children” of $v$ whose value is 1 if $(S \cup \phi^{-1}(I) \setminus \phi^{-1}(R)) \cap ((G \setminus T) \cup A[v])$ is a vertex cover of $(G \setminus T) \cup A[v]$ and 0 otherwise. We define forget$(T[v_1, \ldots, v_j])$ to be the table $T[v]$ such that

$$T[v][\phi, i, r] = \bigvee_{\phi' \ominus v} T[v_1, \ldots, v_j][\phi', i', r'],$$

where

$$(i', r') =\begin{cases}
(i - 1, r) & \text{if } \phi'(v) = I \\
(i, r - 1) & \text{if } \phi'(v) = R \\
(i, r) & \text{otherwise.}
\end{cases}$$

The operation join$(T[v_1, \ldots, v_{j-1}], T[v_j])$ creates a table $T[v_1, \ldots, v_j]$ so that

$$T[v_1, \ldots, v_j][\phi, i, r] = \bigvee_{\substack{i = i_1 + i_2 \\ r = r_1 + r_2}} T[v_1, \ldots, v_{j-1}][\phi, i_1, r_1] \land T[v_j][\phi, i_2, r_2].$$

### 7.4.2 Correctness

We prove that the forget and join operations preserve the invariant that $T[v_1, \ldots, v_j][\phi, i, r] = 1$ iff there is a vertex cover $S'$ of $(G \setminus T) \cup A[v] \cup \mathcal{T}_{\leq v_j}$ meeting the following conditions:

1. $S' \cap (G \setminus T) = S \cap (G \setminus T)$
2. $S' \cap A[v] = ((S \cap A[v]) \cup \phi^{-1}(I)) \setminus \phi^{-1}(R)$
3. $|S' \cap \mathcal{T}_{\leq v_j} \setminus S| = i$
4. $|S \cap \mathcal{T}_{\leq v_j} \setminus S'| = r$. 

96
Lemma 19. If every table entry appearing on the right hand side of the forget and join expressions satisfies the invariant, then the left hand side also satisfies the invariant.

Proof. Forget: First assume table entry $T[v][\phi, i, r] = 1$. In order for the right hand side of the expression to evaluate to 1, there is a table entry $T[v_1, \ldots, v_\ell][\phi', i', r'] = 1$; let $S'$ be its corresponding vertex cover that satisfies (1)-(4). Since $A(v) \cup T_v = A[v] \cup T_{\leq v}$, $S'$ is also a solution that satisfies conditions (1) and (2) with respect to $\phi = \phi' \oplus v$. If $\phi'(v) = \star$, then $S'$ immediately satisfies (3) and (4) for $i = i'$ and $r = r'$. Otherwise, $\phi'(v) = I$ or $\phi'(v) = R$ and setting $i = i' + 1[\phi'(v) = I]$ and $r = r' - 1[\phi'(v) = R]$ accounts for the addition of one vertex in $S' \Delta S$ from the ancestors to the subtree.

We then suppose that there is a vertex cover $S'$ of $(G \setminus T) \cup A[v] \cup T_v$ that satisfies (1)-(4). If $v \notin S \Delta S'$ then with respect to the labeling $\phi'$ that is identical to $\phi$ except $\phi'(v) = \star$, $S'$ is a vertex cover of $(G \setminus T) \cup A[v] \cup T_{\leq v}$ that satisfies (1)-(4). If $v \in S' \setminus S$, then $S' \cap (T_{\leq v}) \setminus S$ contains one fewer vertex than $S' \cap T_v \setminus S$ but $S \cap T_{\leq v} \setminus S' = S \cap T_v \setminus S'$. Since we see analogous results when $v \in S \setminus S'$, by decreasing $i$ or $r$ by 1 and adding the appropriate label to $v$, we have a triple $(\phi', i', r')$ whose corresponding entry in the table $T[v_1, \ldots, v_\ell]$ must be 1. Thus the forget operation preserves the invariant.

Join: In the join step, first assume there are two pairs $(i_1, r_1)$ and $(i_2, r_2)$ such that $T[v_1, \ldots, v_{\ell-1}][\phi, i_1, r_1] = T[v_\ell][\phi, i_2, r_2] = 1$. Let $S_1'$ and $S_2'$ be the vertex covers associated with the two table entries, respectively. We prove that $S' = S_1' \cup S_2'$ is the vertex cover satisfying (1)-(4) with respect to the labeling $\phi$. Because $S_1'$ and $S_2'$ satisfy (1)-(4), $(S_1' \cup S_2') \cap (G \setminus T) = S \cap (G \setminus T)$, so all edges outside of $T$ must be covered and $S'$ satisfies (1). Since $S_1'$ and $S_2'$ are both consistent with $\phi$ and the ancestors of all children of $v$ are identical, (2) is satisfied and no edge with $A(v)$ as an endpoint can be uncovered. Outside of $A[v]$, $S_1'$ and $S_2'$ share no common vertices or edges, which means that $S'$ covers all edges with an endpoint in $T_{\leq j}$. Thus, $S'$ is indeed a vertex cover of $(G \setminus T) \cup A[v] \cup T_{\leq v}$. Furthermore, the fact that $T_{\leq v_{j-1}}$ is disjoint from $T_v$ indicates that $i = i_1 + i_2$ and $r = r_1 + r_2$ are the integers that satisfy (3) and (4).

In the opposite direction, we assume $S'$ is a vertex cover of $(G \setminus T) \cup A[v] \cup T_{\leq v}$ satisfying (1)-(4). Since no edges have one endpoint in $T_v$ and the other in $T_{\leq v_{j-1}}$, $S' \cap (A(v_j) \cup T_{v_j} \cup (G \setminus T))$ is a vertex cover of $(G \setminus T) \cup A(v_j) \cup T_{v_j}$ satisfying (1) and (2). By the same reasoning we can prove $S'$ has a similar intersection with $(G \setminus T) \cup A[v] \cup T_{\leq v_{j-1}}$. Thus, there are two table entries $T[v_1, \ldots, v_{\ell-1}][\phi, i_1, r_1] = T[v_\ell][\phi, i_2, r_2] = 1$ for some pair $(i_1, r_1)$ and $(i_2, r_2)$. Since the $r$ elements of $S \cap T_{\leq v_{j-1}} \setminus S'$ must be contained in $T_{\leq v_{j-1}}$, if $z$ of those elements are contained in $T_{v_{j-1}}$, then $r - z$ of them are contained in the other subtrees. Therefore, $r = r_1 + r_2$ and the join operation’s enumeration over all such choices of $r_1 + r_2$ will eventually find the right one. This argument can be extended to show that $i = i_1 + i_2$, proving that the invariant holds under joins. \qed

97
Using Lemma 19 as an inductive hypothesis, we now show that the root table has a set of entries that accurately report the (non-)existence of a $k$-exchange in $T$.

**Lemma 20.** $T[\text{root}(T)][e, i, r] = 1$ if and only if there exist $I, R \subseteq V(T)$ such that $(S \cup R) \setminus I$ is a vertex cover of $G$, $I \cap R = \emptyset$, $|I| = i$, and $|R| = r$.

**Proof.** We first observe that the dummy tables initialized for the “children” of the leaves trivially satisfy (1)-(4). When combined with Lemma 19, this implies that $T[\text{root}(T)][e, i, r] = 1$ if and only if there is a vertex cover $S'$ of $G$ that satisfies (1)-(4).

To prove the forward direction, let $S'$ be the vertex cover associated with $T[\text{root}(T)][e, i, r]$ satisfying (1)-(4). By setting $I = S' \setminus S$ and $R = S \setminus S'$, (3) and (4) indicate that the two sets have size $i$ and $r$, respectively. Furthermore, $I$ and $R$ are disjoint by construction and subsets of $T$ by (1), which completes the proof.

In the reverse direction, $S' = (S \cup R) \setminus I$ is a vertex cover of $G$ such that $|S' \setminus S| = i$ and $|S \setminus S'| = r$. Moreover, since $I, R \subseteq V(T)$, $S'$ only differs from $S$ inside of $T$. Thus $S'$ satisfies (1)-(4) for $\phi = e$ and $T[\text{root}(T)][e, i, r] = 1$.

Having proven its correctness, we now show our algorithm runs in FPT time.

**Lemma 21.** If $d$ is the depth of $T$, the above algorithm finds in $O(2^d k^4 n)$ time a solution to Minimum Vertex Cover $k$-Exchange that lies entirely in $T$ or determines one does not exist.

**Proof.** Because the sets $I, R$ defined for an entry of $T[\text{root}(T)]$ in Lemma 20 form a $k$-exchange exactly when $1 \leq r \leq k$ and $i < r$, there are only $O(k^2)$ entries that need to be tracked at the root. Every table entry $T[\cdot][\cdot, i, r]$ can be computed using only the values from table entries $T[\cdot][\cdot, i', r']$ such that $i' \leq i$ and $r' \leq r$, which means that for a fixed vertex set and labeling function, there are at most $O(k^2)$ relevant values of $i$ and $r$.

For any vertex in $T$ there are $2^{|A(v)|}$ valid labelings, since each vertex can take on exactly one of the two labels $I$ and $R$ and checking whether a labeling implies a vertex cover takes time linear in the degree sum of all vertices in $A(v)$. Thus, initialization of all leaves of $T$ takes at most $O(2^d \cdot n)$ time. Since there are exactly two labelings $\phi'$ such that $\phi = \phi' \ominus v$, each forget requires constant time and each join requires $O(i \cdot r)$ time. In total, this leads to a complexity of $O(2^d k^4 n)$.

### 7.4.3 Weighted Variant

To extend our algorithm to handle a vertex weight function $w$, we use an alternate dynamic programming table $W$ whose entries record the maximum decrease in weight within subtrees rooted at particular vertices. Specifically, we set $W[v_1, \ldots, v_j][\phi, i, r] = x$ if and only if $x$ is the
smallest value such that there is a vertex cover \( S' \) of \((G \setminus T) \cup A[v] \cup T_{\leq v_j} \) that satisfies (1)-(4) and
\[
\sum_{u \in U} s(u) \cdot w(u) = x
\]
where \( s(u) \) is \(-1\) if \( u \in S \) and \( 1 \) otherwise and \( U = (S \Delta S') \cap (T_{\leq v_j}) \). To compute this, we initialize \( W \) with dummy tables at the children of the leaves with value 0 if the corresponding entry in \( T \) is 1 and \((|G| + 1) \cdot \max_{u \in V(G)} w(u) \) otherwise. We let forget \((W[v_1, \ldots, v_{\ell}])\) be defined as
\[
W[v][\phi, i, r] = \min_{\phi' \in \phi \circ \phi} (W[v_1, \ldots, v_{\ell}][\phi', i', r'] + y),
\]
where \( i' \) and \( r' \) are defined as in the unweighted variant and
\[
y = \begin{cases} 
  w(v) & \text{if } \phi'(v) = I \\
  -w(v) & \text{if } \phi'(v) = R \\
  0 & \text{otherwise}. 
\end{cases}
\]
We define join \((W[v_1, \ldots, v_{j-1}], W[v_j])\) so that
\[
W[v_1, \ldots, v_{j}][\phi, i, r] = \min_{i = i_1 + i_2 \atop r = r_1 + r_2} W[v_1, \ldots, v_{j-1}][\phi, i_1, r_1] + W[v_j][\phi, i_2, r_2].
\]
In other words, when forgetting we increase the solution weight when \( v \in S' \setminus S \), decrease it when \( v \in S \setminus S' \), and do nothing otherwise. At the root table, any entry \( W[\text{root}(T)][\epsilon, i, r] \) with a negative value corresponds to a \( k\)-exchange that decreases the solution weight. The proof of correctness and running time is analogous to that of the unweighted version. For Maximum Weighted Independent Set \( k\)-Exchange, the problem remains largely unchanged except we initialize the non-zero table entries to \(-(|G| + 1) \cdot \max_{u \in V(G)} w(u)\), take the maximum value in the join and forget steps, and return entries that are positive at the root.

### 7.5 Maximal Matching

The algorithm for Minimum Maximal Matching \( k\)-Exchange operates similarly to the algorithm for Minimum Vertex Cover \( k\)-Exchange. However, because a maximal matching is a set of edges while a vertex cover is a set of vertices, in order to perform dynamic programming over the vertices of the treedepth decomposition the labeling function must now encode edge assignments rather than vertex assignments. Note that if edge \( uv \) is removed from the solution, then \( u \) or \( v \) must be the endpoint of an edge included in the solution or else \( u \) and \( v \) form an adjacent unmatched pair. Thus at most \( \frac{3}{2}(2k - 1) = 3k \) vertices can participate
in the $k$-exchange and it is sufficient to consider color sets of size $3k$ in a $(3k + 1)$-centered coloring.

### 7.5.1 Algorithm Description

For **Minimum Maximal Matching $k$-Exchange**, we define the labeling function

$$
\phi : A(v) \rightarrow \{1, 2, \ldots, d - 1, \infty, *, O, B\}
$$

where $d = \text{depth}(v)$. The labeling of each $u \in A(v)$ will be used to describe its role in $S'$: integer label $j$ indicates $ua_j \in S'$; $\infty$ indicates $uw \in S'$ for some $w$ in $T_v$; $*$ indicates $u$ is unmatched in $S'$; $O$ indicates $uw \in S'$ for some $w \not\in T$; $B$ indicates $uw \in S'$ for $w \in T \setminus (A(v) \cup T_v)'$. To enforce these properties, we say $\phi$ is **valid** if each of the following holds:

- For all integers $1 \leq \delta_1, \delta_2 \leq d - 1$, $\phi(a_{\delta_1}) = \delta_2 \iff \phi(a_{\delta_2}) = \delta_1$.
- $\phi(a_j) \neq j$ for all $j$.
- If $\phi(a_{\delta_1}) = \delta_2$, then $a_{\delta_1}a_{\delta_2} \in E(G)$.
- $\phi(u) = O$ iff there is an edge $uw \in S$ such that $w \not\in T$.
- $\phi^{-1}(*)$ is an independent set.

In the rest of this section we only consider valid labelings. Given two labelings $\phi_1, \phi_2$ of $A(v)$, we say $\phi_1$ and $\phi_2$ are **compatible** if

- $\phi_1^{-1}(\gamma) = \phi_2^{-1}(\gamma)$ for all $\gamma \not\in \{B, \infty\}$ and
- $\phi_1^{-1}(\infty) \cap \phi_2^{-1}(\infty) = \emptyset$.

Thus, compatible labelings must only differ on vertices that are assigned $\infty$ in one and $B$ in the other. For compatible labelings $\phi_1, \phi_2$, we define the labeling $\phi = \phi_1 \oplus \phi_2$ such that $\phi(u) = \phi_2(u)$ if $\phi_2(u) \in \{1, \ldots, d, \infty\}$ and $\phi(u) = \phi_1(u)$ otherwise. If $\phi$ is a labeling of $A(v)$ such that $\phi(v) \neq B$, we define $\phi \ominus v$ to be the labeling function such that

$$
\phi \ominus v(u) = \begin{cases} 
\emptyset & \text{if } u = v \\
\infty & \text{if } \phi(u) = d - 1 \\
\phi(u) & \text{otherwise}
\end{cases}
$$

for all $u \in A(v)$. Note that $\phi \ominus v$ is undefined when $\phi(v) = B$. 

100
To initialize the tables for each leaf \( v \), we create a dummy table \( T[v_1, \ldots, v_j][\phi, 0, 0] \) for the “children” of \( v \) whose value is 1 if \( \phi^{-1}(\infty) = \emptyset \) and 0 otherwise. We define forget(\( T[v_1, \ldots, v_j] \)) to be the table \( T[v] \) such that

\[
T[v][\phi, i, r] = \bigvee_{\phi' \in \mathcal{V}} T[v_1, \ldots, v_j][\phi', i', r'],
\]

where \( i \) and \( r \) satisfy

\[
i' = i - 1[\phi'(v) \in \{1, \ldots, d - 1\} \land va_{\phi'(v)} \notin S]
\]

\[
r' = r - 1[\exists vw \in S \land (\phi'(v) = * \land a_{\phi'(v)} \neq w)].
\]

The operation join(\( T[v_1, \ldots, v_{j-1}], T[v_j] \)) creates a table \( T[v_1, \ldots, v_j] \) so that

\[
T[v_1, \ldots, v_j][\phi, i, r] = \bigvee_{\phi, i_1, r_1} T[v_1, \ldots, v_{j-1}][\phi, i_1, r_1] \land T[v_j][\phi, i_2, r_2].
\]

7.5.2 Correctness

We prove that the forget and join operations preserve the invariant that \( T[v_1, \ldots, v_j][\phi, i, r] = 1 \) iff there is a maximal matching \( S' \) of \( (G \setminus \mathcal{T}) \cup (A[v] \setminus \phi^{-1}(B)) \cup \mathcal{T}_{\leq v_j} \) meeting the following conditions:

1. For any \( uw \in S \) such that \( \{u, w\} \cap (G \setminus \mathcal{T}) \neq \emptyset \), then \( uw \in S' \) and \( \phi(u) = O \) if \( u \in A[v] \).
2. For integers \( 1 \leq \delta_1, \delta_2 \leq d - 1 \), \( \phi(a_{\delta_i}) = \delta_2 \) iff \( a_{\delta_i} a_{\delta_2} \in S' \).
3. Each vertex in \( \phi^{-1}(\ast) \) is unmatched in \( S' \).
4. \( u \in \phi^{-1}(\infty) \) iff there exists a \( w \in \mathcal{T}_{\leq v_j} \) such that \( uw \in S' \).
5. Exactly \( i \) edges in \( (S' \setminus S) \cap \mathcal{T} \) have at least one endpoint in \( \mathcal{T}_{\leq v_j} \).
6. Exactly \( r \) edges in \( (S \setminus S') \cap \mathcal{T} \) have at least one endpoint in \( \mathcal{T}_{\leq v_j} \).

**Lemma 22.** If every table entry appearing on the right hand side of the forget and join expressions satisfies the invariant, then the left hand side also satisfies the invariant.

**Proof.** Forget: We first prove the “\( i' \)” part of the invariant. The forget expression is a large disjunction, which implies that if \( T[v][\phi, i, r] = 1 \), then \( T[v_1, \ldots, v_j][\phi', i', r'] = 1 \) for some \( (\phi', i', r') \) as defined in the forget expression. Let \( S' \) be the maximal matching corresponding to \( T[v_1, \ldots, v_j][\phi', i', r'] \) that satisfies (1)-(6). We only need consider the case where \( \phi'(v) \neq B \).
since \( \phi' \ominus v \) is only defined in this case. Because \( A[v] \cup T_{\leq v} = A(v) \cup T_v \), \( S' \) is also a maximal matching in \((G \setminus T) \cup (A[v] \setminus \phi^{-1}(B)) \cup T_{\leq v} \). By the definition of valid labelings and the fact that \( \phi'^{-1}(\gamma) = \phi^{-1}(\gamma) \) for \( \gamma \in \{1, \ldots, d-1, O, *, 0\} \), (1)-(3) hold for \( \phi \) with respect to \( S' \). Since \( v \in A[v] \) but \( v \notin A(v) \), if \( \phi'(v) = j \) for some \( 1 \leq j \leq d-1 \), then \( \phi(a_j) = \infty \) and (4) holds. The subset of \( S' \setminus S \) with at least one endpoint in \( T_v \) but no endpoints in \( \bigcup_{i=1}^d T_{v_i} \) is either empty or is exactly the edge \( va \phi'(v) \). Selecting \( i = i' \) or \( i = i' + 1 \) satisfies (5) in the former and latter cases, respectively, which matches the definition of the forget step. Likewise, if \( S \setminus S' \) includes an edge \( vw \), then \( v \) is either matched with a different vertex or unmatched in \( S' \), meaning (6) is satisfied.

To show the “only if” part, let \( S' \) be a maximal matching of \((G \setminus T) \cup (A[v] \setminus \phi^{-1}(B)) \cup T_v \) satisfying (1)-(6). We claim there is a labeling function \( \phi' \) of \( A[v] \) such that \( \phi = \phi' \ominus v \) and \( S' \) is a maximal matching of \((G \setminus T) \cup (A[v] \setminus \phi'^{-1}(B)) \cup T_{\leq v} \) that satisfies (1)-(6). As long as \( \phi'^{-1}(B) = \phi^{-1}(B) \), \( S' \) is immediately a maximal matching of the desired subgraph; we will show below that there must be a \( \phi' \) with this property. Since \( v \in T_v \), \( v \) is either matched in \( S' \) or all of its neighbors in \((G \setminus T) \cup (A[v] \setminus \phi^{-1}(B)) \cup T_v \) are matched in \( S' \). Whenever \( v \) is matched with a vertex \( a_j \), then defining \( \phi' \) such that \( \phi'(v) = j \), \( \phi'(a_j) = d \), and \( \phi'(u) = \phi(u) \) for all \( u \notin \{v, a_j\} \) satisfies (1)-(4). Otherwise, there is a unique label\(^2 \lambda \in \{O, \infty\} \) such that \( \phi'(v) = \lambda \) and \( \phi'(u) = \phi(u) \) for \( u \neq v \) satisfies (1)-(4). Whenever \( v \) is unmatched in \( S' \), then no vertex in \( \phi'^{-1}(\ast) \) is adjacent to \( v \). Thus, we can set \( \phi'(v) = \ast \) and \( \phi'(u) = \phi(u) \) for \( u \neq v \) to satisfy (1)-(4). In any case \( \phi'(v) \neq B \) and \( \phi = \phi' \ominus v \). Since \( T_v = \{v\} \cup T_{\leq v} \) and every vertex appears in at most one edge in a maximal matching, there are at least \( i-1 \) edges in \((S' \setminus S) \cap T \) with an endpoint in \( T_{\leq v} \). If the \( r \)th edge does not have an endpoint in \( T_{\leq v} \), then it is of the form \( vw \) for some \( w \in A(v) \) and thus \( \phi'(v) \in \{1, \ldots, d-1\} \). Likewise, there are at least \( r-1 \) edges in \((S \setminus S') \cap T \) with an endpoint in \( T_{\leq v} \) and if the \( r \)th edge \( vw \in S \) lies in \( A[v] \), then either \( v \) is unmatched in \( S' \) or matched with a vertex different from \( w \). Therefore for some \( \phi' \) such that \( \phi = \phi' \ominus v \), there is a corresponding \( i' \) and \( r' \) as defined in the forget expression such that \( S' \) is a maximal matching satisfying (1)-(6). Because \( T[v][\phi, i, r] \) is the disjunction over all choices for \( \phi' \), \( T[v][\phi, i, r] = 1 \).

**Join:** We first prove the “if” direction. If \( T[v_1, \ldots, v_j][\phi, i, r] = 1 \), there are table entries \( T[v_1, \ldots, v_{j-1}][\phi_1, i_1, r_1] = T[v_j][\phi_2, i_2, r_2] = 1 \) such that \( \phi = \phi_1 \ominus \phi_2 \), \( i = i_1 + i_2 \), and \( r = r_1 + r_2 \). Let \( S'_1 \) and \( S'_2 \) be the maximal matchings associated with these entries. Because \( \phi_1 \) and \( \phi_2 \) are compatible and their common vertices are all in \( A[v] \), every vertex in \( A[v] \) that is matched in \( S_1 \cup S_2 \) is either matched in both \( S'_1 \) and \( S'_2 \) with the same vertex in \( A[v] \) or unmatched in one of \( S'_1 \) and \( S'_2 \). Moreover, because \( \phi'^{-1}(B) = \phi_1^{-1}(B) \cap \phi_2^{-1}(B) \), \( S' = S'_1 \cup S'_2 \) is a maximal matching of \((G \setminus T) \cup (A[v] \setminus \phi^{-1}(B)) \cup T_{\leq v} \). This overlap also implies that \( S' \) satisfies (1)-(4).

---

\(^2\)The choice depends on whether \( v \) is matched with a vertex outside of \( T \).
we arrive at the conclusion that with respect to some integers \( \phi \), which 

The dummy entries for each table initialized at the “children” of the leaves satisfy (1)-(3). Proof. By the definition of valid labelings and trivially satisfy (4)-(6). Thus, by applying Lemma 22 we again prove the root table provides the exact information to decide whether \( T \) contains a k-exchange.

Lemma 23. \( T[\text{root}(T)][e,i,r] = 1 \) iff there exist \( I, R \subseteq E(T) \) such that \((S \cup R) \setminus I \) is a maximal matching of \( G \), \( I \cap R = \emptyset \), \( |I| = i \), and \( |R| = r \).

Proof. The dummy entries for each table initialized at the “children” of the leaves satisfy (1)-(3) by the definition of valid labelings and trivially satisfy (4)-(6). Thus, by applying Lemma 22 we arrive at the conclusion that \( T[\text{root}(T)][e,i,r] = 1 \) iff there is a maximal matching \( S' \) of \( G \) satisfying (1)-(6).

If \( T[\text{root}(T)][e,i,r] = 1 \), let \( I = S' \setminus S \) and \( R = S \setminus S' \). Since \( S' \) satisfies (5) and (6), \( I \) and \( R \) fit the requirements of the lemma.

If there exist sets \( I \) and \( R \) as described in the lemma statement, then \( S' = (S \cup R) \setminus I \) is a maximal matching of \( G \). Since both endpoints of each edge in \( I \) and \( R \) lie in \( T \), \( S' \) satisfies (5) and (6). The null function trivially satisfies (1)-(4) regardless of \( S' \), meaning \( T[\text{root}(T)][e,i,r] = 1 \).

Finally, we prove the run time is in FPT.
Lemma 24. If \( d \) is the depth of \( \mathcal{T} \), the above algorithm finds in \( O((2d + 6)^d \cdot k^4 \cdot n) \) time a solution to \( \text{Minimum Maximal Matching} \) \( k \)-Exchange that lies entirely in \( \mathcal{T} \) or determines one does not exist.

Proof. For the same reasons as presented in Lemma 21, there are at most \( O(k^2) \) relevant values of \( i \) and \( r \) for each labeling function and vertex set.

For any vertex in \( \mathcal{T} \) there are \( \binom{|A(v)| + 3}{|A(v)|} \) valid labelings, since there are \( |A(v)| + 3 \) values in the domain of \( \phi \). Checking whether a labeling is valid takes time linear in \( |A(v)| \).

Thus, initialization of all leaves of \( \mathcal{T} \) takes at most \( O((d + 3)^d) \) time. Since there are up to \( 2|A(v)| \) labelings \( \phi' \) such that \( \phi = \phi' \oplus v \), each forget requires \( O(|A(v)|) \) time. The joins require iterating through pairs of compatible labelings, of which there are \( 2^{|A(v)|} \) since each \( u \in \phi^{-1}(\infty) \) has a corresponding \( \phi_j \) for which \( \phi_j(u) = B \). Thus, each join requires \( O(2^{|A(v)|} ir) \) time. In total, this leads to a time complexity of \( O((2d + 6)^d \cdot k^4 \cdot n) \). \( \square \)

7.5.3 Weighted Variant

The extension to include edge weights for maximal matching is analogous to that for vertex cover (Section 7.4); we note the key differences below. First, we set \( W[v_1, \ldots, v_j] = x \) if and only if \( x \) is the smallest value such that there is a maximal matching \( S' \) of \((G \setminus \mathcal{T}) \cup (A[v] \setminus \phi^{-1}(B)) \cup \mathcal{T}_{\leq v_j} \) that satisfies (1)-(6) and

\[
\sum_{uv \in U} s(uv) \cdot w(uv) = x,
\]

where \( s(uv) = -1 \) if \( uv \in S \) and \( 1 \) otherwise and \( U \) is the set of edges with an endpoint in \((S' \triangle S) \cap \mathcal{T}_{\leq v_j} \). The non-zero table entries are initialized to \( (\|G\| + 1) \cdot \max_{e \in E(G)} w(e) \). We let forget\([W[v_1, \ldots, v_j]]\) be defined as

\[
W[v]\phi, i, r = \min_{\phi' = \phi \oplus v} (W[v_1, \ldots, v_j][\phi', i', r'] + y)
\]

where \( i' \) and \( r' \) are defined as in the unweighted variant and

\[
y = \begin{cases} w(v) & \text{if } i \neq i' \\ -w(v) & \text{if } r \neq r' \\ 0 & \text{otherwise.} \end{cases}
\]
The operation join\( (W[v_1, \ldots, v_{j-1}], W[v_j]) \) is defined so that
\[
W[v_1, \ldots, v_j][\phi, i, r] = \min_{\phi = \phi_1 \oplus \phi_2, \begin{array}{c} i = i_1 + i_2 \\ r = r_1 + r_2 \end{array}} W[v_1, \ldots, v_{j-1}][\phi_1, i_1, r_1] + W[v_j][\phi_2, i_2, r_2].
\]

As in the previous section, any table entries at the root with negative values correspond to the existence of valid \( k \)-exchanges.

### 7.6 Conclusion

We developed practical dynamic programming algorithms for local search problems in classes of bounded expansion. Though we have only specifically shown algorithms for Minimum Vertex Cover \( k \)-Exchange and Minimum Maximal Matching \( k \)-Exchange, we are hopeful this approach can be extended to more FO-verifiable problems like Minimum Dominating Set \( k \)-Exchange and Minimum Total Dominating Set \( k \)-Exchange. However, these problems seem to be intrinsically more difficult due to the fact that they are slightly “less local”. For example, in Minimum Vertex Cover \( k \)-Exchange we are able to guarantee that if edge \( uv \) is covered exclusively by \( u \) in \( S \) but exclusively by \( v \) in \( S' \), then \( \{u, v\} \subseteq T \) and \( u, v \) have an ancestor-descendant relationship. In contrast, if vertex \( w \notin T \) is dominated exclusively by \( u \in T \) in \( S \) but exclusively by \( v \in T \), \( u \) and \( v \) may not be in an ancestor-descendant relationship. In this case, we need to ensure that when the subtrees containing \( u \) and \( v \) are joined at their least common ancestor, at least one of them dominates \( w \). This is problematic because \( u \) and \( v \) may have arbitrarily many neighbors outside of \( T \), so “communicating” they have dominated their neighbors naively requires storing \( \Omega(2^n) \) entries in the dynamic programming tables. Finding a way to encode the vertices outside of \( T \) using only a polynomial number of entries thus plays a crucial role in using our techniques on a broader class of problems.
Summary

Graph-based methods are a common part of *metagenomic sequencing* in which genomes from a large community of species are assembled simultaneously from short reads of DNA. In these graphs, edges capture overlap between reads and thus individual genes are manifested as connected subgraphs. Under the assumption that subgraphs corresponding to closely related species are close in graph distance, it is useful to obtain subgraphs “surrounding” a known (partial) genome.

We address Research Question 5 through the creation of an information retrieval system for large metagenomic data sets that exploits the sparsity of the DNA graphs to efficiently summarize and organize its neighborhood structure [13]. The first step of this methodology is to create an index of the neighborhoods by partitioning the graph into small connected subgraphs in fpt time. Through defining an auxiliary graph based on the connectivity between indexed neighborhoods, we recursively apply the neighborhood indexing to create a progressively coarser representation of the graph. We build a novel data structure, the *CATlas*, from this hierarchy of neighborhoods that is optimized to perform *locality-sensitive searches*. Using a metagenomic data set with known ground truth, we conduct a proof-of-concept implementation and experimentation that confirms that each component of the system captures biologically relevant information in its neighborhood-based approach. Finally, we discuss ways
in which the locality-sensitive searching can be extended to other data domains such as social networks.

8.1 Introduction

The ultimate goal of increasing the practicality of structural graph algorithms is to aid researchers from other disciplines in their computational problems. To demonstrate the immediate viability of structural graph algorithms, we focused our attention on the problem of querying genomic data.

Computational biologists have long been interested in assembling unknown genomes of organisms from fragments of DNA, known as \textit{de novo} assembly [78, 81]. In microbial ecosystems in which the individual organisms cannot be easily isolated for separate analysis, scientists have used \textit{metagenomic sequencing} to assemble the genomes of multiple organisms simultaneously. Under the paradigm of \textit{shotgun sequencing}, short subsequences called \textit{reads} are randomly sampled from the genomes of all organisms, and then the aggregated reads are partitioned into sets belonging to the individual organisms in a process called \textit{binning} [54, 58]. These data sets are frequently represented as graphs whose vertices represent the reads and edges capture overlap between them in such a way that a properly sampled genome will be manifested in the graph as a connected subgraph.

In this context, it may be useful to extract a portion of the data that is close in the graph to a known sequence. Metagenomes frequently contain many strain variants of a particular organism whose genomes largely overlap but diverge at certain places. Additionally, many binning algorithms struggle to process regions of the graph with “high complexity” and often simply disregard them; thus the bins may not contain entire genomes. In both cases, the sequences we are trying to extract will be in close proximity to original genome or bin.

Although neighboring regions of the graph to the known sequence could be recovered by performing a small number of iterations of a breadth-first search (BFS), this approach suffers from two major setbacks. First of all, a single query may only utilize a small fraction of the graph and thus loading the entire graph into memory to compute the neighboring region is inefficient. Second, the BFS on its own lacks important details to contextualize its output. For example, we cannot \textit{a priori} determine how many vertices will be visited for a given number of BFS iterations, nor determine whether disconnected portions of the query are close together or far apart in the graph.

To overcome the first setback, we utilized a structure-based algorithm to pre-index the graph in a way that allows for random access to the neighborhood structure. This pre-indexing can be performed once to enable many queries. To provide the context necessary to interpret and filter the extracted neighborhoods, we complemented our indexing with
We implemented these algorithms and data structures in an open-source Python software package, spacegraphcats [14], and performed proof-of-concept tests to show their ability to recover meaningful biological information in large metagenomic data sets. We believe our methodology can also be generalized to be useful in domains outside metagenomics.

8.2 Background

8.2.1 De Bruijn Graphs

To create a more uniform data set, shotgun sequencing reads are commonly viewed through a sliding window of length \( k \), creating atomic units of equal length called \( k \)-mers [78]. These \( k \)-mers define a De Bruijn graph (DBG) whose vertices are the observed \( k \)-mers. An edge in the DBG between \( u \) and \( v \) indicates the last \( k - 1 \) bases of \( u \) match the first \( k - 1 \) bases of \( v \). In this way, two \( k \)-mers are adjacent if they could have occurred consecutively in the original genomes, and under idealized conditions each genome will manifest itself as a connected subgraph in the DBG. Thus assembly and binning both correspond to identifying connected subgraphs in the DBG.

Under this framework, long paths of degree two vertices have relatively low complexity and are easier to assemble and/or bin. The more difficult cases are subgraphs that contain vertices of degree three or more, which can appear when multiple organisms have common genes or when subsequences are repeated in different places on the same genome. To concentrate on the high-degree subgraphs while maintaining the overall graph structure, we form a contracted De Bruijn graph (cDBG) by contracting each path whose internal vertices all have degree two into a single vertex. We note that since DNA is constructed from four bases (A, C, T, G), each vertex in the DBG and cDBG has degree at most eight. Since classes of bounded degree also have bounded expansion, structure-based algorithmic techniques are appropriate for these data sets.

8.2.2 Data

As a proof-of-concept, we will test our methodology on podar-ref [90], a synthesized community of 64 known archaea and bacteria. From this data set, we produced 31-mers and constructed the cDBG using BCALM 2 [18]. The 64 genomes present in the data will be labeled \( g_0, \ldots, g_{63} \), using the numbering scheme that corresponds to the one used in [90]. Unlike some other metagenomic data sets, the ground truth genome identities are known, which allow us to compute the evaluation metrics described below.
8.2.3 Evaluation Metrics

The methods described in this chapter largely revolve around taking a query set of k-mers, Q, and outputting additional k-mers in the “neighborhood” of Q in the cBDG. Given a set X and a query Q, we define the similarity of X with respect to Q to be \( \frac{|Q \cap X|}{|X|} \), the overhead to be 1 minus the similarity, and the containment to be \( \frac{|Q \cap X|}{|Q|} \).

It will be important to evaluate whether the structure-guided organization of the genomic data places similar genomes into the same groups. Our metric of interest is the taxonomic purity at various taxonomic ranks, i.e., strain, species, genus, family, etc., because it captures relations between genomes regardless of whether they share any k-mers. Given a subset of k-mers X corresponding to known genomes and a taxonomic rank ρ, the taxonomic purity is the largest proportion of k-mers in X belonging to genomes with a common taxon at rank ρ; we use soursasm [12] to compute this measure. If our methods capture biologically relevant information, then the purity at low ranks will be high.

8.3 Neighborhood Indexing

To organize the cDBG into neighborhoods, we first find a set of “waypoints” that are collectively “close” to all vertices and then partition the remaining vertices into neighborhoods by assigning them to a nearest waypoint. The algorithm to compute the neighborhoods is described below.

8.3.1 Algorithmic Description

Formally, the “waypoints” are an r-dominating set D of G; that is, the minimum distance from every vertex in G to a member of D is at most r. Though computing a minimum-sized r-dominating set is NP-hard, we use a linear time approximation algorithm from Dvořák and Reidl [83] (Algorithm 9) for classes of bounded expansion. This algorithm first computes r rounds of distance-truncated transitive fraternal augmentations (Section 2.2.3), which guarantees that if the minimum distance \( d \) between vertices \( u \) and \( v \) is at most \( r \), the augmented graph \( \vec{G}_r \) contains one of the following:

- arc \( uv \) with label \( d \)
- arc \( vu \) with label \( d \)
- arcs \( wu \) and \( wv \) with labels \( i \) and \( d - i \) for some vertex \( w \) and integer \( i \)

By “pushing” and “pulling” the labels of arcs, we can in linear time compute all pairwise distances that are at most \( r \). Then, we greedily select vertices until we have formed an r-dominating set. The approximation factor depends on the grad (Definition 7) at depth \( r \); since
Algorithm 9 rdomset($G, r$)

**Input:** cDBG $G$, radius $r$

**Output:** $r$-dominating set $D$ of $G$

1. $\vec{G}_1 \leftarrow \text{orient}(G)$
2. for $i \in 2, \ldots, r$ do
3.   $\vec{G}_i \leftarrow \text{dtf}_{\text{aug}}(\vec{G}_{i-1})$
4. end for
5. $d(v) \leftarrow \infty \; \text{forall} \; v \in V(\vec{G}_r)$
6. $D \leftarrow \emptyset$
7. for all $v \in \vec{G}_r$ do
8.   for all $u \in N^-(v)$ do
9.     if $d(u) + \omega(uv) < d(v)$ then
10.    $d(v) \leftarrow d(u) + \omega(uv)$
11.   end if
12. end for
13. if $d(v) > r$ then
14.   $D \leftarrow D \cup \{v\}$
15. $d(v) \leftarrow 0$
16. for all $u \in N^-(v)$ do
17.   if $d(u) > \omega(uv)$ then
18.     $d(u) \leftarrow \omega(uv)$
19. end if
20. end for
21. end if
22. end for
23. return $D$

the class of De Bruijn graphs has maximum degree 8, it also has bounded expansion and thus has constant grad for any fixed depth.

To construct the neighborhoods, we assign each vertex $v \in G$ to a nearest member of $D$, denoted $\delta(v)$, such that all vertices that have $\delta(v) = u$ form a connected subgraph. This can be done by propagating the assignments outwards from the $r$-dominating set, as shown in Algorithm 10.

**8.3.2 Biological Impact**

To show the neighborhood indexing captures relevant biological information, we indexed the neighborhoods for podar-ref using a 1-dominating set. Then, for each genome $g_i$ in podar-ref we extracted the corresponding neighborhoods $N_i$ that have non-zero containment with respect to $g_i$ and evaluated taxonomic purity at various taxonomic ranks. Because each $N_i$ has a containment of 1 with respect to $g_i$, the taxonomic purity is trivially at least as large.
Algorithm 10 idx_nbrhsds(G, r)

Input: cDBG $G$, radius $r$
Output: Partition of $G$ into neighborhoods

1. $D \leftarrow \text{rdomset}(G, r)$
2. $\delta(v) \leftarrow \emptyset \ \forall v \in G$
3. $\delta(v) \leftarrow v \ \forall v \in D$
4. while $\exists u \in G$ such that $\delta(u) = \emptyset$ do
5. $X \leftarrow \{(u, w) : \delta(u) = \emptyset \land w \in N(u) \land \delta(w) \neq \emptyset\}$
6. for all $(u, w) \in X$ do
7. $\delta(u) \leftarrow \delta(w)$
8. end for
9. end while

10. for all $v \in D$ do
11. $B(v) = \{u : \delta(u) = v\}$
12. end for
13. return $\{B(v) : v \in D\}$

as the similarity. Thus, the more meaningful metric is overhead purity, defined as the proportion of k-mers of $N_i \setminus g_i$ that share a common taxon with $g_i$ at a particular rank. In Figure 8.1, we plot the overhead purity as a function of the overhead and observe that most genomes whose neighborhoods have large overhead obtain high purity at the family level or below.

![Figure 8.1](image-url)

Figure 8.1 Overhead purity at varying ranks of each $N_i$ as a function of the overhead of $N_i$. 
8.4 The CAtlas

Though the neighborhood indexing allows for efficiently extracting the neighborhoods of vertices in the cDBG, it lacks the ability to contextualize the proximity of neighborhoods in relation to one another. We capture the locality of the neighborhoods by introducing the contraction atlas (CAAtlas), a data structure that captures a multi-resolution view of \( G \) and is optimized for performing locality-sensitive queries.

8.4.1 Description

At its core, the CAtlas \( C \) is a rooted tree whose nodes are labeled by increasing integers, the levels, from bottom to top. Each node \( u \in C \) represents a connected subgraph of the cDBG, called its shadow, denoted \( S(u) \). Every vertex \( v \in G \) belongs to the shadow of exactly one node at each level \( i \) of \( C \). That is, the shadows of nodes in any given level form a partition of \( G \). In this way, the highest level of the CAtlas consists of a single node (the root of the tree) whose shadow is the entire cDBG, while the nodes at the lowest level (the leaves) have shadows corresponding to much smaller subgraphs. The parent-child relationships in \( C \) capture a finer/coarser-grained view of a particular region of \( G \); node \( u \) is a child of node \( v \) if and only if \( S(u) \subseteq S(v) \).

8.4.2 CAtlas Construction

The CAtlas is constructed level-wise from the leaves up to the root (Algorithm 11). The \( i \)th level has an associated graph, which we will denote \( G_i \), letting \( G_1 = G \). For each graph \( G_i \), we compute an \( r \)-dominating set \( D_i \) and assign vertices in \( G_i \) to members of \( D_i \) to identify the neighborhoods as described in the previous section; let \( d_i(u) \) be the assignment of \( u \in G_i \) to a member of \( D_i \). There is a one-to-one correspondence between level \( i \) nodes and the vertices in \( D_i \); let \( \eta_i(v) \) be the node at level \( i \) associated with vertex \( v \). For each \( u \in G_i \), \( \eta_i(d_i(u)) \) becomes the parent of \( \eta_{i-1}(u) \), and the shadows of the level \( i \) nodes are the unions of the shadows of their children\(^1\).

It is necessary to show how to construct \( G_i \) from \( G_{i-1} \). The vertices of \( G_i \) are exactly \( D_{i-1} \) and two vertices \( u, v \in V(G_i) \) are adjacent if there exists an edge \( u'v' \in E(G_{i-1}) \) such that \( u' \in S(\eta_i(u)) \) and \( v' \in S(\eta_i(v)) \). For this reason, we call \( G_i \) the domination graph of \( G_{i-1} \). Note that \( G_i \) is an \( r \)-shallow minor of \( G_{i-1} \), which means that each \( G_i \) belongs to a common class of bounded expansion and the \( r \)-dominating set algorithm retains a low approximation factor at each level. We continue making domination graphs from previous levels until \( G_i \) is sufficiently

\(^1\)For convenience, we define \( S(\eta_0(v)) = \{v\} \) for all \( v \in G_1 \).
Algorithm 11 catlas($G, r, \epsilon$)

**Input:** Graph $G$, radius $r$, positive real number $\epsilon$

**Output:** CAtlas root $R$

1. $G_1 \leftarrow G$
2. $R \leftarrow$ CAtlas node
3. $\text{children}(R) \leftarrow \emptyset$
4. $i \leftarrow 1$
5. **for all** $v \in G_1$ **do**
6. \hspace{1em} $\eta_0(v) = \emptyset$
7. \hspace{1em} $S(\eta_0(v)) = \{v\}$
8. **end for**
9. **repeat**
10. $D_i \leftarrow \text{rdomset}(G_i, r)$
11. $B_i \leftarrow \text{idx_nbrhds}(G_i, r)$
12. **for all** $v \in D_i$ **do**
13. \hspace{1em} $\text{children}(\eta_i(v)) \leftarrow \{\eta_{i-1}(u) : u \in B_i(v)\}$
14. \hspace{1em} $S(\eta_i(v)) \leftarrow \bigcup_{u \in B(v)} S(\eta_{i-1}(u))$
15. **end for**
16. $V(G_{i+1}) \leftarrow D_i$
17. **for all** $u, v \in D_i$ **do**
18. \hspace{1em} **if** $\exists u', v'$ such that $u' \in S(\eta_i(u))$, $v' \in S(\eta_i(v))$, and $u'v' \in E(G_1)$ **then**
19. \hspace{2em} $E(G_{i+1}) \leftarrow E(G_{i+1}) \cup \{uv\}$
20. **end if**
21. **end for**
22. **for all** isolated vertices $v \in G_{i+1}$ **do**
23. \hspace{1em} $V(G_{i+1}) \leftarrow V(G_{i+1}) \setminus \{v\}$
24. \hspace{1em} $\text{children}(R) \leftarrow \text{children}(R) \cup \eta_i(v)$
25. **end for**
26. $i \leftarrow i + 1$
27. **until** $|G_i| \leq \epsilon$
28. $\text{children}(R) \leftarrow \text{children}(R) \cup \bigcup_{v \in G_i} \eta_i(v)$
29. **return** $R$

small or until every component of $G_i$ is a single vertex. To finish the CAtlas, we add a root at level $i + 1$ whose children are all the level $i$ nodes and whose shadow is the entirety of $G_1$.

Since creating the domination graph preserves connectivity, i.e., each component of $G_i$ corresponds to a unique component of $G_{i-1}$, an isolated vertex in $G_i$ will correspond to an isolated vertex in $G_j$ for every $j \geq i$. To reduce the CAtlas size, if $v$ is an isolated vertex in $G_i$, we remove $v$ from $G_i$ and make $\eta_{i-1}(v)$ immediately a child of the root.
8.4.3 Biological Impact

We consider partitioning the $k$-mers into bins using the CAtlas rather than an intermediate assembly algorithm. In lieu of partially assembled genomes, our CAtlas-based approach creates equal-sized bins that preserve locality. Given a CAtlas $C$, we find the set of nodes $U$ such that each $u \in U$ satisfies $2000 \leq |S(u)| \leq 10000$ but no ancestor of $u$ satisfies this property. The bins constructed in this manner contain $k$-mers corresponding to genomes that are very taxonomically similar (Figure 8.2), demonstrating that the local structure captured by the CAtlas is meaningful in the domain.

![Figure 8.2 Histogram of taxonomic purity values of the shadows of the binned CAtlas nodes.](image)

8.5 CAtlas Search

Because the CAtlas provides a multi-resolution summary of a cDBG, it naturally lends itself to locality-sensitive searches. In these searches, we return neighborhoods “near” (parts of) a query based on user-specified overhead and containment.

8.5.1 Algorithmic Description

To search the CAtlas we are given a query $k$-mer set $Q$, a maximum overhead threshold $\alpha$, and a minimum containment threshold $\beta$. We build up a frontier of CAtlas nodes $F$ starting with $F = \{R\}$, where $R$ is the root. To refine the frontier, we pick the node $u \in F$ whose shadow has the largest overhead with respect to $Q$ and remove it from $F$. Then we add to $F$ each child of $u$ whose shadow has non-zero containment with respect to $Q$; when $u$ is a leaf, it is simply removed from the frontier entirely with no replacement. We continue refining the frontier until $\bigcup_{u \in F} S(u)$ has overhead less than $\alpha$ or containment less than $\beta$. 

114
8.5.2 Query Preprocessing

Searching the CAtlas requires efficiently computing the intersection between the shadow of a node and the query. However, the shadows of nodes at the upper levels of the CAtlas can be quite large; in particular, the shadow of the root is the entire cDBG. To ensure this intersection operation remains fast and memory-efficient at large graph sizes, we perform an overhead precomputation for each query \( Q \) using a bottom-up approach. Given a CAtlas node \( u \), it is straightforward to compute the overhead of \( u \) if the overheads of each of its children are known, since the shadows of CAtlas nodes not in an ancestor-descendant relationship are disjoint. Thus, it is sufficient to compute the overheads at the leaves and propagate the results upwards. At the leaves, we find the set of cDBG vertices whose shadows contain at least one \( k \)-mer in \( Q \). This can be done by creating a lookup table \( K \) that maps each observed \( k \)-mer to the unique vertex that contains that \( k \)-mer in its shadow; we implement \( K \) using a minimal perfect hash function. Note that using such a lookup table allows us to compute the overheads in \( O(|Q|) \) time rather than \( O(|G|) \) time.

8.5.3 Biological Impact

To show the locality-sensitive search can generate new insights beyond simply extracting the neighborhoods with non-zero containment with a query, we performed a final series of experiments. For each combination of \( \alpha \in \{.04, .08, .16, .32\} \) and \( \beta \in \{.99, .95, .90\} \), we searched with each genome \( g_i \) in podar-ref and retrieved the corresponding frontier \( F_i(\alpha, \beta) \). Reusing the general methodology from Section 8.3.2, the overhead purity of each frontier \( F_i \) was computed and plotted (Figure 8.3).

As expected, very few frontiers have high overhead purity at the strain or species rank for large \( \alpha \). However, at the genus or family rank, the large \( \alpha \) values do not prevent us from recovering many relatives of the query. Additionally, pairing a low value of \( \beta \) with a low value of \( \alpha \) results in large overhead purity at the strain and species rank. This suggests that the parts of the query with large overhead being removed from the frontier tend to contain \( k \)-mers that are not as closely related to the query. As such, we believe this parameter combination may be an effective way to “denoise” the neighborhoods containing the query \( k \)-mers.

8.6 Beyond Metagenomics

In a generic sense, the CAtlas can efficiently extract neighborhoods of vertices whose metadata corresponds to the query. Accordingly, we believe this methodology can be used on graphs from a broad set of domains. For example, the CAtlas could be used to query a social network for the people who are socially close to alumni of a particular school or to query a internet
Figure 8.3 Overhead purity at varying ranks of each \( F_i \) as a function of the overhead of \( F_i \). The colors of the points correspond to \( \alpha, \beta \) pairs.

network for computers near those that have been infected with a virus.

Furthermore, the neighborhood indexing and CAtlas construction algorithms can be adapted to accommodate different definitions of neighborhoods that are more applicable to other domains. Given an algorithm for partitioning a graph into disjoint connected subgraphs and a criterion for deciding whether two neighborhoods are “adjacent”, these subroutines can be substituted into the neighborhood indexing and CAtlas construction, respectively, without requiring changes to the querying procedure. This would allow users to ensure neighborhoods maintain specific properties such as high density, metadata homogeneity/heterogeneity, and/or uniform size. Although our proposed algorithms are guaranteed to be efficient for any class of graphs with bounded expansion, the alternative algorithms could be tailored to graphs with more or less restrictive structural properties as appropriate.

8.7 Conclusion

We introduced a structure-based approach for extracting neighborhoods in large De Bruijn graphs arising in metagenomic data analysis. Through a parameterized algorithm for \( r \)-dominating set, we indexed the neighborhoods of the graph for efficient extraction of \( k \)-mers close to a query set that maintained high taxonomic purity. The recursive indexing of neighborhoods on the domination graph was used to build a CAtlas to provide further context for
queries. It also led to efficient partitioning of the $k$-mers into similarly-sized bins containing $k$-mers from genomes in the same genus. We defined a locality-sensitive search that can be tailored to fit containment and overhead specifications. In future work, we would like to apply these methods to sequencing data for which there is no ground truth, e.g., [45], to draw new biological insights, as well as to adapt the CAtlas and its search for use in other data domains.
In this dissertation, we explored multiple strategies for bridging the gap between structure-based algorithms from the theoretical computer science community and the computational needs of the network science community. We identified five general fronts on which to make improvements: dealing with missing or incomplete data, evaluating viability of existing algorithms, proving new theoretical frameworks with practical consequences, designing algorithms without massive constants hidden in the computational complexity, and collaborating with domain scientists to tailor structural approaches to their data.

First, we considered the ability to measure structural features of a graph using only local information in the context of core numbers. We introduced a flexible scheme for locally estimating the core number of a vertex \( v \) using only the vertices at a distance of at most \( \delta \) away from \( v \). This is significant because determining the core number of any single vertex may require computing all core numbers in the graph despite the fact that the core number is a property of an individual vertex. Empirical analysis verified that these estimates are remarkably accurate on most vertices in real-world networks from different domains. It would be useful to see if these methods can be extended to estimate more restrictive types of structure such as treewidth or expansion.

Second, we conducted the first empirical analysis of a pipeline for counting subgraph isomorphisms in classes of bounded expansion. After implementing this pipeline in CONCUSS, we tested a number of subroutine choices that had similar asymptotic complexities but led to
different performances. Surprisingly, we discovered that some subroutines that tried to make “smart” decisions were outperformed by their naïve alternatives. Although it was unable to compete with non-structure-based algorithms in random graph models because the $p$-centered colorings used too many colors, CONCUSS outperformed the existing VF2 algorithm in graphs where the size of the $p$-centered coloring was small and the number of isomorphisms was large. Whether the random graphs actually have large (but bounded) $p$-centered coloring numbers or the algorithms are consistently finding colorings that are far from optimal is an important question to resolve in order to understand where to target new theoretical contributions.

The evaluation of CONCUSS also demonstrated that a low-treedepth coloring that uses fewer colors but creates subgraphs with larger treedepth would likely result in faster run times. In response, we studied the properties of $p$-linear and linear colorings and showed that they indeed fit these requirements. The crucial objective in this study was to bound the treedepth of a graph with fixed linear coloring number; we improved an exponential upper bound in general graphs to a cubic bound in trees and a quadratic bound in interval graphs, but were unable to prove the conjectured linear upper bound on general graphs.

Two additional developments would greatly improve the use of linear colorings in practice: a general algorithm for transforming a linear coloring into a treedepth decomposition of low depth and a sufficient condition for recognizing linear colorings. The latter of these may prove difficult to find given our co-NP-completeness proof for recognizing linear colorings.

In contrast to these positive theoretical results, we showed that there is unlikely to be an efficient—let alone practical—algorithm for finding the densest $\frac{r}{2}$-shallow minor for any $r > 0$. This result extends to show the hardness of finding a single-exponential algorithm parameterized by treewidth for any $r > 1$. Given these relatively strong hardness results, it appears there is some intrinsic difficulty finding dense substructures that are even slightly less local than subgraphs. Finding a new substructure definition that counters this claim would not only be interesting from a theoretical perspective but could also lead to new definitions of structural classes that enable efficient algorithms.

In terms of designing algorithms that exploit structure without “heavy machinery”, we turned our attention to the local search variant known as $k$-exchange. We designed practical FPT dynamic programming algorithms based on those successfully implemented in CONCUSS to find $k$-exchanges for Vertex Cover, Independent Set, and Maximum Matching. These algorithms can be easily modified to solve variants in which vertices and/or edges have weights and the objective is to minimize or maximize the sum of weights in the solutions. To validate their practicality, these algorithms should be implemented and incorporated into a program like CONCUSS in future work. We would also like to extend our algorithmic techniques to more first-order verifiable problems, such as Dominating Set or Total Dominating Set.
On the fifth front, we collaborated with computational biologists from UC Davis to create a proof-of-concept of a structure-based framework for organizing and analyzing metagenomic De Bruijn graphs. Our approach had three components: an algorithm that partitions the graph into neighborhoods, a hierarchical data structure—the CAtlas—based on partitioning the “domination graph” that captures adjacency between neighborhoods, and a search strategy to extract neighborhoods in the graph that uses the CAtlas to preserve locality. Because the De Bruijn graphs belong to a class of bounded expansion, the $r$-dominating set algorithm at the core of the partitioning runs in linear time with a constant factor approximation ratio and the domination graphs are guaranteed to also be sparse. In a proof-of-concept, we used these three components to organize sequencing data based on graph structure and retrieve subgraphs coinciding with known biological taxonomy. The CAtlas can easily be adapted to allow querying by generic metadata, which means it may be applicable to the network science community at large. Finding a second use case for this methodology would further demonstrate the immediate usefulness of structure-based algorithms in practice.

Overall, this dissertation has documented a promising avenue on each of the five fronts for improving the practicality of structural graph algorithms. We believe that future research tackling the open problems outlined above will yield useful tools for practitioners that scale well to large, sparse data sets.
BIBLIOGRAPHY


In the following, we detail options for configuring CONCUSS [75].

Configuration options

It is possible to make a custom configuration file by adhering to the following format:

[color]
low_degree_orientation = <low degree orientation module>
step = <step-by-step addition of edges module>
coloring = <coloring heuristic module>
check_tree_depth = <tree depth checking module>
optimization = <optimization module, optional>
preprocess = <preprocessing module, optional>

[decompose]
sweep = <decomposition generator module>

[compute]
k_pattern = <k-pattern module>
table_forward = <True or False>
table_reuse = <True or False>

[combine]
count = <count combiner module>

where the entries in angular brackets are replaced with the appropriate choices.

**Description of module choices**

Below are options for each of the entries in the configuration files.

**low_degree_orientation**

These modules transform an undirected graph to a directed acyclic one in a way that minimizes the maximum indegree.

- basic.low_degree_orientation Minimize the indegrees as the edges are being oriented.
- basic.sandpile_orientation Find one orientation of the edges and then use sandpiling to reduce the indegrees.

**step**

These modules augment directed edges to the graph that in turn place restrictions on the subsequent coloring. They may be executed multiple times as “steps”, and thus the criteria for adding edges depends on the state of the graph at the start of the step. In the following descriptions we write $ab$ to denote a directed edge from $a$ to $b$.

- basic.trans_frater_augmentation
  
  Add edges between all vertex pairs that were fraternal or transitive at the beginning of this step. Vertices $a$ and $b$ are fraternal if there is some vertex $c$ such that $ac$ and $bc$ are both edges in the graph. Vertices $a$ and $b$ are transitive if there is a vertex $c$ such that $ac$ and $cb$ are both edges in the graph.

- basic.truncated_tf_augmentation
  
  Add edges between some vertices that were transitive and fraternal at the beginning of this step. If $ac$ was added during step $i$ and $cb$ was added at step $j$, we do not need to add $ab$ until step $i + j$. 

131
**coloring**

These modules give heuristics about how to order the vertices of a graph to color them greedily.

- **coloring.min_deg**
  Prioritize coloring vertices with small degree.

- **coloring.max_deg**
  Prioritize coloring vertices of large degree.

- **coloring.dsatur**
  Use the DSATUR heuristic [11], which prioritizes coloring vertices that already have many colors represented among its neighbors.

**check_tree_depth**

These modules check to see whether each subgraph containing fewer than \( p \) colors has a center (a color appearing exactly once).

- **basic.check_tree_depth**
  Currently the only checker, which uses union-find data structures.

**optimization**

These modules take an existing \( p \)-centered coloring and attempt to reduce the number of colors.

- **basic.optimization_interval**
  Add random transitive and fraternal edges to the graph and re-run the greedy coloring.

**preprocess**

These modules preprocess the graph in order to give a better coloring.

- **basic.trim_high_degree**
  Remove vertices whose degrees are above some threshold (default to \( 4\sqrt{n} \)). After performing the desired coloring on the remainder of the graph, replace the removed vertices and give them each a unique color.

- **basic.trim_low_and_high_degree**
  Same as **basic.trim_high_degree**, but also removes vertices of degree 1 and gives them all the same (unique) color when added back.
sweep

These modules determine the order in which the sets of colors are analyzed.

- **CombinationsSweep**
  
  Use the Python built-in `itertools.combinations` method.

- **DFSSweep**
  
  Traverse the color space in a DFS-like manner. For example, in a 4-centered coloring with 9 colors this would look like: \{1\}, \{1,2\}, \{1,2,3\}, \{1,2,3,4\}, \{1,2,3,5\}, ... \{1,2,3,9\}, \{1,2,4\}, \{1,2,4,5\}, \{1,2,4,6\}, etc. Store partial vertex sets in order to reduce redundant computation i.e. retain vertices with colors \{1,2,3\} when computing \{1,2,3,4\} because they will all appear in \{1,2,3,5\}.

k_pattern

These modules represent the pieces of the pattern that can be found along root paths of the treedepth decomposition.

- **KPattern**
  
  Basic representation using Python built-in set data structures.

- **BVKPattern**
  
  Replace Python built-in sets with bitvectors as to do set operations as integer operations.

- **MemoizedKPattern**
  
  Memoize (store and remember) the set of all valid k-patterns for fast repeated iterations.

- **MemoizedBVKPattern**
  
  Use bitvectors for set operations and memoize the set of all valid k-patterns, join operations, forget operation, and boundary iterators.

table_forward

This option controls the order in which the dynamic programming table is filled. The value of a particular entry in the table is affected by various contributions from other entries in the table.

- **True**
  
  For each entry, add its contributions to all other entries later in the table that it affects.
- False
  For each entry, add all the entries that contribute to it from earlier in the table.

**table_reuse**

This option decides where to store the dynamic programming table of a particular treedepth decomposition.

- True
  Overwrite the entries of the table from the previous treedepth decomposition.

- False
  Allocate a new table for each treedepth decomposition

**count**

These modules determine how double counting of patterns is resolved.

- InclusionExclusion
  Counts occurrences of pattern that use fewer colors than its number of vertices, and then apply the inclusion-exclusion principle to correct the overcounting.

- ColorCounts
  Store the number of pattern occurrences for each set of colors and do not count a color set that has already been seen.

- HybridCount
  Store the number of pattern occurrences for a small, select set of colors.

- BVColorCounts
  Same as ColorCounts but use bitvectors instead of Python sets.

- BVHybridCount
  Same as HybridCounts but use bitvectors instead of Python sets.