STRUCTURAL ROUNDING ON A PARAMETERIZED GRAPH CLASS USING HEURISTICS

by

Cole Perschon

A senior honors thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Honors Degree in Bachelor of Science

In

Computer Science

School of Computing

The University of Utah

April 2021

ABSTRACT

Structural rounding is a framework for approximating NP-hard optimization problems on graphs near structured classes [10]. It has previously been empirically shown to outperform standard 2-approximations for VERTEX COVER on near-bipartite graphs [21]. Though promising, it is unclear if these findings are representative of structural rounding in general since the remainder of the framework's theoretical results have yet to be tested in practice. In this thesis, we consider the problem of DOMINATING SET on near-bounded treewidth graphs. We engineer structural rounding in this setting and test its performance against a $\log \Delta$-approximation algorithm. We implement two treewidth heuristics to improve runtime during editing, at the cost of theoretical guarantees on solution quality. We show that for both methods editing to smaller target treewidth increases edit set sizes but improves overall solution quality, which contradicts structural rounding's previous evaluation. We also present a synthetic graph generator that allows us to produce tunable random graphs with bounded distance to a target treewidth.

For my mother, Meg.

CONTENTS

ACKNOWLEDGEMENTS

I am grateful to all of the brilliant and kind people with whom I have had the pleasure of working with on this project. Firstly, I'd like to thank my mentor, Dr. Sullivan, for her unwavering guidance and support, and for distilling in me some of her abundant passion for science. Secondly, I'd like to thank Brian Lavallee, who was always available to teach me something new and who spent countless hours assisting me at every step of the way. Finally, I'd like to thank Madison Cooley for supplying research materials and feedback. This thesis truly would not have been possible without any of you.

This research was supported in part by The University of Utah Office of Undergraduate Research under Fall 2020 and Spring 2021 Undergraduate Research Opportunities Program (UROP) awards to Cole Perschon, and by the Gordon & Betty Moore Foundation's Data-Driven Discovery Initiative under Grant GBMF4560 to Blair D. Sullivan. This research was done in collaboration with Madison Cooley, Brian Lavallee, and Blair D. Sullivan.

CHAPTER 1

INTRODUCTION

Finding exact solutions to many optimization problems on graphs is often computationally intractable, especially as datasets scale. However, in many real-world scenarios (social, biological, etc.) optimal solutions are not always necessary [13][24][25]. Approximation algorithms offer an approach to solving these problems with a reasonable trade-off between runtime and guaranteed solution quality. In graphs, many existing approximations require rigid structural properties of the domain data, preventing their use on noisy real-world networks. Structural rounding [10] is a framework which extends approximation algorithms for restricted classes to graphs "near" those classes. Theoretically, structural rounding can be applied to a broad set of optimization problems and structural classes [10]. Extending the practical evaluation of the structural rounding framework to a parameterized class (bounded treewidth) will be the primary focus of this work.

Roughly, the steps of structural rounding are to **edit** an input graph into a desired structural class, efficiently **solve** the problem on this class, then **lift** the solution onto the original graph. The framework, formally defined in Chapter 2, was previously evaluated in [21] for approximating VERTEX COVER on near-bipartite graphs. It showed that structural rounding's solution quality was consistently better than traditional 2-approximations, performing best when the distance to bipartite was a small fraction of the instance size. Further, the authors showed that editing strategies which produced the smallest edit sets also produced the best (smallest) solutions. It is unclear, however, if these results are representative of structural rounding in general, or whether they are specific to this problem/class combination. Our work aims to answer this question in part.

In this thesis, we explore the effects of editing heuristically within structural rounding which removes our solution quality guarantees in pursuit of smaller edit sets and runtimes. We do so by studying structural rounding for solving DOMINATING SET on near-bounded treewidth graphs. Detailed in Chapter 3, this setting was selected because DOMINATING SET is a harder problem to solve than VERTEX COVER [11] and, in fact, has no constant-factor approximation algorithms on general graphs [12][19]. Further, bounded treewidth is a parameterized class which measures how close a given graph is to being a tree. This relaxation of structural requirements expands our set of feasible edited graphs, but also necessitates another parameter choice, treewidth. To that end, we also investigate how to select a target treewidth for editing to balance solution quality with runtime.

Our primary engineering contribution, described in Chapter 4, is comprised of two endeavors. First, to perform our experiments, we needed graphs with tunable structure. To achieve this, we present a synthetic graph generator that allows us to produce graphs with a bounded distance to a target treewidth. Second, we augment the existing graph library [16] which implements structural rounding for VERTEX COVER on near-bipartite graphs [21] to include greedy editing heuristics for treewidth from [6].

Ultimately, we were surprised to find that *larger* edit sets produced better overall approximations which contradicts the results of [21]. As a direct result, there was found to be no trade-off when editing to smaller treewidth values; smaller target treewidths produced better overall approximations, in less time, for both heuristic methods tested. Our data and figures to support these findings are contained in Chapter 5.

Questions still remain with respect to how our results will generalize to other problem/class combinations amenable to structural rounding. Chapter 6 concludes our thesis by proposing future practical evaluations which might strengthen our empirical results.

CHAPTER 2

BACKGROUND AND PRELIMINARIES

In this work, we assume that graphs are simple, i.e., unweighted and undirected, containing no loops or multiple edges. We denote a *graph* $G = (V, E)$ and let the number of *vertices* or *nodes* $n = |V|$ and the number of *edges* $m = |E|$. Edges are denoted by (u, v), or simply uv, for vertices $u, v \in V$. Further, we say v is *adjacent* to u if $(u, v) \in E$ and say v is *incident* to an edge if v is one of the two vertices the edge connects. We denote the set of *neighbors* of a given vertex v to be $N(v) = \{u \colon (u, v) \in E\}$ with $\deg(v) = |N(v)|$ being the *degree* of v. Finally, we denote the *maximum degree* of G to be $\Delta(G)$, or simply Δ, and ignore *isolated vertices* v where $\deg(v) = 0$. Given some $V' \subseteq V$ in G, we define $G[V']$ to be the subgraph of G induced on the vertices V'.

2.1 Algorithms and Complexity

In this section, we cover key concepts from theoretical computer science including NP-hardness, approximation, optimization problems on graphs, and parameterization.

2.1.1 NP-hardness

The output to any *decision* problem is Boolean, i.e., yes or no. In computational complexity theory, *P* (polynomial-time) and *NP* (non-deterministic polynomial time) are two classes of decision problems with restrictions on how fast solutions can be achieved, where P $\subseteq$ NP. While problems from class P are solvable in polynomial time, it is widely believed that there are no polynomial-time algorithms for some NP problems, that is, P $\neq$ NP. In this work, we assume the P $\neq$ NP conjecture holds [9].

NP-hard problems are the class of problems that are at least as "hard" as the hardest problems in NP, i.e., given an algorithm for an NP-hard problem, you could solve any problem in NP in polynomially more time. Problems which are in both NP and NP-hard are called *NP-complete*, i.e., NP-complete = NP $\cap$ NP-hard. These relationships are illustrated in **Figure 2.1** on this page. Both decision problems we discuss in this thesis (VERTEX COVER, and DOMINATING SET) are known to be NP-complete.

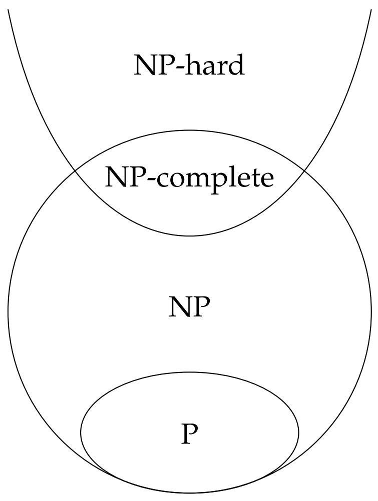

Figure 2.1. Euler diagram for P, NP, NP-complete, and NP-hard sets of problems under the assumption that P $\neq$ NP.

2.1.2 Optimization Problems

In contrast to decision problems, *optimization problems* no longer ask whether a solution of a given problem exists, but rather for you to find one which minimizes or maximizes an objective function. Here, we define the optimization problems from [10][21] needed for our work in Chapter 3.

We begin with the VERTEX COVER problem which, as one of Karp's 21 NP-complete problems [20], is an example of a classical decision problem. A *vertex cover* of a graph is a set of vertices which is incident to every edge. It's easy to see that finding such a set is trivial: the set of all vertices from a graph is also a vertex cover for the graph. Finding a cover with a minimum amount of vertices is much more interesting and is an example of an optimization problem; we formally define it below.

VERTEX COVER (VC)

Input:	A graph $G = (V, E)$.
Problem:	Find a set of vertices $X \subseteq V$ such that for every (u, v) in E either $u \in X$ or $v \in X$.
Objective:	Minimize $\lvert X \rvert$.

For example, a business interested in employing the fewest amount of security guards to cover all hallways (edges) would need to solve VERTEX COVER as described. If instead they wanted to cover all rooms (vertices) they would need to solve a similar problem, DOMINATING SET.

DOMINATING SET

Input:	A graph $G = (V, E)$.
Problem:	Find a set of vertices $X \subseteq V$ such that every vertex in V is either in X or is adjacent to a vertex in X.
Objective:	Minimize $\lvert X \rvert$.

Optimization problems stemming from DOMINATING SET have broad practical applications in the analysis of social networks [31], biological networks [25], and other domains. Although both problems are of practical interest, they remain challenging to solve on general graphs because they are NP-hard [28].

2.1.3 Approximation Algorithms

A natural approach to reducing runtime when solving an optimization problem is to allow suboptimal solutions. *Approximation algorithms* do this by achieving polynomial runtime while still providing a solution quality guarantee. For minimization problems, such methods provide an upper-bound on the multiplicative factor of deviation from optimal, known as the *approximation ratio*. For example, an approximation algorithm which produces a solution that is guaranteed to be at-worst twice as big as an optimal solution is known as a 2-approximation.

The best known 2-approximation was discovered by Gavril and Yannakis in [27] for approximating VERTEX COVER. The algorithm repeatedly picks an arbitrary edge (u, v) in G, places both u, v in X (the cover), then repeats on $G = G[V \setminus \{u, v\}]$ until no edges remain. **Figure 2.2** on the next page provides a visualization of an exact solution and an approximation (using this approach) to solve VERTEX COVER. Although the 2-approximation from [27] always produces solutions twice larger than optimal in this example, that is not true in the general case.

Approximation algorithms, while often a preferred approach for applications that don't

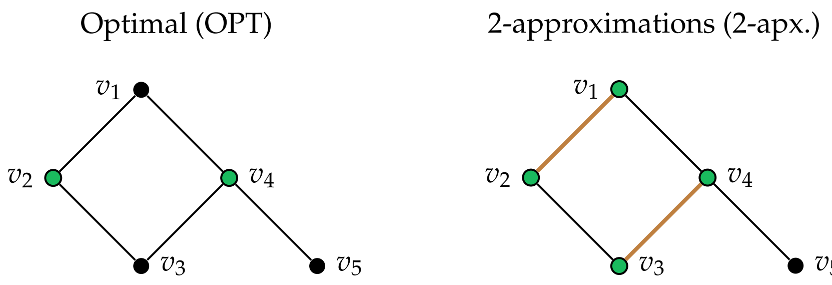

Figure 2.2. VERTEX COVER is both solved and approximated for a given graph G, with solution vertices colored green. An optimal solution is given on the left where $|X_{OPT}(G)| = 2$, and a 2-approximation produced by Gavril et al. on the right, with selected edges in brown, where $|X_{2\text{-}apx.}(G)| = 4$.

require optimal solutions, are not feasible for every optimization problem. Like NP-hardness, problems also exhibit an *approximation hardness* which determines their best possible theoretical approximation ratio. Some problems can even be shown to be *non-approximable*, meaning they will never have approximation algorithms. In fact, many NP-hard problems are non-approximable.

Furthermore, approximation algorithm runtimes, though required to be polynomial, are still not always fast enough in practice. Non-approximability and/or runtime infeasibility can therefore prevent the use of approximation algorithms. When this is the case, *heuristics* are often implemented. Heuristics depart from approximations by providing no provable limits on their worst-case solution quality in the hope that their typical solutions will still have reasonably good quality while running much faster.

2.1.4 Parameterized Algorithms

Parameterized algorithms offer another approach to tackling hard optimization problems by exploiting extra information about an instance (bounded as an integer k) to produce an optimal solution in time which depends on both n and k. For example, if we restrict the maximum size of a vertex cover to be $k \leq n$ (denoted k-VERTEX COVER), there is an $O(1.2738^k + kn)$-time exact solution [7]. Parameterized problems like k-VERTEX COVER which have solutions that are exponential only in the size of a fixed parameter (k) are known as *fixed-parameter tractable* and belong to the parameterized complexity class FPT. In this way, parameterized algorithms are best employed in domains which are known to possess small optimal solutions to a problem that is in FPT.

2.2 Graph Theory

In this section, we look at some of the fundamentals of graph theory.

2.2.1 Density

Graph *density* is defined to be the ratio of the number of edges m with respect to the maximum number of edges possible, denoted $\text{den}(G)$. Alternatively, *average degree* $\left(\frac{m}{n}\right)$ can often be used as a measure of "density." *Sparse* graphs, where $m \approx n$, in turn possess small densities. Conversely, *dense* graphs are known to have larger densities.

For simple graphs, the maximum number of edges is $(n-1)+(n-2)+\ldots+1+0 = \frac{n(n-1)}{2}$. This value is obtained by noticing that a node can be connected to at most $n-1$ other nodes, hence, $n(n-1)$. However, we must divide this amount by 2, because we are effectively counting every edge twice. We call a graph with $m = \frac{n(n-1)}{2}$ a *clique* or a *complete graph*. Note that cliques have density 1.0. We illustrate density and cliques in **Figure 2.3** on this page.

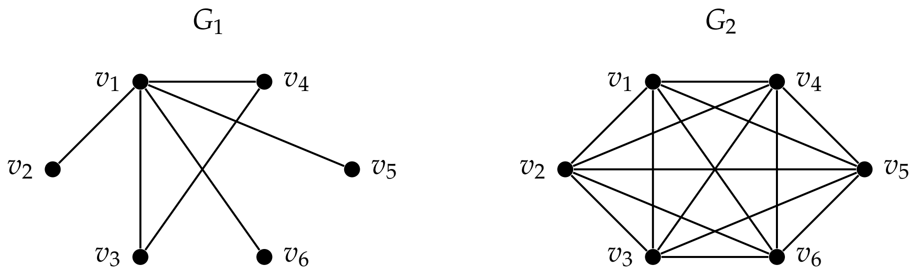

Figure 2.3. Two graphs, $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$. Notice that in this example, $\text{den}(G_1) = 6/15 = 0.4$ and $\text{den}(G_2) = 15/15 = 1.0$. Hence, G_2 is a clique.

2.2.2 Structural Classes

A *structural class* is a set of graphs which share common restrictions. Such rules often influence the density of the graphs in the class and present potential algorithmic advantages. In this subsection, we'll introduce the classes of trees, bipartite graphs, and bounded treewidth graphs.

Starting simply, the class of *trees* is all connected graphs which have *exactly* one path between any two vertices. For instance, consider a graph created by ordering vertices in a row with an edge connecting each vertex to the next. Notice that this graph is an example of a tree and that it will always have *exactly* one less edge than vertex. This is a necessary condition, and all trees have $m = n - 1$. An example tree is shown in **Figure 2.4** on the current page.

Tree

Definition: A graph $G = (V, E)$ in which any two vertices $v_1, v_2 \in V$ are connected by *exactly* one path.

Trees are one of the most structurally restrictive classes and inspire many others. They also have small densities, i.e., $\text{den}(T) = \frac{(n-1)}{\max(m)} = \frac{(n-1)}{\frac{n(n-1)}{2}} = \frac{2}{n}$ for any tree T. The density of a tree with just $10,000$ vertices is $\frac{2}{10{,}000} = 0.0002$, or only 0.02%. Approximation algorithms often benefit from the sparsity that comes with certain structured classes. However, structure does not necessarily imply sparsity; consider cliques. Certain classes permit the use of faster approximation algorithms altogether—some even admit fast exact solutions. One such structured class that has a polynomial time exact solve [17] for VERTEX COVER is *bipartite* graphs which are a superset of trees (i.e., every tree is also a bipartite graph).

Bipartite

Definition: A graph $G = (V, E)$ whose vertices can be partitioned into two disjoint sets (L, R) where, for every edge $(u, v) \in E$, $u \in L \iff v \in R$.

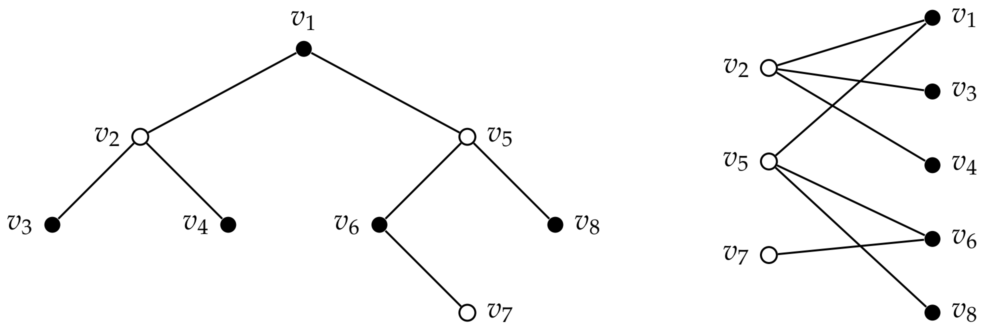

Figure 2.4. An example of a bipartite graph colored black and white. The representation on the right illustrates our graph rearranged to be partitioned with $L = \{v_2, v_5, v_7\}$ and $R = \{v_1, v_3, v_4, v_6, v_8\}$.

Structured classes such as trees and bipartite graphs present good opportunities for better approximations and solutions to some hard optimization problems. However, requiring graphs to have a rigid structure is often untenable in practice. Like parameterized algorithms, there also exist parameterized structural classes which possess an integer variable bounding how much of a certain structure is required. *Treewidth*, for example, is a heavily studied parameterized class which intuitively measures how “close” an input graph is to being a tree. To find this measure, we utilize *tree decompositions* which (non-exclusively) join vertices to form trees.

Tree Decomposition

Definition: Given a graph $G = (V, E)$, a tree decomposition is a pair (X, T) where $X = \{X_1, \dots, X_n\}$ is a set of (potentially overlapping) subsets of V, called *bags*, and T is a tree with nodes X, satisfying the following properties:

1. $X_1 \cup X_2 \cup \ldots \cup X_n = V$, i.e., the union of all bags X_i is V.
2. For every edge $(u, v) \in E$, there exists a bag X_i such that $u, v \in X_i$.
3. If $v \in X_i$, X_j, then all X_k of the tree in the (unique) path from X_i to X_j has $v \in X_k$.

Treewidth

Definition: The minimum width over all tree decompositions of a graph G, denoted $\text{tw}(G)$.

Treewidth is defined on all graphs, and smaller treewidth implies sparser graphs. Moreover, the class of graphs with treewidth at most k admits FPT algorithms parameterized by k (in contrast to solution size) for many NP-hard optimization problems. For example, if we restrict the maximum treewidth of the graph to $k \leq n$, there is an exact solution to DOMINATING SET in $O(4^k n)$-time [2].

2.3 Structural Rounding

Network science has empirically shown that many real-world networks are sparse. There is also some evidence they are “close” to structured classes, implying they can be converted to a graph from a structural class with few edits. Structural rounding is a framework by Demaine et al. [10] which offers a method for approximating NP-hard optimization problems on graphs that don't fall into a nice structural class. The idea is that structural rounding extends algorithms from restricted classes to general graphs while maintaining an approximation guarantee.

Earlier work from Magen and Moharrami [22] imagined something similar to this where a desired structure within the original graph is the "true" version of the graph and that all edits, (i.e., vertex- or edge-deletions), required to obtain it are considered "noise" and thus, can be ignored in the solution. The direct consequence of this approach is that it does not produce valid solutions on the original graph, only the edited network. Structural rounding differs from this model by *lifting* the solution found on the edited graph via carefully reincorporating the edit set ("noise"), to produce provably approximate solutions on the *original* graph.

2.3.1 General Framework

The three steps of the structural rounding framework are editing, solving, and lifting. Each step can be performed either exactly or approximately and is amenable to both minimization and maximization problems. We restrict our attention to the minimization setting throughout the rest of our discussion. We start by illustrating structural rounding in **Figure 2.5** on this page and give the relevant definitions from [10].

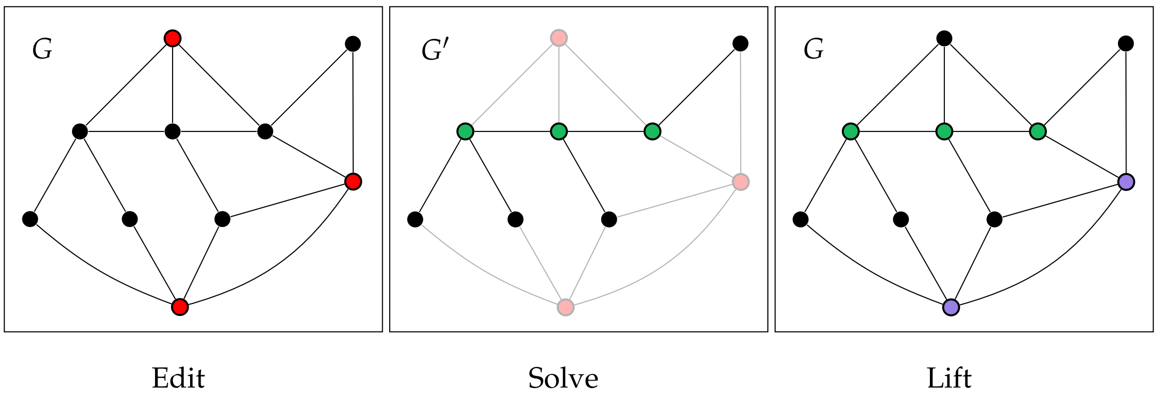

Figure 2.5. Structural rounding is demonstrated by editing the graph G to bipartite G' with vertex-deletions (in red), solving VERTEX COVER exactly on G' (in green) using [17], then greedily lifting edited vertices (in violet) to form a solution to the original graph, G.

Our first step, editing, starts with an input graph G that is γ-close (see **Definition 2.1)** to a structural class $\mathcal{C}$.

Definition 2.1. A graph G' is ***γ-editable*** from a graph G under edit operation ψ if there is a sequence of $k \leq \gamma$ edits $\psi_1, \psi_2, \ldots, \psi_k$ of type ψ such that $G' = \psi_k(\psi_{k-1}(\cdots \psi_2(\psi_1(G)) \cdots))$. A graph G is ***γ-close*** to a graph class $\mathcal{C}$ under ψ if some $G' \in \mathcal{C}$ is γ-editable from G under ψ.

In other words, at most γ edits of type ψ are needed to convert G to a graph G' in $\mathcal{C}$, where ψ is an edit operation (e.g., vertex deletion, edge deletion, or edge contraction). We approximate the minimum number of edit operations needed to get to $\mathcal{C}$ using the general optimization framework $(\mathcal{C}, \psi)$-EDIT.

$(\mathcal{C}, \psi)$-EDIT

Input:	An input graph $G = (V, E)$, family $\mathcal{C}$ of graphs, edit operation ψ.
Problem:	Find k edits $\psi_1, \psi_2, \ldots, \psi_k$ such that $\psi_k(\psi_{k-1}(\cdots \psi_2(\psi_1(G)) \cdots)) \in \mathcal{C}$.
Objective:	Minimize k.

If the desired structural class is parameterized, such as with treewidth, we denote it as $\mathcal{C}_\lambda$ where λ represents the desired parameter value. The optimization framework $(\mathcal{C}_\lambda, \psi)$-EDIT is similarly defined.

$(\mathcal{C}_\lambda, \psi)$-EDIT

Input:	An input graph $G = (V, E)$, parameterized family $\mathcal{C}_\lambda$ of graphs, a target parameter value λ^*, edit operation ψ.
Problem:	Find k edits $\psi_1, \psi_2 \ldots, \psi_k$ such that $\psi_k(\psi_{k-1}(\cdots \psi_2(\psi_1(G)) \cdots)) \in \mathcal{C}_\lambda$ where $\lambda \geq \lambda^*$.
Objective:	Minimize k.

Notice $\mathcal{C}_\lambda \subseteq \mathcal{C}_{\lambda+1}$. Thus, $(\mathcal{C}_\lambda, \psi)$-EDIT can also approximate λ by broadening the graph class we are aiming for to achieve membership in a super-class of the original target λ^*. By approximating both ψ and λ, $(\mathcal{C}_\lambda, \psi)$-EDIT is an example of a bicriteria problem (see **Definition 2.2**).

Definition 2.2. An algorithm for $(\mathcal{C}_\lambda, \psi)$-EDIT is a ***(bicriteria) (α, β)-approximation*** if it guarantees that the number of edits is at most α times the optimal number of edits into $\mathcal{C}_\lambda$ and that $\lambda \leq \beta \cdot \lambda^*$.

In either framework, the editing operation is chosen with respect to the optimization problem. For structural rounding to work for a specific problem, the amount a solution to the problem on the edited graph can increase for each edit operation must be measurable and bounded. To that end, we define a property called *stability* (see **Definition 2.3**) which we denote with constant c'. We follow [10] and use $\mathrm{OPT}_\Pi(G)$ for the value of the optimal solution to the problem Π on graph G.

Definition 2.3. A graph minimization problem Π is ***stable*** under an edit operation ψ with constant c' if $\mathrm{OPT}_\Pi(G') \leq \mathrm{OPT}_\Pi(G) + c'\gamma$ for any graph G' that is γ-editable from G under ψ. In the special case where $c' = 0$, we call Π ***closed*** under ψ.

For example, VERTEX COVER is closed under vertex deletion, meaning removing a vertex will not increase the optimal solution on the edited graph G'. In contrast, DOMINATING SET is not stable under vertex deletion, meaning removing a vertex could increase the optimal solution on G' by an unbounded amount. Consider removing the center vertex from a star, for instance.

Once G has been edited into G', the next step is to solve Π on the edited graph G' using a polynomial-time exact or approximate algorithm designed specifically for graphs in the class. The final step is to extend this partial solution found on G' to a full solution on the input graph G. In addition to stability, we now need to bound the amount the full solution on G can increase for every element in the edit set. To do this, we define another property called *lifting* (see **Definition 2.4**) which we denote with constant c. We follow [10] again and use Cost_Π to represent the cost function for a problem Π.

Definition 2.4. A minimization problem Π can be ***structurally lifted*** with respect to an edit operation ψ with constant c if, given any graph G' that is γ-editable from G under ψ and the corresponding edit sequence $\psi_1, \psi_2, \ldots, \psi_k$ with $k \leq \gamma$, a solution S' for G' can be converted in polynomial time to a solution S for G such that $\mathrm{Cost}_\Pi(S) \leq \mathrm{Cost}_\Pi(S') + c \cdot k$.

We now state the main result of structural rounding (Theorem 4.1) in [10].

Theorem 2.1 (Structural Rounding Approximation). Let Π be a minimization problem that is stable under the edit operation ψ with constant c' and that can be structurally lifted with respect to ψ with constant c. If Π has a polynomial-time $\rho(\lambda)$-approximation algorithm in the graph class $\mathcal{C}_\lambda$, and $(\mathcal{C}_\lambda, \psi)$-EDIT has a polynomial-time (α, β)-approximation algorithm, there is a polynomial-time $((1 + c'\alpha\delta) \cdot \rho(\beta\lambda) + c\alpha\delta)$-approximation algorithm for Π on any graph that is $(\delta \cdot \mathrm{OPT}_\Pi(G))$-close to the class $\mathcal{C}_\lambda$.

Stability and lifting results from [10] for problems on near-bounded treewidth graphs amenable to structural rounding are shown in **Table 2.1** below.

Problem	**Edit type** ψ	c'	c
INDEPENDENT SET (IS)	vertex deletion	1	0
ANNOTATED DOMINATING SET (ADS)	vertex* deletion	0	1
CONNECTED DOMINATING SET (CDS)	vertex* deletion	0	3
VERTEX COVER (VC)	vertex deletion	0	1
FEEDBACK VERTEX SET (FVS)	vertex deletion	0	1
MINIMUM MAXIMAL MATCHING (MMM)	vertex deletion	0	1
CHROMATIC NUMBER (CRN)	vertex deletion	0	1
DOMINATING SET (DS)	edge deletion	1	0
MAX-CUT (MC)	edge deletion	1	0

Table 2.1. A condensed version of Table 2 from [10] showing problems for which structural rounding results in approximation algorithms for graphs near the parameterized structural class treewidth w where the problem has a $\rho(w)$ algorithm. We also give the associated stability (c') and lifting (c) constants which are class-independent. We remark that vertex* is used to emphasize the rounding process has to pick the set of annotated vertices in the edited set carefully to achieve the associated stability and lifting constants.

2.3.2 Prior Work

Structural rounding has previously been practically evaluated only once, in [21]. This paper used structural rounding to approximate VERTEX COVER on near-bipartite graphs under the edit operation of vertex deletion. The authors edit heuristically using a greedy method, solve VERTEX COVER exactly using the Hopcroft-Karp algorithm [17], then lift using a range of novel strategies. Although its worst-case solution quality guarantee was a 2-approximation, structural rounding was shown to consistently give better approximations than VERTEX COVER's best performing traditional 2-approximations [27][29] on a significantly sparse real-world corpus of 130 graphs [1]. This performance is depicted in **Figure 2.6**[1] on the following page.

[1]This figure was used with permission from the original authors.

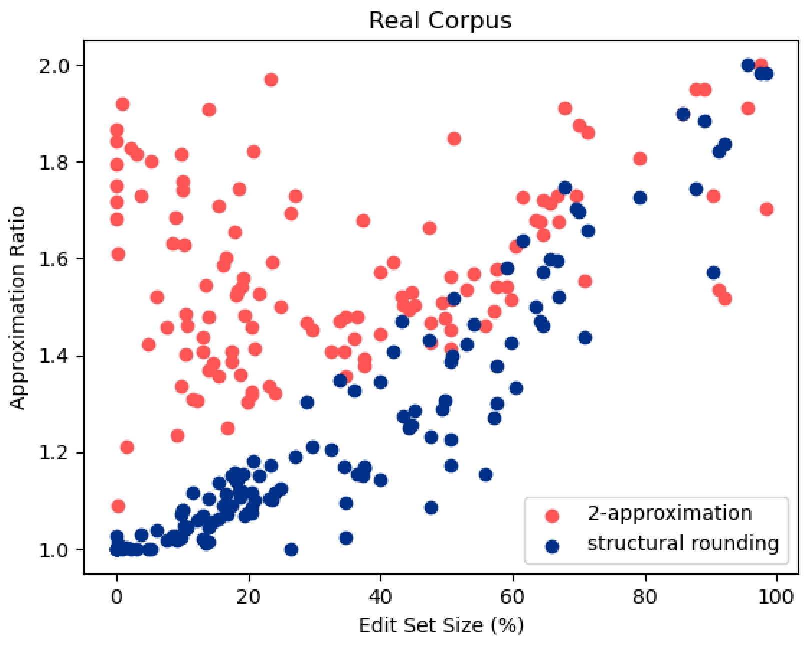

Figure 2.6. Figure 7 from [21] comparing the best approximation ratios achieved by 2-approximations (in red) versus structural rounding (in blue) on graphs from a real-world corpus [1] from various domains including networks, physical infrastructure, and more.

Figure 2.6 shows that for near-bipartite graphs with small edit sets, structural rounding greatly outperformed 2-approximations for VERTEX COVER. What's surprising to us is that structural rounding also worked well with larger edit sets, roughly converging to the solution quality of the best 2-approximations after edit set sizes went above 60%. In terms of runtime, however, structural rounding was about 3.5x slower than the 2-approximations. Intuitively, this is possibly the result of structural rounding making three passes over the vertices, one for each step, compared to the 2-approximation algorithms which make only a single pass.

Initially, a couple of different heuristic editing approaches were tried as shown in **Figure 2.7**[2]. It was found that greedily constructing two maximal independent sets (Greedy IS) generally outperformed using breadth-first search (BFS) to find an edit set. Omitting the details of these methods, what's important to us is that this showed using the editing strategy which produced the smallest edit sets correlated with producing the best overall approximations.

[2]Figure created by Brian Lavallee.

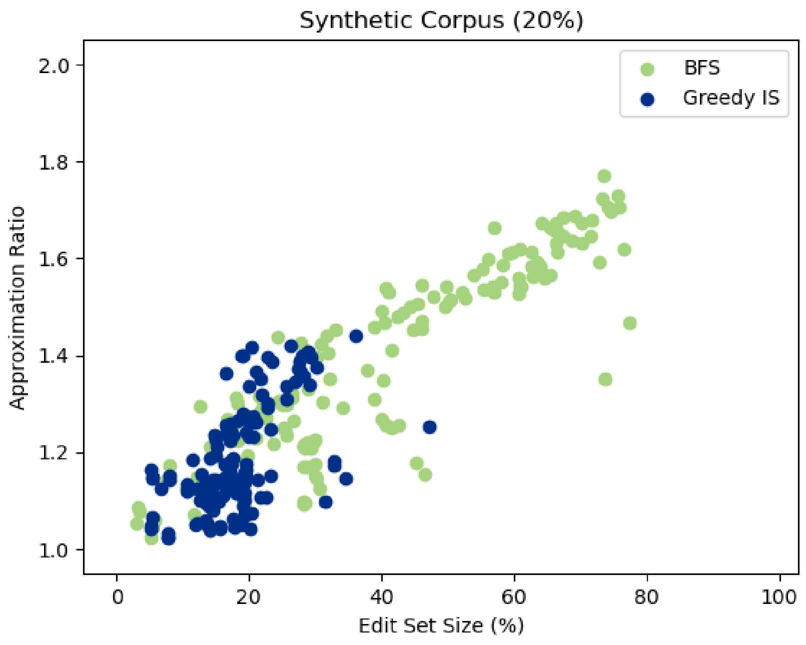

Figure 2.7. Data from [21] comparing the solution quality of structural rounding when used with two competing heuristic editing methods: Breadth-first search (BFS) and a greedy maximal independent set approach (Greedy IS). The 100K edge corpus was generated to have edit set sizes of 20%. Found edit set sizes differ as a result that heuristics provide no guarantees on producing the same edit set as the one prescribed by the generator.

These results were promising and highlighted the importance of empirical evaluation. There were drawbacks, however. Bipartite is a very structured class, meaning it's less likely that general real-world networks will be close; though this may not be an issue, considering structural rounding's shown solution quality on graphs with larger edit sets. As a final test in [21], structural rounding was compared against a simple greedy heuristic for Vertex Cover which creates a cover by repeatedly adding the vertex that covers the most uncovered edges until every edge is covered. Disappointingly, this heuristic produced similar approximation ratios to structural rounding with runtimes comparable to the 2-approximations.

As structural rounding's first (and only other) evaluation, this paper left a myriad of unanswered questions and unexplored directions. Principally, we wanted to know whether these findings on edit set sizes are specific to Vertex Cover on near-bipartite graphs or if they are representative of structural rounding in general.

CHAPTER 3

METHODS

To flesh out the practical evaluation of structural rounding, we consider approximating DOMINATING SET on near-bounded treewidth graphs. In this chapter, we define our primary research questions and outline our experiment design and data.

3.1 Problem Statement

This thesis poses and evaluates two hypotheses. Prior work [21] showed that using the editing method which produces the smallest edit sets leads to better overall approximations when using the structural rounding framework. We evaluate whether this still holds for a different problem/structural class as a step towards resolving this claim.

Hypothesis 1. Smaller edit sets will always produce better solution quality for structural rounding.

By working with near-bounded treewidth graphs, our work is the first application of structural rounding using a parameterized class. This necessitates choosing a treewidth value to edit to. We note that, for DOMINATING SET, the solve step has runtime exponential in the edited graph's treewidth (see Subsection 3.2.2) while the edit step is polynomial (see Subsection 3.2.1). Further, editing to smaller treewidth will produce larger edit sets regardless of the editing method used (and thus more of the solution will come from lifting). If Hypothesis 1 is true, then selecting a target treewidth will be a trade-off between runtime and solution quality. Our second hypothesis below outlines our expectations for this predicted trade-off.

Hypothesis 2. Editing to the minimum allowed target treewidth results in the fastest runtimes for structural rounding whilst editing to the largest target treewidth feasible (given imposed runtime constraints) produces the best solution quality.

3.2 Our Approach

DOMINATING SET is in FPT when parameterized by treewidth, making bounded treewidth a natural selection for our structured class. Further, bounded treewidth has an editing algorithm defined in [10]. In this section, we detail each step of the structural rounding pipeline used in our experiments.

3.2.1 Editing

We edit to treewidth w using the vertex deletion algorithm (Algorithm 3) from [10] shown in **Algorithm 1** below. This algorithm gives a bicriteria $(O(\log^{1.5} n), O(\sqrt{\log w}))$-approximation, meaning the number of edits is at most $O(\log^{1.5} n)$ times the optimum, and the edited graph has treewidth at most $O(w\sqrt{\log w})$. For small w, this algorithm produces unacceptably wide decompositions due to a built-in constant factor. In order to produce practical results, we remove it from the early-out criteria (on line 2 from **Algorithm 1**). This destroys the approximation guarantee for treewidth. We further depart from the original algorithm by modifying it using one of two heuristic approaches: **greedy degree** and **balanced separator**. These are employed to improve runtime, since the original algorithm was impractically slow even with the change to early-out criteria in place. We consider *two* approaches to compare how their edit set sizes relate to overall solution quality.

Algorithm 1 `TreeWidthNodeEdit` $(G = (V, E), w)$

```
1: $t \leftarrow$ compute $\mathrm{tw}(G)$ by invoking the algorithm of [5] together with [14]
2: if $t \leq w$ then    /* In [10] this is $t \leq 32 \cdot w\sqrt{\log w}$ */
3:     return $\emptyset$
4: else
5:     $S \leftarrow$ compute a $(\frac{3}{4})$-vertex separator of $G$ by invoking the algorithm of [14]
6:     let $G[V_1], \cdots, G[V_l]$ be the connected components of $G[V \backslash S]$
7:     return $(\cup_{i \leq l}$ TreeWidthNodeEdit $(G[V_i], w)) \cup S$
```

Greedy degree replaces the original component used to find approximate tree decompositions [5] (on line 1 from **Algorithm 1**) with the fastest heuristic available from [6], detailed in Chapter 4. To better describe what changes in the second heuristic, we define a *vertex separator* as a set of vertices which, if removed, separate the graph into disjoint components. We follow [5] and formally describe this structure on the next page.

α-Vertex Separator

Definition: Given a graph $G = (V, E)$ and a set of vertices $W \subseteq V$, a set $S \subseteq V$ is an α-vertex separator of G if at most $\alpha \cdot |W|$ vertices from W are in each connected component of $G[V \backslash S]$.

Our balanced separator[1] approach simply replaces the vertex separator algorithm from [14] (on lines 1 and 5 from **Algorithm 1**) with a heuristic to find *balanced* separators based on [3]. Balanced means that the disjoint subgraphs produced by removing the separators have similar size. To further save time and memory, we implement[1] [5] iteratively.

Greedy degree also employs this separator based on [3], and is thus a modified version of balanced separator which employs a different tree decomposition. We note that even in the absence of our change to the early-out criteria (on line 2 from **Algorithm 1**), using either of these heuristics to improve runtime still removes the approximation guarantees.

3.2.2 Solving

Recall from Subsection 2.3.1 that DOMINATING SET is not stable for structural rounding with respect to vertex deletions. We instead approximate ANNOTATED DOMINATING SET, which is shown in [10] to work with structural rounding under vertex* deletions (see **Table 2.1**). We follow [10] and refer to graph optimization problems where the input consists of both a graph and a subset of annotated vertices/edges as *annotated* problems. Such subsets are carefully chosen (in the edited graph) to guarantee small stability and lifting constants.

ANNOTATED DOMINATING SET (ADS)

Input:	A graph $G = (V, E)$ and a subset of vertices $B \subseteq V$.
Problem:	Find a set of vertices $X \subseteq V$ such that every vertex in B is either in X or is adjacent to a vertex in X.
Objective:	Minimize $|X|$.

ANNOTATED DOMINATING SET, synonymous with SUBSET DOMINATING SET in [15] and others, is equivalent to DOMINATING SET when $B = V$. Many approaches to exactly solving ANNOTATED DOMINATING SET use dynamic programming [2][4][30]. We imple-

[1]Implementation primarily developed by Brian Lavallee.

ment[2] a dynamic programming algorithm based on [2] which achieves a relatively fast runtime of $O(4^k n)$ when G has treewidth at most k.

3.2.3 Lifting

To lift our partial ANNOTATED DOMINATING SET from the edited graph, we initialize our annotated subset B as the vertices not dominated by the partial solution. We then use a greedy heuristic to select additional dominators. Since B is comprised of exactly the vertices not dominated by the partial solution, the union of the partial solution and the dominators chosen by the greedy lifting algorithm produce our overall solution to DOMINATING SET.

3.3 Experiments

We perform structural rounding on a synthetic corpus of 8,100 sparse graphs created by permuting the following sets of parameters from **Table 4.1**, shown in **Table 3.1** below. Note that we restrict our attention to graphs with $n = 10,000$, and produce 5 graphs for each configuration via seeding. That's 1,620 unique configurations.

Parameter	**Values**
n	$\{10,000\}$
k	$\{2,3,4,5,6,7\}$
dropout	$\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$
editset	$\{100, 500, 1K, 2.5K, 4K\}$
editset density	$\{0.001, 0.005, 0.01\}$
bipartite density	$\{0.001, 0.005, 0.01\}$
seed	$\{0,1,2,5,6\}$

Table 3.1. The input parameters used in the generation of our synthetic 8,100 graph corpus.

We generate $1,350$ graphs for treewidth values $k \in \{2, 3, \ldots, 7\}$. For each k, we tested editing to target treewidth values $w \in \{k, k-1, \ldots, 2\}$ using both greedy degree and balanced separator methods. This results in $56,700$ total runs of structural rounding. Notice we sparsify the structural class up to 60% and that our editsets contain up to 40% of the overall graph. Notice also that our editset density and bipartite density settings

[2]Implementation primarily developed by Brian Lavallee.

are considerably sparse. These parameters have been selected to mimic the behavior of real-world networks.

DOMINATING SET admits an $O(\log \Delta)$-approximation[3] [8] which is our competitor for structural rounding in this setting. Similar to lifting, this approximation greedily selects maximum degree vertices as dominators. We use this approximation as a benchmark to measure structural rounding's relative performance, in terms of both solution quality and runtime, since it is a traditional choice for solving DOMINATING SET on general graphs.

[3]Implementation primarily developed by Brian Lavallee.

CHAPTER 4

IMPLEMENTATION AND ENGINEERING

We now describe two major engineering contributions made in support of answering the questions outlined in Chapter 3. We begin by describing the existing implementation of structural rounding for VERTEX COVER on near-bipartite graphs in an open-source C++ library. We then present a new synthetic graph generator, necessitated by our choice of bounded treewidth as the target class. Finally, we describe selection and implementation of several greedy heuristic strategies from [6] for finding tree decompositions. We denote software classes and functions using `monospace font`; functions are postfixed with `()`, regardless of parameters.

4.1 Structural Rounding Graph Library

Structural rounding is implemented in the publicly available open-source library at `https://github.com/TheoryInPractice/structural-rounding`. All work in this thesis was built on top of version 2.0 [16]. This section describes the existing structures and classes available in this version.

Initially developed for approximating VERTEX COVER in [21], the library includes implementations of several approximate and exact solvers for VERTEX COVER as well as a variety of lifting strategies. The codebase also includes multiple algorithms for vertex deletion to bipartite. The underlying data structures provided here that are relevant to us are `Graph`, `Set`, and `Map`. Dealing with typically sparse data, `Graph` uses adjacency lists, i.e., `Map<Set> adjlist`, to store its connections. `Graph` natively supports common operations such as: adding and removing edges, removing vertices, getting the degree of a vertex, determining if a vertex is contained in a graph, determining if two vertices are adjacent, and getting the neighbors of a vertex.

4.2 Synthetic Graph Generator

In order to assess the impact of target treewidth w on solution quality and runtime, we needed to produce graphs with a bounded distance to treewidth k. To meet this demand, we present a new synthetic graph generator which allows tuning of the parameters shown in **Table 4.1** below.

Parameter	**Variable**	**Type**	**Description**
n	`n`	`int`	Number of vertices in the partial k-tree.
k	`k`	`int`	Upper bound of k in the partial k-tree.
dropout	`spar`	`float`	Sparsity of the partial k-tree.
editset	`ess`	`int`	Size of the edit set.
editset density	`esd`	`float`	Density of the edit set.
bipartite density	`esdbw`	`float`	Density between the edit set and partial k-tree.
seed	`seed`	`int`	Seed for the random number generator.

Table 4.1. Variable names, types, and descriptions of the graph generator parameters.

The first step of the graph generator is to create the *partial k-tree* which will serve as our underlying structure within the overall graph. A k-tree is a maximal graph for a given treewidth k, meaning not another edge can be added without increasing the graph's treewidth [26]. Partial k-trees are simply subgraphs of k-trees and also have treewidth at most k, since treewidth is subgraph-closed.

We start this process by first generating a tree T of size n, using the random tree generator `igraph_tree_game()` from igraph [18]. We define a method `tree_to_directed()`, shown in **Algorithm 2** below, which converts undirected trees to directed trees with a user-specified root node in $O(n\Delta)$-time and $O(n)$-space.

Algorithm 2 `tree_to_directed` $(T, root)$

```
1: $T_d \leftarrow$ empty tree
2: stack $S \leftarrow \{root\}$
3: **while** $S \neq \emptyset$ **do**
4:     $u = S.\text{pop}()$
5:     **for** each $v \in N_T(u)$ **do**
6:         **if** first time seeing $v$ **then**
7:             add directed edge $(u, v)$ to $T_d$
8:             $S.\text{push}(v)$
9: **return** $T_d$
```

Letting $T_d =$ `tree_to_directed`$(T, root)$, we initialize G to be a directed clique of size k. We initialize *list* to be the vertices in G. We then define a recursive method `make_ktree()`, shown in **Algorithm 3** below, which alters these two data structures to convert the input clique G into a directed k-tree in $O(n(n+\Delta))$-time.

Algorithm 3 `make_ktree` (T_d, G, *list*, *root*)

```
1: add a new vertex $x$ to $G$
2: for each $u \in list$ do
3:     add directed edge $(u, x)$ to $G$
4: remove random item from $list$
5: add $x$ to $list$
6: $children \leftarrow$ outgoing $N_{T_d}(n)$
7: for each $c \in children$ do
8:     make_ktree$(T_d, G, list, c)$
```

Once we've transformed our k-clique G into a k-tree, we "sparsify" G uniformly to transform it into a partial k-tree. We define a method `sparsify()`, which removes every edge with a given probability in $O(mn)$-time. Using igraph's random graph generator required the use of their data structures. The pitfall of this engineering decision is that igraph's graph class does not provide an easy way to access edge lists. We initialize G as directed solely to optimize this method to loop over vertices and visit every edge only once, rather than n times. After we sparsify G, we convert it to undirected, trivially, using `igraph_to_undirected()`. Our partial k-tree, now simple, is complete.

The second step of the graph generator is to create the prescribed edit set to be removed exactly from the overall graph in order to obtain structure. We define a method `add_edit_set()`[1] which creates a random graph *ER* using the *Erdös-Renyi Random Graph Model* defined below, and attaches it to our now partial k-tree G.

Erdös-Renyi Random Graph Model $G(n, p)$

Definition: A random graph is constructed by adding each potential edge of an n-vertex graph uniformly at random with probability p.

[1]Implementation primarily developed by Madison Cooley.

We determine the size and density of ER using our input editset size and density parameters, respectively. To attach ER to G, a random bipartite graph B is made by adding edges uniformly at random between ER and G such that B's density is roughly equivalent to the input bipartite density parameter, and B's vertices have coefficient of variation 0.5, or about even. Thus, we have connected our partial k-tree and our edit set to create a graph which is a known distance from treewidth k.

4.3 Greedy Elimination Ordering Algorithms

Treewidth can be found via "elimination orderings" of vertices that can be used to form tree decompositions. Here, we discuss some heuristics for finding orderings for giving small treewidth. We list the relevant algorithm from [6] below (see **Algorithm 4**) used to created orderings of vertices. For our implementation, each iteration of the loop in **Algorithm 4** also creates a bag containing $N(v_i)$ which is added to the resulting tree decomposition.

Algorithm 4 `GreedyDegree` $(G = (V, E))$

```
1: $H = G$
2: for $i = 1$ to $n$ do
3:     Choose a vertex $v$ of minimum degree in $H$
4:     Let $v$ be the $i$th vertex in ordering $\pi$
5:     $H \leftarrow H[V \backslash v]$
6: return ordering $\pi$.
```

This algorithm represents the greedy heuristic we defined in Subsection 3.2.1 (greedy degree). Originally designed by Markowitz [23], this was the fastest performing heuristic from the suite Bodlaender et al. introduced in [6]. This method was selected in an effort to improve the runtime of our editing step.

We initially tried an additional heuristic from [6], `GreedyFillIn`, where line 3 of **Algorithm 4** instead selected a vertex that has the smallest number of pairs of non-adjacent neighbors. Though `GreedyFillIn` was shown in [6] to return slightly lower treewidth bounds on average, computationally finding these pairs proved to have untenable runtimes on larger graphs. Since all other heuristics provided from [6] are slower than this method, we did not evaluate them in this work.

4.4 Evaluation Hardware Specifications

All experiments were performed on identical hardware; each server had two over-clocked 20-core Intel Xeon Gold 6230 CPUs (2.10 GHz) and 192 GB DDDR4 memory. The servers ran Linux kernel 3.10.0-957.27.2.el7.x86_64 and were hosted on the Center for High Performance Computing's (CHPC) Notchpeak cluster. The code and experiments are written entirely in C++11 and run using GNU compiler version 10.2.0 on CentOS Linux 7.6.1810 AMD64. We plan to release all code open source under a BSD 3-clause license.

CHAPTER 5

RESULTS

In this chapter, we describe the results of both experiments outlined in Chapter 3. To improve readability, we denote balanced separator as `Sep` and greedy degree as `Deg` throughout. Input values from the synthetic graph generator are abbreviated to match their associated parameter names from **Table 4.1**.

5.1 Editing Heuristics

We first evaluated our editing heuristics to test Hypothesis 1. From our runs, we observe that `Deg` consistently found smaller edit sets than `Sep` on all graphs from our corpus as shown in **Figure 5.1** on the next page. We note that prescribed editsets are only valid when target treewidth is equal to generated treewidth (i.e., $w = k$), since the editset is generated to complement a partial k-tree. We observe that the difference between procured edit set sizes was largest when target treewidth was equal to generated treewidth. This indicated to us that `Deg` was more proficient at identifying the editset prescribed by the generator.

Despite the difference in edit set sizes, the heuristics had similar solution qualities. To measure their performance, we define *performance ratio* (PR) to be the size of the DOMINATING SET found using a structural rounding solution relative to the size of the DOMINATING SET found using the competing $\log \Delta$-approximation (i.e., $PR_{\texttt{Deg}} = |X_{\texttt{Deg}}|/|X_{\log \Delta}|$). For example, a PR < 1.0 implies that the structural rounding method used found a smaller DOMINATING SET than the $\log \Delta$-approximation (thus, lower PR is better). **Figure 5.1** shows us that both methods had similar performance ratios and that `Deg` performs poorly in only a small number of graphs.

Hypothesis 1 conjectured that smaller edit sets lead to better overall solution quality for structural rounding in any setting. From our testing, we see that this is false and that larger edit sets from `Sep` performed equally as well—even better in certain situations. This

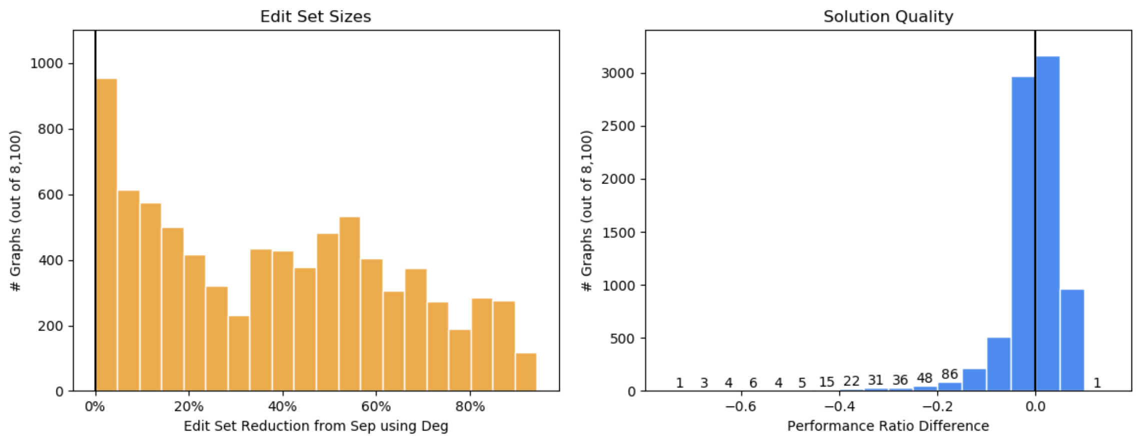

Figure 5.1. At left (orange) we plot the distribution of the reduction in procured edit set size using `Deg` instead of `Sep` (i.e., $(|\texttt{Sep}\text{ edit set}| - |\texttt{Deg}\text{ edit set}|)/|\texttt{Sep}\text{ edit set}|$). We normalize by the size of `Sep`, so the difference represents how much of `Sep` has been removed using `Deg`. This indicates all edit sets found by `Sep` were larger than those found by `Deg`. At right (blue) we plot the distribution of performance ratio differences (i.e., $\text{PR}_{\texttt{Sep}} - \text{PR}_{\texttt{Deg}}$). Graphs where `Deg` found better solutions are to the right of 0, and graphs where `Sep` found better solutions are to the left of 0.

is in direct opposition with the results of [21].

5.2 Target Treewidth Manipulation

To test Hypothesis 2, we needed to evaluate how `Deg` and `Sep` performed in terms of both runtime and solution quality when editing to vary the target treewidth w. We observe that, for both methods, lowering w increased average edit set size but improved average solution quality. An increase in edit set size is consistent with our intuition that achieving smaller treewidth would require removing more vertices from the graph. What's surprising is that larger edit sets lead to better solution quality. Using `Deg`[1] we show how edit set sizes and solution qualities are affected as w varies in **Figure 5.2** on the following page.

We also considered the effect of w on runtime. As we predicted, runtimes for the solve step dramatically improved as we lowered the target treewidth w as shown in **Table 5.1**. Since solving DOMINATING SET was the most costly step in both methods, editing to smaller w decreased our average total runtimes. Since, smaller target treewidth also led

[1]Similar results held for `Sep`; these are omitted in the interest of clarity of presentation.

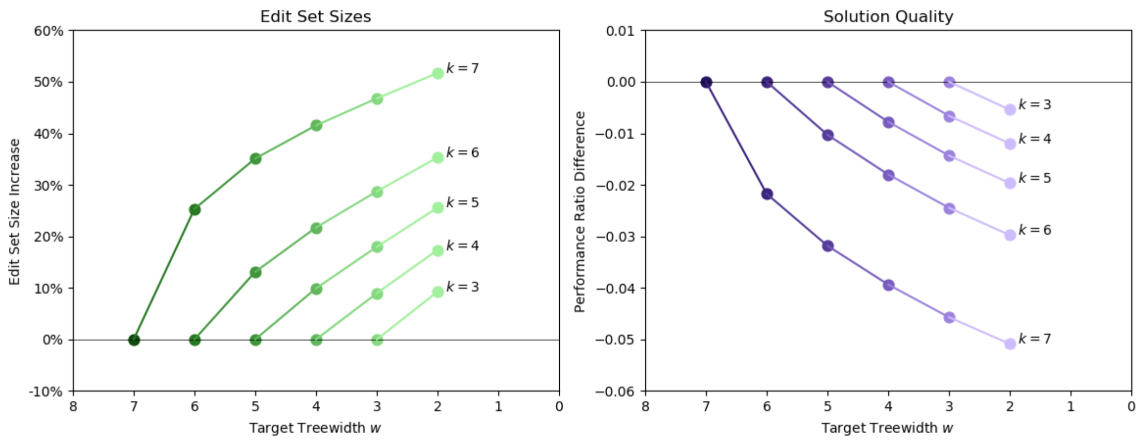

Figure 5.2. At left (green) we plot the average edit set size increase when editing from generated treewidth k to target treewidth w using `Deg`. We observe edit set size increases as w decreases. At right (purple) we plot the average solution quality improvement in the same setting. We observe solution quality improves as w decreases.

to better average solution quality (**Figure 5.2**), there was no observed trade-off between runtime and solution quality. This partially contradicts Hypothesis 2.

w	`Deg`	`Sep`
7	12.23s	4.92s
6	1.00s	0.59s
5	0.20s	0.15s
4	0.06s	0.05s
3	0.03s	0.03s
2	0.02s	0.01s

Table 5.1. Average runtimes for the solve step of structural rounding on DOMINATING SET using `Deg` and `Sep` for all graphs with generated treewidth $k = 7$.

To directly compare the solution qualities of each method against their found edit set sizes, we plot the raw data from our runs in **Figure 5.3** on the next page. Notice the "spikes" in performance ratio that `Deg` admits for a few graphs. These graphs correspond to the bars in the right-hand panel of **Figure 5.1** on the preceding page where the performance ratio difference was significantly negative. They occur in some situations when both $w \approx k$ and the prescribed editset is high (i.e., 25-40%). Notice that runs in which w is smaller are displayed over their larger $\{w+1, w+2, \ldots, 7\}$ counterparts. In this way,

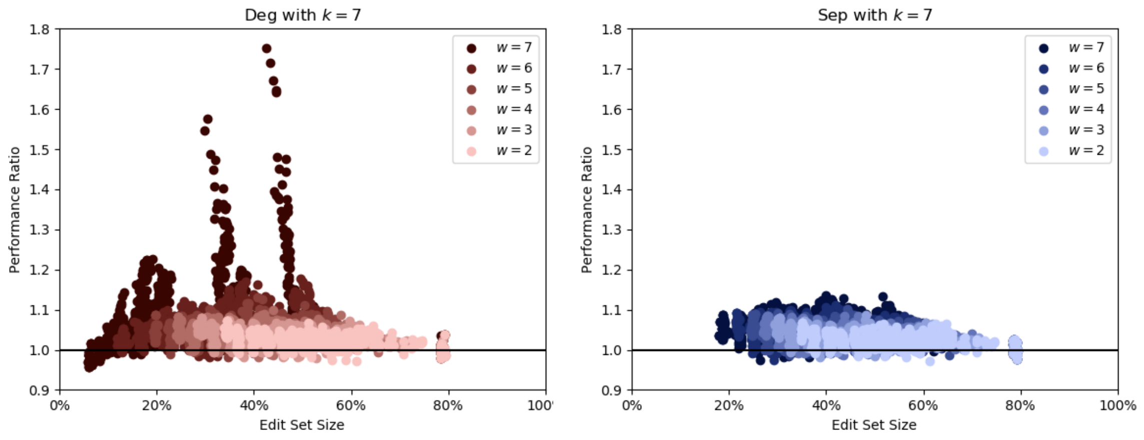

Figure 5.3. Procured edit set sizes compared against performance ratios for all graphs with generated treewidth $k = 7$, using `Deg` in the left plot (red) and `Sep` in the right (blue). We observe that `Deg` performed especially poorly in configurations where both the generated treewidth and prescribed editset were large.

Figure 5.3 provides a crude visualization of the observation lowering target treewidth w improves the worst case bounds on solution quality and increases edit set sizes.

CHAPTER 6

CONCLUSION

Our results demonstrated a setting in which structural rounding performed better as procured edit set sizes grew. This went against our intuition that solving DOMINATING SET exactly on a larger portion of the graph would result in smaller overall solution sizes. We believe this may be explained in part by the necessity of solving ANNOTATED DOMINATING SET. Since annotated vertices are optional to dominate in the partial solution, it's conceivable that this has a greater effect on solution quality than edit set size. Most significantly, these results give credence to the claim that smaller edit sets do not always lead to better solution quality for structural rounding. These results contrast sharply with those from the framework's first practical evaluation in [21].

In the future, we wish to see how these results scale to corpuses of larger graphs, including real-world networks. If a domain exists in which there is a natural trade-off between runtime and solution quality as we vary the target parameterized class, it would be interesting to explore the ideal target treewidth to optimize for both. Further, we note that all configurations of structural rounding we present have solution quality similar to the $\log \Delta$-approximation, but have considerably longer runtime. Identifying parameterized settings in which structural rounding decisively improves solution quality or is more competitive with respect to runtime would be of significant practical interest.

REFERENCES

[1] network-corpus. github.com/microgravitas/network-corpus, February 2018.

[2] J. Alber and R. Niedermeier. Improved tree decomposition based algorithms for domination-like problems. In *Latin American Symposium on Theoretical Informatics*, pages 613–627. Springer, 2002.

[3] H. Y. Althoby, M. D. Biha, and A. Sesboüé. Exact and heuristic methods for the vertex separator problem. *Computers & Industrial Engineering*, 139:106135, 2020.

[4] H. L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In *International Colloquium on Automata, Languages, and Programming*, pages 105–118. Springer, 1988.

[5] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *J. Algorithms*, 18(2):238–255, 1995.

[6] H. L. Bodlaender and A. M. Koster. Treewidth computations i. upper bounds. *Information and Computation*, 208(3):259–275, 2010.

[7] J. Chen, I. A. Kanj, and G. Xia. Improved parameterized upper bounds for vertex cover. In R. Královič and P. Urzyczyn, editors, *Mathematical Foundations of Computer Science 2006*, pages 238–249, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[8] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

[9] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery.

[10] E. D. Demaine, T. D. Goodrich, K. Kloster, B. Lavallee, Q. C. Liu, B. D. Sullivan, A. Vakilian, and A. van der Poel. Structural rounding: Approximation algorithms for graphs near an algorithmically tractable class. *27th Annual European Symposium on Algorithms (ESA)*, 2019.

[11] R. G. Downey, M. R. Fellows, C. McCartin, and F. Rosamond. Parameterized approximation of dominating set problems. *Information Processing Letters*, 109(1):68–70, 2008.

[12] B. Escoffier and V. T. Paschos. Completeness in approximation classes beyond apx. *Theoretical Computer Science*, 359(1):369–377, 2006.

[13] S. Eubank, V. S. A. Kumar, M. V. Marathe, A. Srinivasan, and N. Wang. Structural and algorithmic aspects of massive social networks. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, page 718–727, USA, 2004. Society for Industrial and Applied Mathematics.

[14] U. Feige, M. Hajiaghayi, and J. R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38(2):629–657, 2008.

[15] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4):374–387, Apr 1998.

[16] M. Hooper, B. Lavallee, C. Perschon, B. D. Sullivan, and A. van der Poel. Structural-rounding: Version 2.0, Apr 2020.

[17] J. Hopcroft and R. Karp. An n5/2 algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2:225–231, 1973.

[18] T. igraph Core Team. igraph, Apr. 2020.

[19] V. Kann. *On the Approximability of NP-complete Optimization Problems*. PhD thesis, Royal Institute of Technology, S-100 44 Stockholm, Sweden, 1992.

[20] R. M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.

[21] B. Lavallee, H. Russell, B. D. Sullivan, and A. van der Poel. Approximating vertex cover using structural rounding. In *2020 Proceedings of the Twenty-Second Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 70–80. SIAM, 2020.

[22] A. Magen and M. Moharrami. Robust algorithms for on minor-free graphs based on the sherali-adams hierarchy. In *Proceedings of the 12th International Workshop and 13th International Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, APPROX '09 / RANDOM '09, page 258–271, Berlin, Heidelberg, 2009. Springer-Verlag.

[23] H. M. Markowitz. The Elimination form of the Inverse and its Application to Linear Programming. *Management Science*, 3(3):255–269, April 1957.

[24] T. Milenković, V. Memišević, A. Bonato, and N. Pržulj. Dominating biological networks. *PloS one*, 6(8):e23016–e23016, 2011. 21887225[pmid].

[25] J. C. Nacher and T. Akutsu. Minimum dominating set-based methods for analyzing biological networks. *Methods*, 102:57–63, 2016.

[26] J. Nešetřil. Structural properties of sparse graphs. *Electronic Notes in Discrete Mathematics*, 31:247–251, 2008. The International Conference on Topological and Geometric Graph Theory.

[27] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., USA, 1982.

[28] A. Paz and S. Moran. Non deterministic polynomial optimization problems and their approximations. *Theoretical Computer Science*, 15(3):251–277, 1981.

[29] C. Savage. Depth-first search and the vertex cover problem. *Information Processing Letters*, 14(5):233–235, 1982.

[30] J. M. Van Rooij, H. L. Bodlaender, and P. Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *European Symposium on Algorithms*, pages 566–577. Springer, 2009.

[31] F. Wang, H. Du, E. Camacho, K. Xu, W. Lee, Y. Shi, and S. Shan. On positive influence dominating sets in social networks. *Theoretical Computer Science*, 412(3):265–269, 2011.