Application of Quotient Graphs in Total Domination

Jean-Pierre Appel

Advisor: Dr. Benjamin Coleman
Advisor: Dr. Shannon Talbott
Liaison: Dr. Michael Fraboni

2024

Moravian University
Bethlehem, Pennsylvania

Abstract

Determining the Total Domination Number of a graph is a NP-Hard problem, with the time to generate a solution scaling impractically unless P = NP. This thesis seeks to improve the real-world runtimes of existing Total Domination algorithms by introducing a preprocessing step. We develop a novel similarity measure between vertices, which enables our algorithm to condense graphs while retaining relevant characteristics. Our approach is based on the concept of quotient graphs, but is less restrictive. In the worst case, our algorithm's runtime scales quadratically with graph order, offering a preprocessing step that may enhance existing algorithms.

Acknowledgements

I would like to thank my advisors, Dr. Coleman and Dr. Talbott, for guiding my research and keeping me on track. I also want to express my gratitude for my family, who have supported me unconditionally through the best and worst times. Lastly, I would like to thank the members of the Vargtimmen King Koffee creative workshop; their unique perspectives mean the world to me.

Contents

Chapter 1

Introduction

1.1 Motivation

Electrical networks, plumbing systems, social groups, and other complex systems can be modeled using a graph, which is also known as a network. Graphs model these real world phenomena by representing an entity in the system as a node or vertex. Each vertex in a graph can have connections to other vertices are called edges. The collection of nodes and edges that comprise the graph describe how entities in the original system (power stations, drains, people) relate to one another.

When attempting to model an electrical grid as a graph, one might wonder how to monitor the state of each node. Assume that monitors are placed on nodes, and that each monitor can observe only the states of neighboring nodes, not its own. Given these conditions, we may want to know how many monitors do we need to place so that every vertex is being watched by a

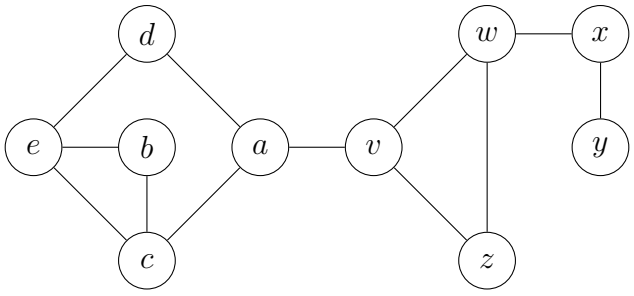


Figure 1.1: An example graph

monitor. Trivially, a monitor could be placed at every vertex in the electrical grid. However, in reality, there are real expenses incurred for each additional monitor we use. So, we want to minimize the number of monitors placed while still ensuring that every vertex is monitored.

Let us start by looking at the simple configuration seen in Figure 1.1. Each vertex represents a power station, while the lines between them represent power connections. Using this model, we can describe placements of monitors which can observe every vertex of the network. Figure 1.2 shows a collection of monitor placements, which is visualized in blue. However, there exists a solution which uses fewer monitors, as shown in Figure 1.3. To generalize the problem, instead of thinking in terms of monitor placements, we can view the problem as selecting a subset of nodes. The problem of choosing a set of nodes from a graph so that every vertex is adjacent to something in the set is known as Total Domination.

What if we consider a larger, more complicated graph as in Figure 1.4? If we were to check every subset of the 22 nodes in Figure 1.4, we would have to check up to $2^{22} = 4,194,304$ different combinations. In general, for a graph on n vertices, we would have to check at most 2^n different sets. While it

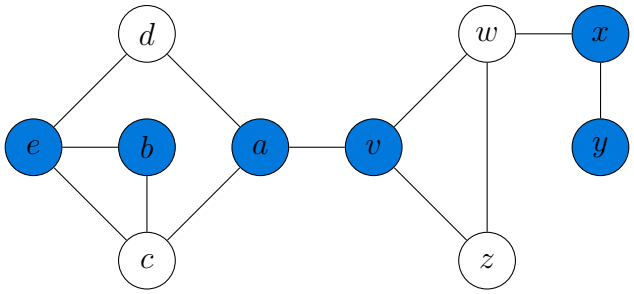


Figure 1.2: A suboptimal solution to Figure 1.1

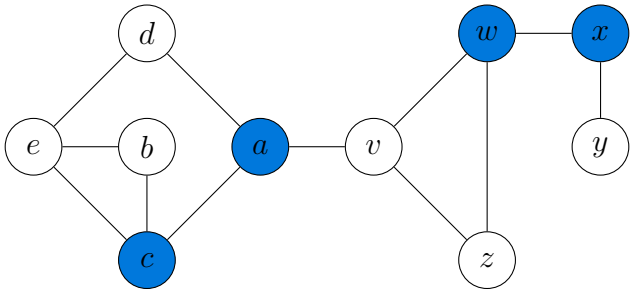


Figure 1.3: An optimal solution to Figure 1.1

would not be too difficult to check an individual configuration, approaching Total Domination Number this way will not work for larger graphs. Figure 1.5 shows one possible set of vertices with a total dominating set. Through methods discussed later, we can remove redundant information to produce the graph in Figure 1.6, which has a similar total dominating set shown in Figure 1.7.

Given some set of vertices in a graph, it is easy to verify if the set forms a total dominating set. On the other hand, it is difficult to generate sets of vertices that we know are optimal total dominating sets. Informally, this suggests that the problem is NP-Complete, which is a class of problems whose solutions can be checked quickly, but it is unknown if optimal solutions can be generated quickly. For smaller graphs, a Total Dominating set can be gener-

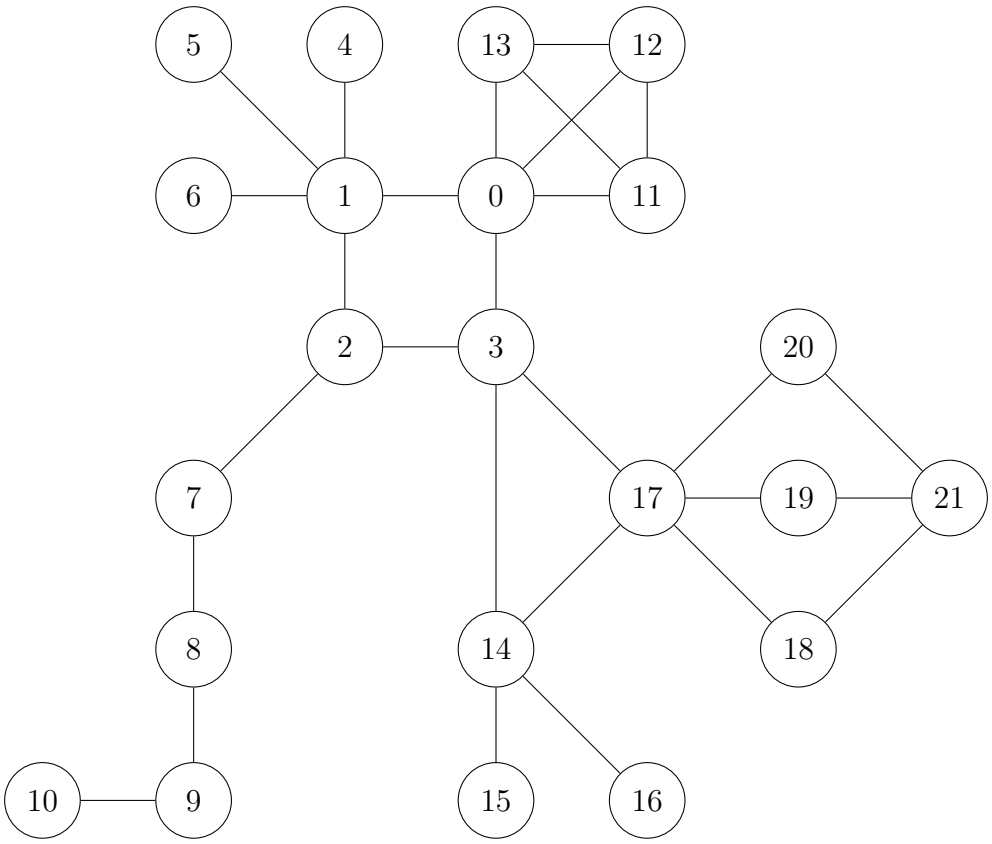


Figure 1.4: An example graph

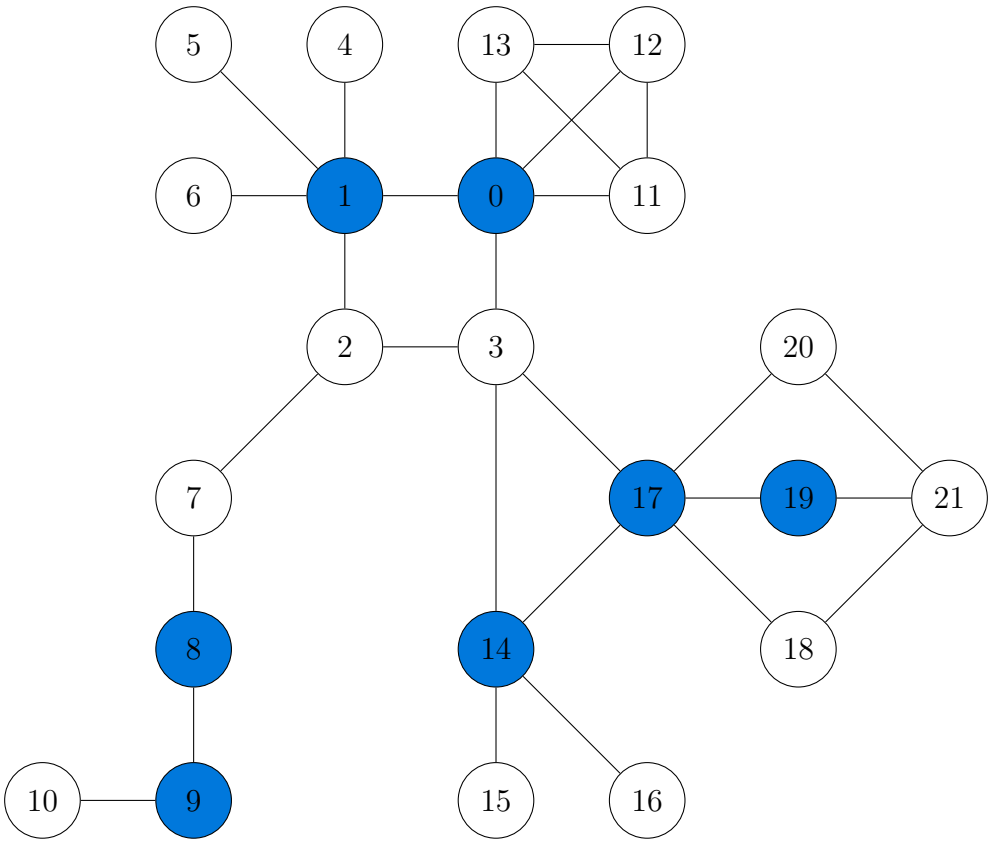


Figure 1.5: Figure 1.4 with a Minimum Total Dominating Set

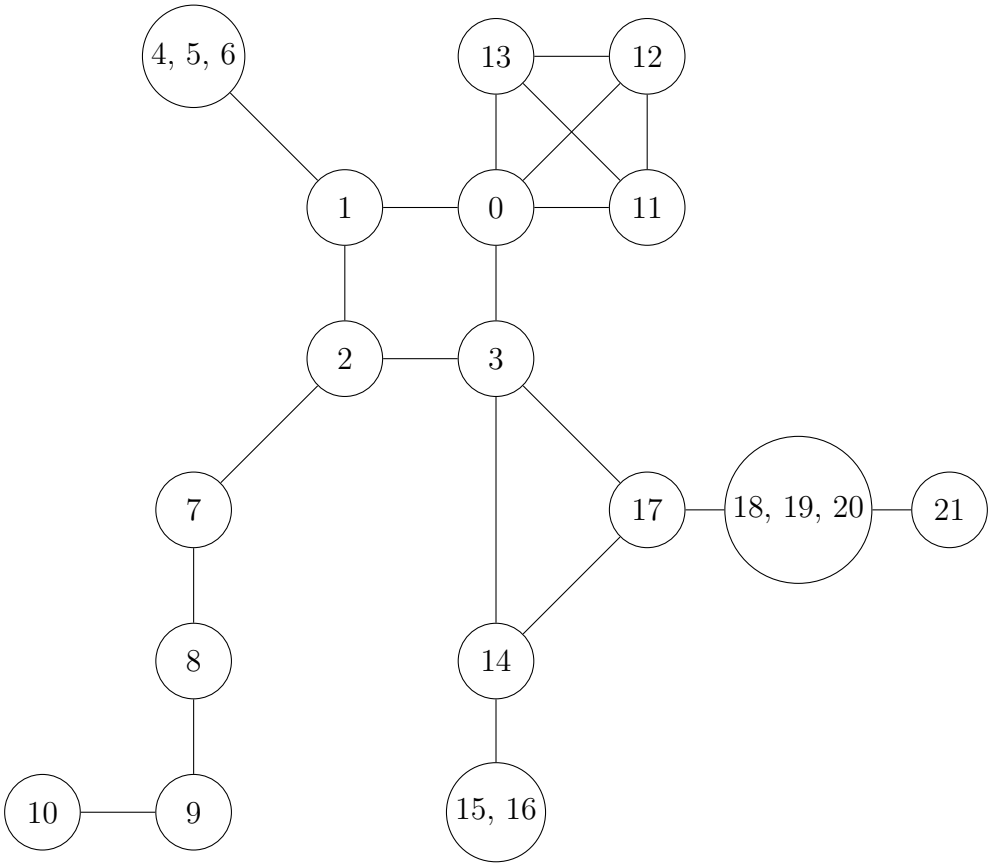


Figure 1.6: The condensed version of Figure 1.4

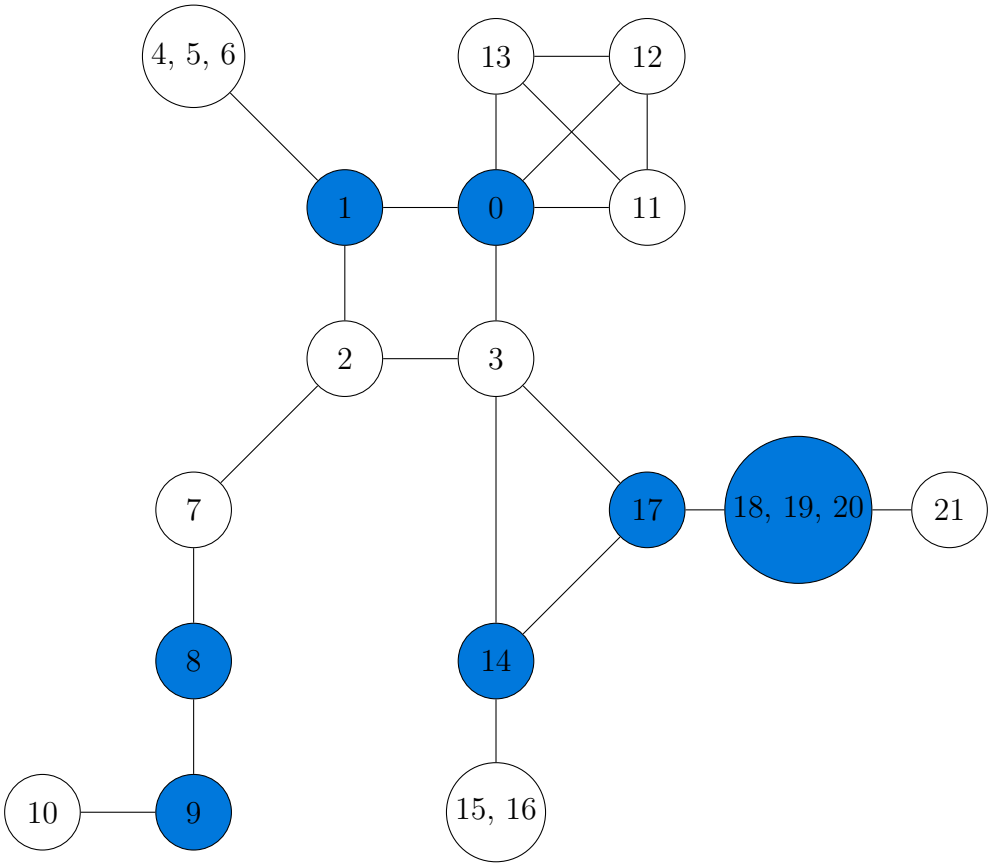


Figure 1.7: Figure 1.6 with a Minimum Total Dominating Set

ated using brute force, but the time to use this method grows exponentially as the graph grows. Ideally a faster algorithm would be written, one that scales better with the size of the graph. Regardless of a better program, we can find places in a graph where placing multiple monitors is redundant. Consider two separate power plants which are both connected directly to the same housing developments. Since the power plants are connected to the same nodes, it makes no difference which power plant we place an external monitor on. By finding situations like these in graphs, we can avoid checking as many possible cases. Reducing the number of cases does not change the exponential scaling of the problem but would reduce real world runtimes.

In this thesis, we explore the NP-Complete problem of finding the Total Domination Number of a graph. We begin in Chapter 2 examining structural properties that affect the choice of a total dominating set and Total Domination Number. Next in Chapter 3 we define a measure of similarity between vertices based off of their neighborhoods. Using this similarity measure, we develop a condensation algorithm in Chapter 4 which reduces the graph while retaining relevant characteristics. Finally, in Chapter 5 we analyze the effectiveness of our algorithm on various graph classes.

Chapter 2

Background

Throughout this paper, we assume a basic knowledge of graphs, sets, and big O notation. For those unfamiliar with either topic, it is suggested to read [3], [5], and [11], before proceeding. Appendix A contains all relevant notation.

2.1 Total Domination

Domination problems are a popular class of problems in graph theory, with real world applications in auditing and the monitoring of electrical networks. Two common questions involving domination are: given a fixed subset of vertices, is every vertex of the graph adjacent to or in the subset, and what is the smallest subset of vertices such that every vertex is adjacent to or in the subset. There are many variants of domination including Roman Domination [10] and Mixed Domination [9].

The variant of domination we study in this paper is Total Domination. The definition requires additional notation. Note that $V(G)$ represents the set

(a) Minimal Total Dominating Set (b) Minimum Total Dominating Set

Figure 2.1: Minimal and Minimum Total Dominating Sets pictured in blue

of vertices of a graph, and $N(S)$ is the set of vertices which are neighbors to the vertices in S. A more precise definition of $N(S)$ is available in Appendix A.

Definition 2.1.1 (Total Dominating Set)**.** For a graph G, a set of vertices $S \subseteq V(G)$ is a *Total Dominating Set* of G if $N(S) = V(G)$.

An important consequence of the definition of a Total Dominating Set is that every vertex in the set must also have a neighbor in the set. We represent this notationally by using the open neighborhood of a vertex $N(v)$, which does not include the vertex itself.

Definition 2.1.2 (Minimal Total Dominating Set)**.** A Total Dominating Set S of a graph G is a *Minimal Total Dominating Set* of G if no proper subset of S is a total dominating set of G.

Definition 2.1.3 (Minimum Total Domintating Set)**.** A Total Dominating Set S of a graph G is a *Minimum Total Dominating Set* of G if S has the same cardinality as the smallest of the minimal total dominating sets.

There are cases when a minimal total dominating set is not a minimum total dominating set. In Figure 2.1a, removing any vertex from the minimal

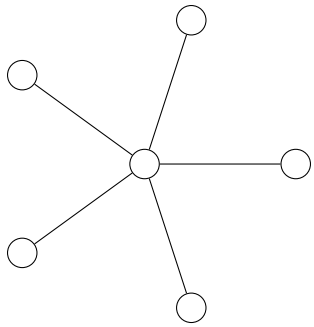


Figure 2.2: A Star graph S_5

total dominating set results in a not dominated vertex. Since Figure 2.1b has a smaller total dominating set than Figure 2.1a, we know the later is minimal but not a minimum.

Definition 2.1.4 (Total Domination Number)**.** The *Total Domination Number* of a graph G, denoted $\gamma_t(G)$, is the size of a Minimum Total Dominating Set.

2.1.1 Star, Complete, and Wheel Graphs

The Star, Complete, and Wheel graph classes are reasonable starting examples for Total Domination because of how small the diameter in each class is. First, let us define each graph class as follows:

Definition 2.1.5 (Star Graph)**.** A *Star Graph* S_n is a connected graph on $n+1$ vertices where there is a vertex $u \in V(S_n)$ such that $\deg(u) = n$, and for all other vertices $v \in V(S_n) \setminus \{u\}$, $\deg(v) = 1$.

Definition 2.1.6 (Complete Graph)**.** A *Complete Graph* K_n is a connected graph on n vertices where every vertex is connected to every other vertex.

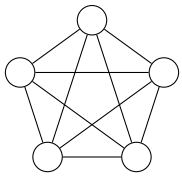


Figure 2.3: A complete graph K_5

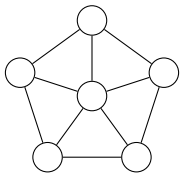


Figure 2.4: A Wheel graph W_5

Definition 2.1.7 (Wheel Graph)**.** A *Wheel Graph* W_n is a connected graph on $n+1$ for $n \geq 3$ consisting of a central vertex $u \in V(W_N)$ such that $\deg(u) = n$. The remaining vertices $v_0, v_1, \ldots, v_{n-1} \in V(W_n) \setminus \{u\}$ form a Cycle, and each vertex v_i is adjacent to u, v_{i-1}, v_{i+1}, where indices are take modulo n.

Under the classical definition of domination, any vertex that is adjacent to all other vertices forms a minimum dominating set. Total domination differs in this regard, requiring at least two vertices to be in the minimum total dominating set. This difference is because domination uses closed neighborhoods while total domination uses open neighborhoods. Total domination in Star, Complete, and Wheel graphs are a trivial but useful exercise to familiarize ourselves with the domination variant. As such, we can determine the Total Domination Number of Star, Wheel, and Complete graphs.

Lemma 2.1.1. *Let $G = (V, E)$ be a connected graph where, for some vertex $u \in V$, we have $N(u) = V \setminus \{u\}$. Then for any vertex $v \in N(u)$, the set $\{u, v\}$ forms a minimum total dominating set of G.*

Proof. Given a connected graph $G = (V, E)$ with a vertex $u \in V$ such that $N(u) = V \setminus \{u\}$, note the only vertex not dominated by $\{u\}$ is the vertex u. So, for any vertex $v \in N(u)$, the set $N(\{v, u\}) = V$. Since the open neighborhood of any vertex cannot include itself, $\gamma_t(G) \geq 2$. Thus, there exists no smaller total dominating set of G. Therefore, $\{u, v\}$ is a minimum total dominating set of G. □

It follows that any graph which meets the conditions for Lemma 2.1.1 have a Total Domination Number of two.

Corollary 2.1.2. *FOr all integers $n \geq 3$, $\gamma_t(K_n) = \gamma_t(W_n) = \gamma_t(S_n) = 2$.*

2.1.2 Path and Cycle Graphs

Paths and Cycles are simple graph classes with more interesting total dominating sets. What makes Paths and Cycles more interesting than Stars, Wheels, and Complete graphs is how the Total Domination Number changes depending on the order of the graphs. We denote a path with n vertices as P_n. As an example, the Paths in Figure 2.5 all have the same Total Domination Number, despite the fact they have differing numbers of vertices. Henning and Yeo noted in [7] that Cycles behave similarly, with a Cycle and Path of the same order having the same Total Domination Number. Below is the Total Domination Number of Paths and Cycles proved in [7].

Theorem 2.1.3 ([7])**.** *For $n \geq 3$, the Total Domination Number of Path*

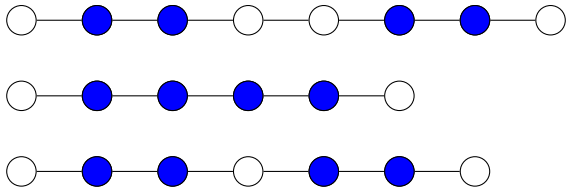


Figure 2.5: Minimum Total Dominating Sets of P_8, P_6, and P_7

graphs P_n and Cycle graphs C_n are

$$\gamma_t(P_n) = \gamma_t(C_n) = \begin{cases} \frac{n}{2} & \text{if } n \equiv 0 \pmod 4 \\ \frac{n+1}{2} & \text{if } n \equiv 1, 3 \pmod 4 \\ \frac{n}{2} + 1 & \text{if } n \equiv 2 \pmod 4 \end{cases}$$

2.1.3 Total Domination in General

In general, finding the Total Domination Number of a graph or graph class is not trivial. It is so difficult that any general algorithm to find the Total Domination Number of a graph is bound to have its run time scale exponentially with graph order given P $\neq$ NP. This is because, in general, the problem of Total Domination Number belongs to a complexity class known as NP-Complete [7].

A naive algorithm that checks all possible subsets for the smallest total dominating set can be written to run in $O(2^n)$ time for graphs on n vertices. Much work has been done for better practical and theoretical performance by Alipour and Salari, van Rooij and Bodlaender, Álvarez-Miranda and Sinnl and others in [1, 13, 2]. Each paper uses a different technique, with [1] using

a distributed algorithm, [13] using measure and conquer, and [2] applying a heuristic to generate approximate solutions. We approach Total Domination Number by creating a preprocessing algorithm which may be combined with any other total domination number algorithm.

2.2 Similarity

Preprocessing a graph to improve the performance of an algorithm can take many forms. We explore one technique which constructs a smaller graph with a similar Total Domination Number to that of the original. To do so, we must decide a way to remove vertices from the input graph while not drastically changing the Total Domination Number.

Observe that for any vertex, there are usually multiple vertices which dominate it. Also, multiple vertices can be dominated by the same vertex. By grouping vertices by how likely they are to dominate the same vertex, we can reason about which vertices must be included in a total dominating set.

In order to group vertices, we need a grouping criteria. Since Total Domination is dependent on the open neighborhoods of a set of vertices, our criteria should share this dependency. For two vertices, we want to quantify how similar the vertices that they can dominate are. This has implications for how likely we are to choose one vertex opposed to the other to be in our dominating set. One possible criteria comes from neighborhood similarity, defined below. In Section 3.2, we explore other possible criteria.

Definition 2.2.1 (Neighborhood Similarity)**.** Given a graph $G = (V, E)$, the

neighborhood similarity of two vertices $u, v \in V$ is given by $\sigma(u, v)$, where

$$\sigma(u, v) = |N(u) \triangle N(v)| = |N(u) \setminus N(v)| + |N(v) \setminus N(u)|.$$

It is important to note there is an equivalent definition for the neighborhood similarity that follows from an alternative definition of the symmetric difference $\triangle$. We provide the alternative definition below.

Definition 2.2.2. For a graph G with vertices u and v, we can alternatively define the *neighborhood similarity* as

$$\sigma(u, v) = |(N(u) \cup N(v)) \setminus (N(u) \cap N(v))|.$$

In simple terms, the neighborhood similarity is the number of vertices adjacent to exactly one of u or v. As an example, Figure 2.6 has two vertices a and b with the same neighborhood. When constructing a total dominating set, we can think of a and b as the same vertex, since any vertex $1, 2, 3$, or 4 which is adjacent to a is also adjacent to b. So, the neighborhood similarity of a and b is $\sigma(a, b) = 0$.

While useful, the neighborhood similarity is not normalized, which makes analysis of similarities within a graph class more difficult. For example, consider the two graphs pictured in Figure 2.7. Note that $\sigma(a_1, b_1) = 5$ while $\sigma(c_1, d_1) = 6$. Both vertex pairs a_1, b_1 and c_1, d_1 have a similar relation, being that of a leaf attached to a vertex of a complete subgraph. We normalize the neighborhood similarity to the interval $[0, 1]$ below.

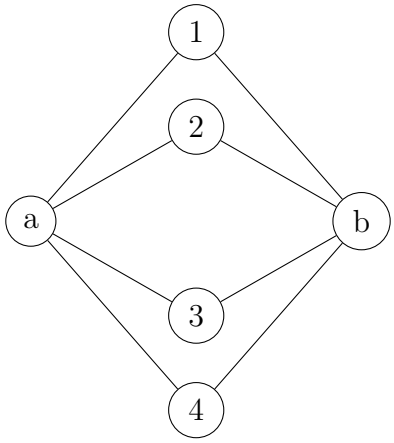


Figure 2.6: Graph with multiple vertices that have identical neighborhoods

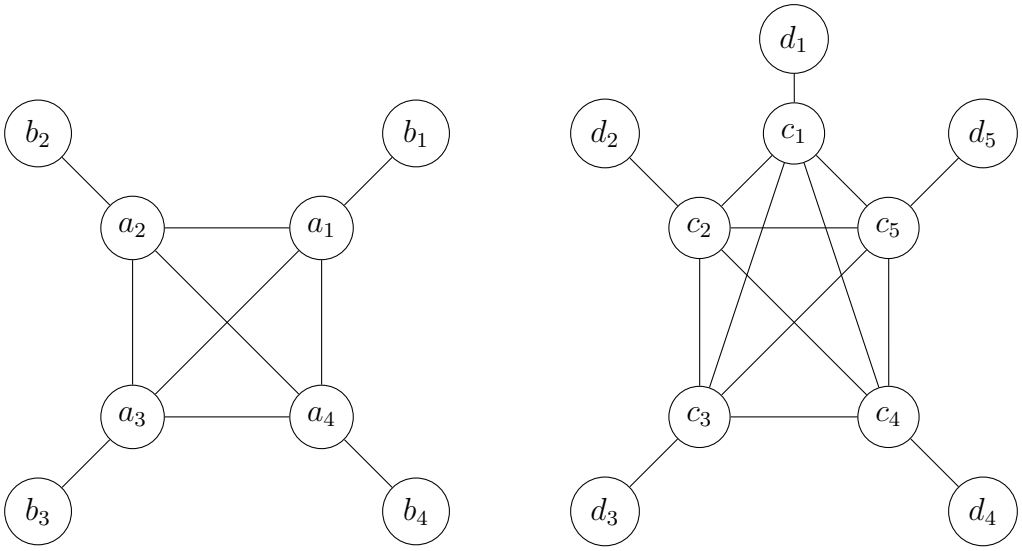


Figure 2.7: Graphs with different similarities but identical normalized similarities

Definition 2.2.3 (Normalized Neighborhood Similarity)**.** For two vertices u and v, we have

$$\hat{\sigma}(u, v) = 1 - \frac{\sigma(u, v)}{\deg(u) + \deg(v)}.$$

The normalized neighborhood similarity ranges from vertices with disjoint neighborhoods ($\sigma = 0$) to vertices with identical neighborhoods ($\sigma = 1$). Let's re-examine Figure 2.7. Recall that $\sigma(a_1, b_1) = 5$ while $\sigma(c_1, d_1) = 6$. However, the normalized similarities are the same, $\hat{\sigma}(a_1, b_1) = \hat{\sigma}(c_1, d_1) = 0$. The normalized neighborhood similarity allows us to make statements about entire graph classes, rather than just specific graphs.

The normalized neighborhood similarity has properties which follow directly from our definitions.

Remark 2.2.1. Some properties of $\hat{\sigma}$ are listed below.

1. $\hat{\sigma}(u, u) = 1$
2. $\hat{\sigma}(u, v) = \hat{\sigma}(v, u)$ Symmetric
3. $\hat{\sigma}(u, v) \in [0, 1]$ Bounded
4. $\hat{\sigma}(u, v) = 1$ iff $N(u) = N(v)$

We will now prove the last property.

Proof. Let u and v be vertices in a connected graph $G = (V, E)$. Assume for the vertices u and v that $\hat{\sigma}(u, v) = 1$. Note the only case when the normalized neighborhood similarity is one is when the neighborhood similarity is zero. So, by the definition of neighborhood similarity, this only occurs when

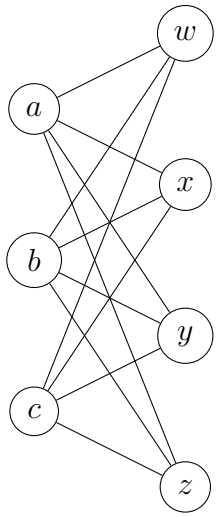


Figure 2.8: A complete bipartite graph $K_{3,4}$ where $\hat{\sigma}(a, b) = 1$ but $a \notin N(b)$

$N(u) \triangle N(v) = \varnothing$. Hence, all elements of $N(u)$ are in $N(v)$ and vice versa. Thus, $N(u) = N(v)$ if $\hat{\sigma}(u, v) = 1$.

Now assume the neighborhoods of u and v are the same. Since $N(u) = N(v)$, the symmetric difference between the neighborhoods is empty. So $\sigma(u, v) = 0$, hence $\hat{\sigma}(u, v) = 1$. Consequently, two vertices normalized neighborhood similarity is one if and only if their neighborhoods are the same. □

Note, vertices may have a large $\hat{\sigma}$ but not be adjacent; see Figure 2.8 as an example. In fact, the only way for $\hat{\sigma}$ to be 1 is if the vertices are not adjacent.

2.3 Condensation

Graphs contain redundant information when it comes to Total Domination Numbers. As an example, the pairs of $\{a, b, c\} \times \{w, x, y, z\}$ all form Total Dominating Sets of the graph in Figure 2.8. To find the Total Domination

Number of the graph, we only need to choose one element of each set. Consequently, finding the Total Domination Number of the example is the same as a P_2. So, by leveraging similarities between vertex neighborhoods, we can treat groups of vertices as a single vertex. The larger the group of similar vertices we can construct, the smaller the graph we can find with the same Total Domination Number. This is important since our goal is to improve the performance of Total Domination Number algorithms by reducing graph order through contraction.

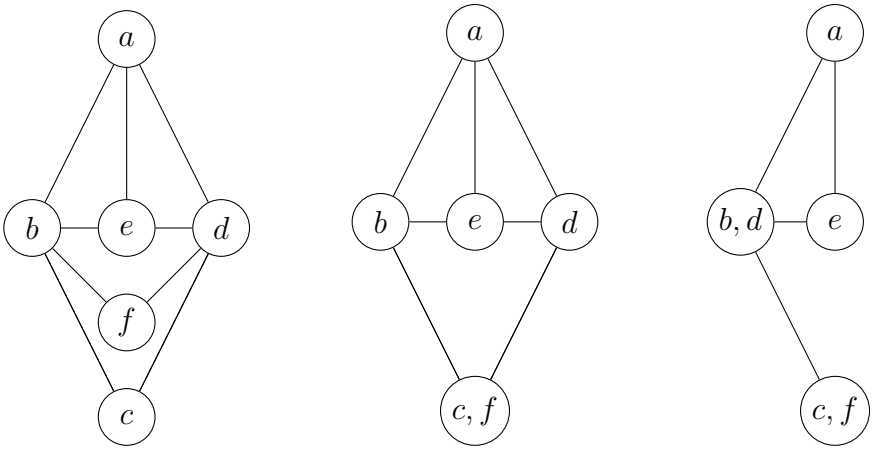
2.3.1 Mathematical Model

There are multiple graph operations which produce graphs with fewer vertices. One such operation is vertex contraction. Vertex contraction combines two vertices in a graph into a single, new vertex, whose neighborhood is the union of the two vertices which wer combined.

Definition 2.3.1 (Vertex Contraction)**.** A *vertex contraction* results in two vertices v_i and v_j "becoming" a single vertex $v_{i,j}$ such that $N(v_i) \cup N(v_j) = N(v_{i,j})$.

In order to combine multiple vertices using vertex contraction, we must compose a sequence of contractions. The sequence of contractions must take into account all early contractions and the intermediate graphs that they produced. Figure 2.9 shows contractions performed on a graph sequentially.

We want to define condensation to use a relation on the vertices instead of a specific order. This makes the operation more general; combining all vertices



(a) Initial Graph (b) c, f (c) b, d

Figure 2.9: A Graph with vertex contractions c, f and b, d

with a normalized neighborhood similarity of a specific value can work for an entire graph class. We do so by the use of quotient graphs.

Definition 2.3.2 (Quotient Graph)**.** The *quotient graph* H of a simple graph G is the graph obtained by partitioning $V(G)$ by an equivalence relation $\sim$ then constructing edges in H if there exists an edge between any member in the two parts:

$$H = (V(G)/\sim, \{xy \mid x \in V(G)/\sim, y \in V(G)/\sim, x \cap N(y) \neq \varnothing\}).$$

Fundamentally, quotient graphs combine groups of vertices which are equivalent by the given relation into a single vertex. If we select a relation which partitions vertices into parts where all vertices have identical neighborhoods, the quotient graph is the condensation of each unique neighborhood into a single vertex. So, the overall reduction is dependent on the choice of equivalence relation. We provide an example relation below.

Let us define a relation to contract vertices for a graph $G = (V, E)$:

$$\sigma_1 := \{(u, v) \in V^2 \mid \hat{\sigma}(u, v) = 1\}.$$

Proposition 2.3.1. *The relation σ_1 is an equivalence relation.*

Proof. By the properties in Remark 2.2.1, we know σ_1 is reflexive and symmetric. Additionally, we know if $\hat{\sigma}(u, v) = \hat{\sigma}(v, w) = 1$, that $N(u) = N(v)$ and $N(v) = N(w)$, so transitively $N(u) = N(w)$. Thus, $\hat{\sigma}(u, v) = \hat{\sigma}(v, w) = 1$ implies $\hat{\sigma}(u, w) = 1$. Therefore, σ_1 is an equivalence relation. □

In most non-trivial graphs, it is rare for a significant number of vertices to have identical neighborhoods. As such, it would be rare for a quotient graph under the relation σ_1 to result in a smaller graph.

We will use the following lemmas to show that the Total Domination Number is invariant for the quotient graph under σ_1.

First, we show that a total dominating set has a corresponding set which totally dominates the quotient graph under σ_1.

Lemma 2.3.1. *On a graph G and a quotient graph $G' = G/\sigma_1$, a set D is a total dominating set of G if and only if a corresponding set $D' = \{[v] \mid v \in D\}$ is a total dominating set of G'.*

Before we prove the statement above, please note equivalence class and partition are used interchangeably. Also consider that $[v]$ is the equivalence class of v.

Proof. Let G be a graph with a quotient graph $G' = G/\sigma_1$. Assume a set of vertices D is a total dominating set of G. Note that each vertex of G' is a subset of $V(G)$ such that all elements of the subset have the same neighborhood. In other words, the vertices of G' are parts of a partitioning of the vertices of G by an equivalence relation σ_1. By the definition of a quotient graph, two vertices of G' are adjacent if they contain elements which are adjacent in G. So if two vertices u, v from G are adjacent, their equivalence classes $[u]$ and $[v]$ are adjacent in G'. Hence if D is a total dominating set of G then $D' = \{[v] \mid v \in D\}$ is a total dominating set of G'. Now assume D' is a total dominating set of G'. So $N_{G'}(D') = V(G')$. Since the vertices of G' are full neighborhood clusters of $V(G)$, we can choose one element of each vertex from D' to be in a set D. Recall vertices in G' are adjacent if they contain elements which are adjacent in G. So if D' is adjacent to all vertices of G' then D must be adjacent to all vertices of G. Thus D is a total dominating set of G. Therefore a set D is a total dominating set of a graph if and only if there is an associated total dominating set D' of the quotient graph. □

Proposition 2.3.2. *For a graph G with a minimum total dominating set (MTD) D and some set $S \subseteq V(G)$ such that for all $u, v \in S$ we have $\hat{\sigma}(u, v) = 1$, then*

$$|D \cap S| \leq 1.$$

Proof. Assume for a graph G with a minimum total dominating set D that there is some set $S \subseteq V(G)$ where all pairs $u, v \in S$ have $\hat{\sigma}(u, v) = 1$. For

sake of contradiction, assume $|D \cap S| > 1$. So, there are at least two elements u, v of S in D. Note $\hat{\sigma}(u, v) = 1$ so $N(u) = N(v)$ by Remark 2.2.1. Consider that both vertices u and v are in D, so removing either one from D does not change the neighborhood of D. So $N(D \setminus \{u\}) = N(D \setminus \{v\}) = V(G)$. This contradicts D being a minimum total dominating set of G, since $D \setminus \{u\}$ and $D \setminus \{v\}$ are both smaller. Therefore, by contradiction, $|D \cap S| \leq 1$. □

In other words, at most one member of each equivalence class of V/σ_1 can be in the MTD set of a graph.

Theorem 2.3.2. *For a graph G and its quotient graph $G' = G/\sigma_1$, both G and G' will have the same Total Domination Number and $|V(G')| \leq |V(G)|$.*

Proof. Assume a graph G has a minimum total dominating set (MTD set) D and a quotient graph $G' = G/\sigma_1$. From Lemma 2.3.1, we know $\{[v] \mid v \in D\}$ is a total dominating set of G'. So any MTD set of G' is the same cardinality or less than $|D|$. If a MTD set of G' smaller than $|D|$ exists, by Lemma 2.3.1, there must be a corresponding total dominating set of vertices in G. But if a smaller total dominating set exists, D is not a MTD set of G. Hence all MTD sets of G and G' are the same size, i.e. $\gamma_t(G) = \gamma_t(G')$.

The order of G' is $|V(G)/\sigma_1|$, which can never be larger than $|V(G)|$ by the definition of the quotient set. Therefore, $\gamma_t(G) = \gamma_t(G')$ and $|V(G')| \leq |V(G)|$. □

We can apply Theorem 2.3.2 on Star graphs to get similar results to Lemma 2.1.1.

Proposition 2.3.3. *The quotient graph of a Star S_n under the relation σ_1 is the Path graph P_2.*

Proof. Given a Star graph S_n, note there are n leaves all adjacent to a central vertex. Let $G = S_n/\sigma_1$. Since all leaves have the same neighborhood, they belong to the same equivalence class in $V(S_n)/\sigma_1$. So, the graph G contains two vertices, one which corresponds to the leaves of S_n and the other which corresponds to the central vertex of S_n. Consider that that the leaves of S_n are adjacent to central vertex, hence the vertices of G are adjacent. Thus $G \cong P_2$. Therefore, the quotient graph of S_n is the Path graph P_2. □

By taking the quotient of S_n under σ_1, we reduce the original problem into the simpler task of finding the total domination number of P_2. Few graphs reduce so much under σ_1, so to produce smaller graphs, we must use a different relation.

We can generalize the relation to relax the conditions of partitioning. Given a set $S \subseteq [0, 1]$, let σ_S be defined as follows:

$$\sigma_S := \{(u, v) \in V^2 \mid \sigma(u, v) \in S\}.$$

Note, σ_S is not always an equivalence relation. For example, if 1 is not an element of S, our relation is not reflexive and is therefore not an equivalence relation. Below we provide criteria for various properties of σ_S.

- Reflexive

  By Remark 2.2.1, σ_S is reflexive if $1 \in S$.

- Symmetric

  It also follows from Remark 2.2.1 that σ_S is always symmetric.

- Transitive

  Transitivity holds if and only if for all vertex triples u, v, w, all of their shared normalized neighborhood similarities $\hat{\sigma}(u, v)$, $\hat{\sigma}(v, w)$, and $\hat{\sigma}(u, w)$ are contained within S.

Ordered Quotient

The quotient set and graph are only defined for equivalence relations. For our use case of vertex contraction, we want to perform a quotient like partitioning of vertices, without requiring the relation to be an equivalence relation. To achieve this, we define the ordered quotient and ordered quotient graph.

Definition 2.3.3 (Ordered Quotient)**.** The *ordered quotient* of a strict totally ordered set $(S, <)$ by some relation $\sim$ is a partitioning of S with the following properties given $a, b, c \in S$ where $a < b < c$: If $a \sim b$ then there is a part P of $S /_< \sim$ where $a, b \in P$. If $a \sim b$, $b \sim c$, but $a \not\sim c$ then there is a part P of $S /_< \sim$ where $a, b \in P$ but $c \notin P$.

Definition 2.3.4 (Ordered Quotient Graph)**.** The *ordered quotient graph* H of a graph G is the graph obtained by partitioning $V(G)$ by a relation $\sim$ with an ordering $<$ then constructing edges in H if there exists an edge between any member of the two parts in G. So,

$$H = (V(G) /_< \sim, \{xy \mid x \in V(G) /_< \sim, y \in V(G) /_< \sim, x \cap N(y) \neq \varnothing\}$$

In chapter 4, we study how different choices of the generalized relation σ_S change the overall reduction in graph order.

2.3.2 Computational Model

For the condensation to have applications, it must computationally be less expensive than finding the Total Domination Number of a graph. In Chapter 3, we explore the practicality of computing neighborhood similarities for use in condensation. Note for now that all the neighborhood similarities of a graph can be computed in polynomial time and accessed in constant time.

To compute the quotient graph or ordered quotient graph, we must first partition the vertices by a relation $\sim$. From Theorem 2.3.2, the only case when we know the Total Domination Number of the resulting graph and initial graph are the same is if $\sim = \sigma_1$. So ultimately, the quotient and ordered quotient graphs produce an approximate solution unless otherwise proven.

A naive algorithm is provided in Algorithm 1, which was inspired by the NetworkX implementation [6].

Performing comparisons against every element in all parts drastically slows Algorithm 1 down, requiring all vertices be compared with every other vertex. The use of a disjoint-set data structure improves the asymptotic performance by only performing comparisons against a single representative of each partition part.

Algorithm 2 uses the disjoint-set data structure to partition vertices. Underlying disjoint-set is a directed forest that supports three different operations: MakeSet, Union, and Find. The MakeSet operation initializes a

Algorithm 1 Compute Quotient Set

Require: Vertex Set V
Require: Edge Set E
Require: Condensation Set S
Require: Neighborhood similarity σ

```
Set partitions ← ∅
Stack stack
for all v ∈ V do
    added ← False
    for all p ∈ partitions do
        for all u ∈ p do
            if σ̂(u, v) ∈ S and not added then
                p_2 ← p
                partitions ← partitions \p
                partitions ← partitions ∪p_2 ∪ {v}
                added ← True
            end if
        end for
    end for
    if not added then
        partitions ← partitions ∪{v}
    end if
end for
return partitions
```

tree in the forest to the supplied vertex with itself as a parent. FIND returns the root of the forest the given vertex is contained in. It can be used to have a "representative" of each partition in order to avoid comparisons against all elements of the partition. The UNION operation joins together the two forests that the two given vertices lie in. There are multiple different implementations of UNION and FIND which have differing time complexities. Tarjan and Leeuwen in "Worst-case Analysis of Set Union Algorithms" showed the worst-case time complexity for a well designed implementation of UNION and FIND to be $\Theta(m\alpha(m,n))$ for n MAKESET operations and m FIND operations where α is the inverse Ackerman function [12]. The inverse Ackerman function is notable for its extremely slow growth, sub-logarithmic growth. Relevant to our algorithm, each UNION requires 2 FIND operations.

Algorithm 2 Compute Quotient Set using disjoint-set

Require: Vertex Set V
Require: Edge Set E
Require: Condensation Set S
Require: Normalized Neighborhood Similarities $\hat{\sigma}$

```
Disjoint-set M
for all v ∈ V do
    MakeSet(v)
end for
partitioned ← ∅
for all u ∈ V do
    if u ∉ partitioned then
        partitioned ← partitioned ∪{u}
        for all v ∈ V \ partitioned do
            if σ̂(u, v) ∈ S then
                M.Union(u,v)
                partitioned ← {v}
            end if
        end for
    end if
end for
```

Chapter 3

Neighborhood Similarity

3.1 Computation

In this section, we explore computational methods for computing the neighborhood similarities. The normalized neighborhood similarities are used to compute a quotient set via Algorithm 2 which ultimately will be used to reduce a graph. We consider two graph representations, Adjacency Lists and Adjacency Matrices.

A naive approach to compute all neighborhood similarities of a graph would be to iterate over all vertex pairs and compute two neighborhood similarities. However, due to the symmetric nature of neighborhood similarities, only a single similarity must be found per pair. Additionally, by Remark 2.2.1 we know the normalized neighborhood for any vertex with itself is reflexively 1. As such, on a graph of order n, exactly $\frac{(n-1)n}{2}$ similarities will need to be computed. In Algorithm 3, we implement a general algorithm that considers

the symmetric nature of the problem.

The structure used to represent the graph has a significant impact on the algorithm's average time complexity. It is important to note that regardless of the graph's original data structure, it is possible to store the results such that access is a constant time operation.

3.1.1 Adjacency Matrix

For an adjacency matrix of a graph, we can define a normalized neighborhood similarity matrix as follows:

Definition 3.1.1 (Normalized Neighborhood Similarity Matrix)**.** Given a graph G of order n with an adjacency matrix A, the *normalized neighborhood similarity matrix* B is given as follows:

$$B_{ij} = 1 - \frac{1}{D(i,j)} \sum_{k=1}^{n} (A_{ik} \oplus A_{jk}),$$

where $a \oplus b := a + b \bmod 2$ and $D(i,j) := \sum_{k=1}^{n} A_{ik} + A_{jk}$.

With some slight modifications, Algorithm 3 can be adapted to produce a normalized neighborhood similarity matrix. Note that to compare the neighborhood of two vertices in an order n matrix, n iterations must occur. Thus, computing a single normalzied neighborhood similarity is $\Theta(n)$. Since we are computing $\frac{(n-1)n}{2}$ similarities, the average time complexity is $\Theta(n^3)$.

It is important to note that the space required to store the neighborhood similarities can be halved. As a result of the symmetrical nature of neighbor-

Algorithm 3 Compute Neighborhood Similarities

Require: Graph $G = (V, E)$

```
procedure SIMILARITY(G, u, v)
    sum ← 0
    d_u ← 0
    d_v ← 0
    for all v' ∈ N(v) do
        d_v ← d_v + 1
        for all u' ∈ N(u) do
            d_u ← d_u + 1
            if u' = v' then
                sum ← sum + 1
            end if
        end for
    end for
    return 1 − sum · (d_u + d_v)^{−1}
end procedure

procedure SIMILARITIES
    for all v ∈ V do
        result[v, v] ← 1
    end for
    visited ← ∅
    for all v ∈ V do
        visited ← visited ∪ {v}
        for all u ∈ V \ S do
            similarity ← SIMILARITY(G, u, v)
            result[u, v] ← similarity
            result[v, u] ← similarity
        end for
    end for
    return result
end procedure
```

hood similarities, only the upper or lower triangle needs to be stored in the result matrix.

With or without the space saving technique, retrieving a similarity from the matrix is a constant time operation.

3.1.2 Adjacency List

As an alternative to the adjacency matrix, the neighborhood similarities can be computed from an adjacency list and stored in a hash table. To achieve constant time similarity retrievals using the hash table, we must use a hash function with no collisions. If we carefully choose our hash function so that symmetric vertex pairs collide, we can achieve a similar space reduction to that of upper triangle normalized neighborhood similarity matrices.

Xie presents a pairing function in [14] that meets our criteria. This function F is a bijective mapping such that $F : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. The function is defined as:

$$F(m,n) = \frac{1}{4}\left((m+n-1)^2 + (m+n-1) \bmod 2\right) + \min(m,n). \tag{3.1}$$

Adjacency lists differ from adjacency matrices with number of iterations to walk a neighborhood being bounded above by $\Delta(G)$ instead of $|V(G)|$. The less connected a graph is, the smaller $\Delta(G)$ is relative to $|V(G)|$. So, for graphs with fewer edges, it is preferable to use an adjacency list. Furthermore, since we only care about the size of the symmetric difference of the neighborhoods,

we only need to compare up to the size of the smaller neighborhood. We provide Algorithm 4, which takes the prior statements into consideration.

The computational complexity of computing the degrees given an adjacency list is dependent on the max degree. Let Δ be the max degree of the graph. In the best case, the graph is a Star, so all Δ of thhe leaf vertices have a time complexity of $O(1)$ while the non-leaf vertex has a time complexity of $\Theta(\Delta)$. Hence, the best case time complexity is $\Theta(\Delta)$. Now let us consider the worst case for computing degrees, a complete graph. For all n vertices, we must iterate over Δ elements. Thus, the worst case has a time complexity of $O(|V|\Delta)$. So the average complexity to get the degrees is $\Omega(\Delta), O(|V| \cdot \Delta)$.

The time complexity of computing a single normalized neighborhood similarity is also dependent on a maximum degree Δ. In the best case, the two vertices are leaves so only 1 comparison is made. It follows that the best case has a time complexity of $\Theta(1)$. The worst case is when both vertices are of the maximum degree Δ. We can sort the vertex neighborhoods on average in $O(\Delta \log \Delta)$ time. Sorting allows the two neighborhoods to be iterated over simultaneously in $O(\Delta)$ time. So the worst-case time complexity to compute a nomralized neighborhood similarity is $O(\Delta^2 \log \Delta)$. Therefore the average complexity of computing a single normalized neighborhood similarity is $O(\Delta^2 \log \Delta)$.

To compute the normalized neighborhood similarity map, we need to compute degrees of each vertex and the similarities of all vertex pairs. Rather than trying to sort the neighborhoods of each vertex as they are encountered, we can sort them all at once in advance. For each vertex, note we can sort

Algorithm 4 List Normalized Neighborhood Similarity

Require: Adjacency list L
Require: $n \geq 2$ vertices
  procedure SIMILARITY(L, u, v, d)
    $i \leftarrow d[u] - 1$
    $j \leftarrow d[v] - 1$
    sum $\leftarrow 0$
    while $i \geq 0$ and $j \geq 0$ **do**
      if $L(u)[i] > L(v)[j]$ **then**
        sum $\leftarrow$ sum $+ 1$
        $i \leftarrow i - 1$
      else if $L(u)[i] > L(v)[j]$ **then**
        sum $\leftarrow$ sum $+ 1$
        $j \leftarrow j - 1$
      else
        $i \leftarrow i - 1$
        $j \leftarrow j - 1$
      end if
    end while
    if $i < 0$ and $j >= 0$ **then**
      sum $\leftarrow$ sum $+ j + 1$
    else if $j < 0$ and $i >= 0$ **then**
      sum $\leftarrow$ sum $+ i + 1$
    end if
    return $1 - \text{sum} \cdot (d[u] + d[v])^{-1}$
  end procedure
  procedure SIMILARITYMAP(L, n)
    Create Hash Table M with hashing function Eqn. 3.1
    Compute degrees and store in array d
    for $i \leftarrow 1 \ldots n$ **do**
      Sort neighborhood of i
      $M[i, i] \leftarrow 1$
    end for
    for $u \leftarrow 2 \ldots n$ **do**
      for $v \leftarrow u + 1 \ldots n$ **do**
        $M[i, j] \leftarrow$ SIMILARITY(L, u, v, d)
      end for
    end for
    return M
  end procedure

their neighborhoods in $O(\Delta \log \Delta)$ time, so we sort all vertex neighborhoods in $O(|V|\Delta \log \Delta)$. Since each vertex neighborhood is sorted, computing a normalized neighborhood similarity of a vertex pair is only $O(\Delta)$. Since we must look at each pair, the time complexity to generate the normalized neighborhood similarity map is $O(|V|^2\Delta)$.

3.1.3 Comparison

Between an adjacency list and adjacency matrix, the adjacency list has a time complexity which is asymptotically better. So, unless otherwise specified, this thesis assumes graphs are represented using adjacency lists and neighborhood similarities are stored using a hash table.

As a brief aside, a real implementation of normalzied neighborhood similarity on adjacency matrices may perform close to on par with adjacency lists. This would be a result of adjacency matrices better utilizing hardware features such as CPU vectorization and bit packing.

3.2 Other Measures of Similarity

Our definition of neighborhood and normalized neighborhood similarity is not unique. In fact, we could define the normalized neighborhood similarity of two vertices u and v to be $\frac{|N(u)\cap N(v)|}{|N(u)\cup N(v)|}$. This potential definition holds all properties of $\hat{\sigma}$ from Remark 2.2.1. However, any alternative definition of normalized neighborhood similarity that considers all neighbors of both vertices performs no better than Algorithm 4. We prove the prior statement below.

Proposition 3.2.1. *For any neighborhood similarity definition that considers all adjacent vertices, the worst case time complexity to compute all neighborhood similarities of a graph is* $\Omega(|V|^2\Delta)$.

Proof. Let s be a function that computes a neighborhood similarity in constant time and takes into account all vertices adjacent to its operands. Note for a graph of order $|V|$, there are exactly $\frac{1}{2}(|V|^2 - |V|)$ neighborhood similarities. So in all cases, the time complexity of computing neighborhood similarities is $\Omega(|V|^2)$. Consider a case of computing $s(u, v)$ for vertices u and v where both vertices are of the maximum degree Δ. If the neighborhoods $N(u)$ and $N(v)$ can be compared in $O(1)$, then they must be in a data structure which allows so. The fastest an insertion of a distinct element into a data structure can be is $O(1)$, so performing Δ insertions must be $\Omega(\Delta)$. So, the time complexity to compute a neighborhood similarity must be $\Omega(\Delta)$ in the worst case. Therefore, the worst-case time complexity to compute all neighborhood similarities using s is $\Omega(|V|^2\Delta)$. □

Chapter 4

Quotient Graphs

Computing the Total Domination Number of a graph in general is NP-Complete, as shown by Henning and Yeo in [7]. If $P \neq NP$, the time complexity for a general algorithm to compute the Total Domination Number of a graph would grow in a non-polynomial manner. Assuming this to be true, the graph's order is highly significant to the real runtime of computing the Total Domination Number. We present a pre-processing algorithm which is guaranteed to yield graphs with orders no larger than the original.

The effectiveness of the algorithm will be measured as a ratio of the resulting order over the original order. This chapter's analysis focuses on optimizing the selection of normalized neighborhood similarities for condensation, aiming to minimize the ratio. Under ideal circumstances, a graph G will have a quotient graph $G' = G/\sigma_1$ where $|V(G')| = \gamma_t(G)$. In these cases, we are able to achieve a reduction ratio of $\frac{\gamma_t(G)}{|V(G)|}$.

4.1 General Algorithm

In this section, we will explore a general algorithm to condense a graph. While the general algorithm works for any graph, there are multiple graph classes where other methods yield asymptotically faster results. Regardless, Algorithm 5 is guaranteed not to increase the Total Domination Number of a graph while decreasing the order in polynomial time. Depending on the choice of normalized neighborhood similarities to condense on, Algorithm 5 produces a graph with either the exact or approximate Total Domination Number of the input. As an optimization, we use the disjoint-set data structure from the output of the quotient set Algorithm 2. This optimization allows us to only check one normalized neighborhood similarity when considering adding a vertex to a part.

We now provide the general Algorithm 5 for producing quotient graphs.

A critical part of condensing a graph via Algorithm 5 is creating a quotient set. In Section 3.1.2, we showed that all normalized neighborhood similarities can be computed in $O(|V|^2\Delta)$ time. The top level loop iterates over all vertices, adding a representative of each part of the partition and its edges to the output graph.

In the worst case, all vertices of the original graph lie in their own part and all vertices have the maximum degree. If each vertex is isolated in its own part, the following operations are required for each vertex: a FIND operation in the outer loop, a loop over its neighbors, and a FIND operation for each neighbor. Recall that α is the slow growing inverse Ackerman function, and that it describes the time complexity of the FIND operation. The

Algorithm 5 The General Condensation Algorithm

Require: Graph G
Require: Normed neighborhood similarities $\hat{\sigma}$
Require: Condensation Set S
  $M \leftarrow$QuotientSet$(G, S, \hat{\sigma})$ ▷ Disjoint-set
  $V' \leftarrow \varnothing$
  $E' \leftarrow \varnothing$
  for all $v \in V(G)$ **do**
    if M.Find$(v) = v$ **then** ▷ iterate over part roots
      $V' \leftarrow V' \cup \{v\}$
      for all $u \in N(v)$ **do**
        $r \leftarrow$ M.Find(u)
        $E' \leftarrow E' \cup \{vr\}$
      end for
    end if
  end for
  return (V', E')

first find operation occurs on each outer loop, so the amortized time complexity of it is $O(|V|\alpha(|V|, |V|))$. The second find operation occurs once per each Δ inner loop. So the amortized time complexity of the inner loop is $O(\Delta\alpha(|V|, |V|))$. Thus, in the worst case the outer loop has a time complexity of $O(|V|\alpha(|V|, |V|) \cdot (1 + \Delta))$. Note that the time complexity of the outer loop is less than that of computing the normalized neighborhood similarities. Therefore, the worst-case time complexity of Algorithm 5 is $O(|V|^2\Delta)$.

4.2 Wheels

Some graphs appear as if they should reduce but do not. As an example, a Wheel on 6 or more vertices does not reduce under the relation σ_1. From Lemma 2.1.1, we know $\gamma_t(W_n) = 2$, so under ideal circumstances we should

u	v	$\hat{\sigma}(u, v)$
peripheral	center	$1 - \frac{n-2}{n+3}$
peripheral	adjacent	$1 - \frac{4}{6}$
peripheral	non-adjacent	$1 - \frac{2}{6}$

Table 4.1: Neighborhood similarities of W_n for $n > 5$

be able to find a set $S \subseteq [0, 1]$ where $W_n/\sigma_S = P_2$.

Table 4.1 illustrates the three classes of neighborhood similarities for W_n with $n \geq 6$. Because of this, we can use the Algorithm 6 which is a modification of Algorithm 5 to condense the graph. The specialized Algorithm 6 produces the same output as the general Algorithm 5 if the later were to use a condensation set of $\{\frac{1}{3}, \frac{2}{3}\}$.

Algorithm 6 Wheel Condense

Require: Wheel Graph W_n
Require: Neighborhood similarities $\hat{\sigma}$
Require: $n \geq 6$
  center $\leftarrow \varnothing$
  periphery $\leftarrow \varnothing$
  for $i \leftarrow 1 \ldots n$ **do**
    for $j \leftarrow i + 1 \ldots n$ **do**
      if $\hat{\sigma}(i, j) = \frac{1}{3}$ or $\hat{\sigma}(i, j) = \frac{2}{3}$ **then**
        periphery $\leftarrow$ periphery $\cup \{i\}$
      else
        center $\leftarrow i$
      end if
    end for
  end for
  $V \leftarrow \{\text{center}, \text{periphery}\}$
  $E \leftarrow \{\{\text{center}, \text{periphery}\}\}$
  return (V, E)

Note that in all cases, Algorithm 6 must iterate exactly $\frac{n(n+1)}{2}$ times and perform a constant amount of work. Thus, the time complexity to condense a

Wheel graph is $\Theta(n^2)$.

4.2.1 Reduction Ratio

Both the general condensation Algorithm 5 and the specialized Wheel condensation Algorithm 6 result in an optimal condensation. For a Wheel of order n, given the relation σ_S where $S = \{\frac{1}{3}, \frac{2}{3}\}$, the quotient graph is of order 2 achieving a reduction ratio of $\frac{2}{n}$.

4.3 Partite Graphs

Partite graphs are a large family of graphs with applications in graph coloring.

Definition 4.3.1 (independent set)**.** A set of vertices which has no edges to other members of the set is an *independent set*.

Definition 4.3.2 (k-partite graph)**.** A graph comprised of k independent sets is a *k-partite graph*.

Definition 4.3.3 (Complete k-partite graph)**.** A k-partite graph is a complete *k-partite graph* if every vertex has an edge to all other vertices not in its independent set.

We notate complete k-partite graphs as $K_{x_1,x_2,\ldots,x_n}$, where x_i is the cardinality of the i-th independent set.

When $k = 2$, a k-partite graph is called a bipartite graph. It is important to note that in general, identifying a k-partite graph is NP-Complete for

$k > 2$, as shown in [4] by Garey and Johnson. We still care to study these graphs for additional insights into the application of our Algorithm 5.

4.3.1 Complete Bipartite and k-Partite

Let us start with a complete bipartite graph.

Consider the graph $K_{a,b}$ with independent sets A and B. Note each vertex in A is adjacent to all vertices of B, so the vertices of A have a degree of b with a neighborhood B. Similarly, the vertices of B have a degree of a and a neighborhood of A. So when condensing a complete bipartite graph, the normalized neighborhood similarities are either zero or one.

We will now show how this generalizes to complete k-partite graphs.

Proposition 4.3.1. *The quotient graph of a complete k-partite graph under σ_1 has exactly k vertices.*

Proof. Let G be a complete k-partite graph. By the definition of a complete k-partite graph, all members of an independent set of vertices X_i for $i \leq k$ are adjacent to every vertex not in X_i. Without loss of generality, all members of X_i have the same neighborhood, so they all have a normalized neighborhood similarity of 1. Thus each independent set of the original graph forms an equivalence class under σ_1. Therefore there are exactly k vertices in the quotient graph of a complete k-partite graph under the relation σ_1. □

Observe that the normalized neighborhood similarity for any adjacent ver-

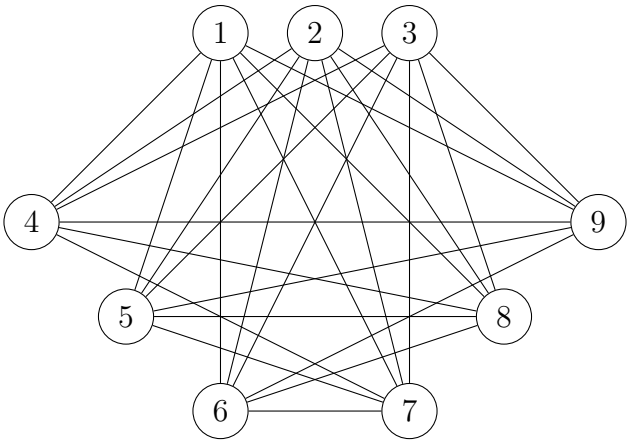


Figure 4.1: $K_{3,3,3,}$

tices u, v in the i-th and j-th sets of size x_i, x_j respectively will be

$$1 - \frac{x_i + x_j}{2\left(\sum_{l=1}^{k}\right) x_l - x_i - x_j}.$$

As an example, let us consider the $K_{3,3,3}$ in Figure 4.1. From our definitions, we know the normalized neighborhood similarity within any independent set is 1. The normalized neighborhood similarity $\hat{\sigma}(1, 4)$ is $\frac{1}{2}$.

4.3.2 Reduction Ratio

Since the vertices of the quotient graph of a complete k-partite graph are its independent sets, we achieve a reduction ratio of $\frac{k}{n}$ when using σ_1 as our relation.

4.4 Trees

Trees are another large family of graphs with interesting properties and many applications.

There are two equivalent definitions of a Tree graph. We leave it as an exercise to the reader to prove their equivalence.

Definition 4.4.1 (Tree)**.** A *Tree* is a simply connected acyclic graph.

An equivalent definition of a tree is the following.

Definition 4.4.2 (Tree)**.** A simple graph is a *Tree* For any two vertices in a simply connected graph, there exists exactly one Path between them.

Both definitions are relevant.

Some examples of Trees are Path graphs, Star graphs, and Caterpillar graphs, as seen in Figure 4.2.

We now show that for any Tree, only the leaf vertices can be reduced by condensation.

Theorem 4.4.1. *The only vertices which can be condensed in a Tree graph are leaf vertices.*

Proof. Given a Tree $T = (V, E)$, let P be the set of leaf vertices; i.e. $P = \{v \in V \mid \deg(v) = 1\}$. For the sake of contradiction, assume there exists $u, v \in V \setminus P$ such that $\hat{\sigma}(u, v) = 1$. So $N(u) = N(v)$ and $\deg(u) \neq 1 \neq \deg(v)$. It follows that for all $w \in N(u)$, there is a uwv path. Hence, without loss of generality, there are at least $\deg(u)$ paths from u to v in T. Recall T is a Tree, so there is exactly one path from u to v. But then $\deg(u) = \deg(v) = 1$ which

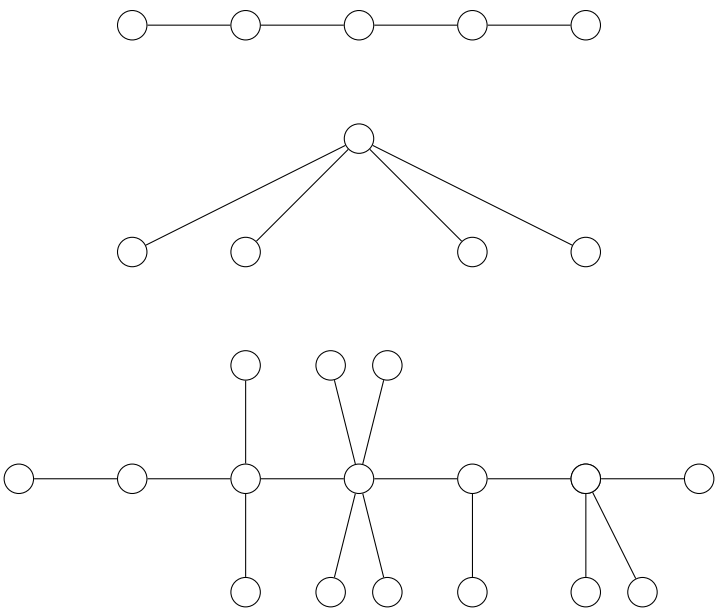
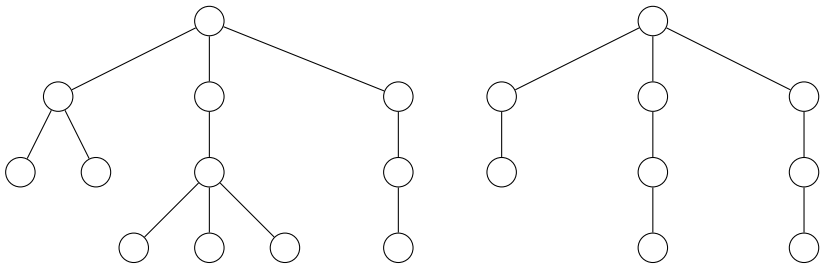


Figure 4.2: Examples of Tree Graphs

contradicts u and v not being leaf vertices. Therefore, only leaf vertices in a Tree are condensable. □

As a result of Theorem 4.4.1, we can expect the reduction ratio obtained from computing a quotient graph to be dependent on the number of leaves. In Figure 4.3, we show an example of a Tree whose quotient graph has an order only 3 smaller than the original. The class of Trees that condense the most are Star graphs, since all vertices except for one are leaves. Conversely, Path graphs condense the least, because there are only two leaves which condense for P_3 and no other case.

Henning and Yeo in [7] provide a linear time algorithm to compute the Total Domination Number of a Tree. Since our general Algorithm 5 runs in quadratic time on average, preprocessing a tree does not yield any benefits when computing its Total Domination Number.



(a) T (b) T/σ_1

Figure 4.3: A graph T and its quotient graph T/σ_1

Chapter 5

Analysis

5.1 Competitive Analysis

Currently it is unknown if NP-Hard problems like Total Domination Number can be solved in polynomial time. In the likely case that $P \neq NP$, the time for an algorithm to generate an optimal solution scales exponentially with the graph size. As such, it is not practical on larger graphs to search for the optimal solutions. So, in order to achieve tangible results, general Total Domination Number algorithms should approximate the actual total domination number in polynomial time. An approximation is meaningless without bounds; one could always approximate the Total Domination of a graph to be its order in linear time. When evaluating the quality of an approximation algorithm, we care about the difference between the optimal solution and the approximate solution.

The quality of an approximation algorithm is typically measured using

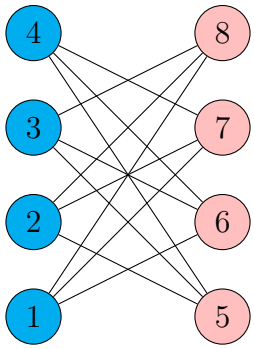


Figure 5.1: An optimal greedy coloring of a crown graph [8]

competitive analysis. To understand this, consider the classic problem of graph coloring. Greedy Coloring is a type of graph coloring in which vertices are assigned a "color" that none of its neighbors have. The coloring is considered greedy if a vertex is assigned the first color available. Figure 5.1 shows an optimal greedy coloring on a graph. If the algorithm which produced Figure 5.1 uses a different vertex ordering, the result is Figure 5.2. Ideally we use two colors, but in the worst case we use eight for a ratio of $\frac{8}{2} = 4$. The ratio can be generalized for larger crown graphs to be $\frac{n}{2}$ for crowns with n vertices [8]. It is important to note that the ratio of $\frac{n}{2}$ only holds for graphs of crown graphs.

Competitive analysis provides insight into worst-case performance across all instances in terms of lower and upper bounds. Proving a lower bound is generally straightforward: if we can identify a single instance with a certain approximation ratio, it establishes a minimum ratio for the algorithm's performance. On the other hand, proving an upper bound is significantly more difficult because it requires that certain ratio holds for all cases. Proving an upper bound is challenging for all but the simplest of problems.

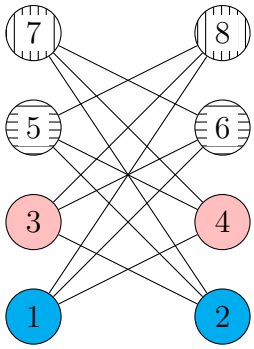


Figure 5.2: An suboptimal greedy coloring of a crown graph [8]

For our Algorithm 5, we care about the Total Domination Number of the output compared to the input. To be specific, our algorithm receives a graph G and a set of normalized neighborhood similarities $S \subseteq [0, 1]$ and outputs a smaller graph G'. Performing competitive analysis on our algorithm means finding a set S which for a specific ordering causes $\frac{\gamma_t(G')}{\gamma_t(G)}$ to be as large as possible. The worst case for any graph G is the set of normalized neighborhod similarities S which which minimizes $\frac{\gamma_t(G')}{\gamma_t(G)}$, while the best case is the set which maximizes the ratio. Similar to the greedy coloring example, we perform analysis on only one family of graphs at a time.

First, let us note that it is fruitless to choose any set S which does not contain a normalized neighborhood similarity from G. This follows from the definition of condensation, which requires the normalized neighborhood similarity of two vertices to be in S for condensation to occur.

Another degenerate case we should exclude from our analysis is if the condensation set S contains all normalized neighborhood similarities found in our graph. It follows from the definitions that the quotient graph produced in this case would be the Trivial graph, for which Total Domination is undefined.

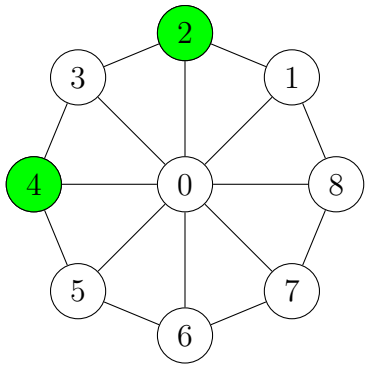


Figure 5.3: W_8 where $\hat{\sigma}(2,4) = \frac{1}{3}$

In all of the following graph classes, we will ignore the cases described above.

5.1.1 Wheels

The following holds for all wheels on 6 or more vertices except for wheels on 13 vertices.

Recall from Table 4.1 that the possible normalized neighborhood similarities of a Wheel are $\frac{1}{3}$, $\frac{2}{3}$, and $1 - \frac{n-2}{n+3}$. A similarity of $\frac{1}{3}$ is attained for any two non-central vertices which share a non-central vertex as a neighbor, as seen in Figure 5.3. All non-central vertex pairs which do not share a non-central vertex as a neighbor have a similarity of $\frac{2}{3}$, similar to Figure 5.5. Lastly, for a Wheel on $n+1$ vertices the similarity between the central vertex and any other is $1 - \frac{n-1}{n+3}$, as shown in Figure 5.4.

Note a Wheel graph W_n is undefined for $n < 3$, and in the special case of $n = 12$, a wheel only has two normalized neighborhood similarities since $1 - \frac{n-2}{n+3} = \frac{2}{3}$.

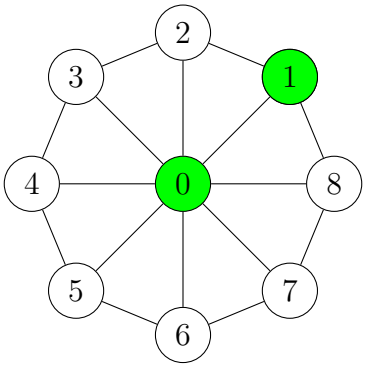


Figure 5.4: W_8 where $\hat{\sigma}(0,1) = 1 - \frac{7}{11}$

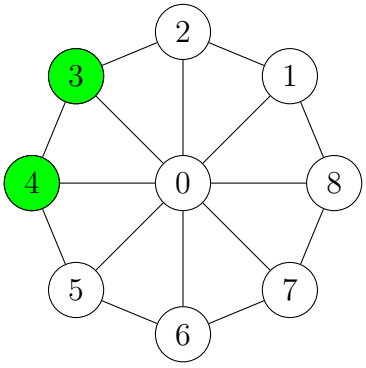


Figure 5.5: W_8 where $\hat{\sigma}(3,4) = \frac{2}{3}$

Proposition 5.1.1. *The worst case for Wheel condensation is using the value* $\frac{n-1}{n+3}$ *for a wheel on* $n+1$ *vertices.*

Proof. Given a Wheel graph G on $n+1$ vertices where $n \in \{5, 6 \ldots, 11, 13, \ldots\}$, let $s = 1 - \frac{n-1}{n+3}$. We know all vertex pairs containing the central vertex have a normalized neighborhood similarity of $1 - \frac{n-1}{n+3}$. So, regardless of order, $G /_< \sigma_s$ produces the Trivial graph, which has an undefined Total Domination Number. Therefore the worst set to condense on is $\{1 - \frac{n-1}{n+3}\}$. □

Proposition 5.1.2. *The best case for Wheel condensation is using the set* $\{\frac{1}{3}, \frac{2}{3}\}$

Proof. Given a Wheel graph G on $n+1$ vertices where $n \in \{5, 6, \ldots, 11, 13, \ldots\}$, let $S = \{\frac{1}{3}, \frac{2}{3}\}$. We know that any vertex pairs where neither vertex is the central vertex have a normalized neighborhood similarity within S. So, regardless of order, the result of the condensation $G /_< \sigma_S$ is a graph with two connected nodes, one containing the central vertex from G and the other containing every other vertex. Hence $G/\sigma_S \cong P_2$. Note $\gamma_t(G) = \gamma_t(P_2)$ so no other choice of S could perform any better. Also note $|V(P_2)| = 2$, so the only potential smaller graphs are the trivial or empty graphs. Total Domination is undefined for both of these graphs. Thus there is no other choice of S which yields a closer Total Domination Number and smaller graph. Therefore the best set to condense Wheels with is $\{\frac{1}{3}, \frac{2}{3}\}$. □

5.1.2 Cycles

The Total Domination Number of Paths and Cycles are found in Theorem 2.1.3. The following holds for all Cycles on 5 or more vertices.

A pair of vertices in a Cycle has two possible normalized neighborhood similarities. Their similarity will be 0.5 if they are a distance of two from each other and 0 if they are not.

Proposition 5.1.3. *The worst case for Cycle condensation is achieved by using the set* $\{0\}$.

Proof. Given a Cycle C_n where $n \geq 5$, let $S = \{0\}$. Assume there is a vertex v which is the first vertex that normalized neighborhood similarity comparisons will be made with. Regardless of the vertex iteration order, $G /_< \sigma_S$ produces a graph where one vertex represents the vertices a distance of two away from v and the other represents everything else. So $C_n/\sigma_S \cong P_2$. Hence by Theorem 2.1.3, $|\gamma_t(C_n) - \gamma_t(C_n /_< \sigma_S)| \in \{\frac{n}{2} - 2, \frac{n+1}{2} - 2, \frac{n+1}{2} - 1\}$. Note that Total Domination is undefined for any smaller graphs. Thus there is no other choice of S which will yield a Total Domination Number farther from that of C_n. Therefore the worst set to condense a Cycle on is $\{0\}$. □

Since Cycles only have two possible normalized neighborhood similarities, we know 0.5 must be the best value to condense on. However, we will still explore what it means to condense using the value 0.5 for completeness.

Proposition 5.1.4. *The best case for Cycle condensation is achieved by using the set* $\{0.5\}$ *and iterating over the vertices such that the next vertex is adjacent to the current.*

Proof. Given a Cycle C_n where $n \geq 5$, let $S = \{0.5\}$. Assume the vertices are labeled using $\{1, 2, \ldots, n\}$ where adjacent vertices have a difference of 1 except for the vertices labeled n and 1. Note the only time a pair of vertices has a normalized neighborhood similarity of 0.5 is when they are a distance of two apart. So $C_n /_< \sigma_S$ produces a smaller Cycle graph. Consider that the nodes of $C_n /_< \sigma_S$ which contain 1 and 2 are $\{1, n-1, 3\}$ and $\{2, n, 4\}$ respectively. Because of the order of iteration, all vertices v greater than 4 are in a vertex with $\{v, v+2\}$. In the cases where $n-6 \not\equiv 0 \pmod 4$, some vertices will map to a vertex containing a single vertex in $C_n /_< \sigma_S$. If $n-6 \equiv 1 \pmod 4$ then $|V(C_n /_< \sigma_S)| = 3 + \frac{n-7}{2}$. Similarity if $n-6 \equiv 2, 3 \pmod 4$ then $|V(C_n /_< \sigma_S)|$ is equal to $4 + \frac{n-8}{2}$ and $3 + \frac{n-7}{2}$. Hence we have the following isomorphisms for $C_n /_< \sigma_S$:

$$C_n /_< \sigma_S \cong \begin{cases} C_{4+\frac{n-8}{2}} & n \equiv 0 \pmod 4 \\ C_{3+\frac{n-7}{2}} & n \equiv 1, 3 \pmod 4 \\ C_{2+\frac{n-6}{2}} & n \equiv 2 \pmod 4 \end{cases}.$$

Thus by Theorem 2.1.3 the condensed graph's Total Domination Number differs from the original by

$$|\gamma_t(C_n) - \gamma_t(C_n /_< \sigma_S)| = \begin{cases} \frac{n}{2} - \frac{n-8}{4} - 2 & n \equiv 0 \pmod 4 \\ \frac{n+1}{2} - \frac{n-7}{4} - 2 & n \equiv 1, 3 \pmod 4 \\ \frac{n}{2} - \frac{n-6}{4} - 1 & n \equiv 2 \pmod 4 \end{cases}.$$

Therefore the best set to condense a Cycle on is $\{0.5\}$. □

5.1.3 Paths

The analysis for Paths is similar to that of Cycles, since both have a Total Domination Number described by Theorem 2.1.3.

Proposition 5.1.5. *The worst case for Path condensation is achieved by using the set* $\{0\}$.

Proof. Given a Path P_n where $n \geq 5$, let $S = \{0\}$. Assume under some vertex ordering $<$, the first vertex is v. In the case that v is a leaf of P_n, every vertex except a vertex u shares no neighbors with v, and consequently has a normalized neighborhood similarity of zero. The vertex u is the vertex a distance of two away from v and has exactly one neighbor in common with v. So, the quotient graph obtained from the relation σ_S is the graph $(\{V(P_n) \setminus \{u\}, u\}, \{(V(P_n) \setminus u, u)\}))$ which is a P_2. Similarly in the case that v is not a leaf of P_n, there are only two vertices u, w which share neighbors with v. Note that the vertices u, w area distance of two from v each, so their neighborhoods have no vertices in common. Thus in this case, the result of the quotient graph of P_2 is $(\{V(P_n) \setminus \{u, w\}, \{u, w\}\}, \{(V(P_n) \setminus \{u, w\}, \{u, w\})\})$. Hence for any ordering of vertices $<$, $P_n /_< \sigma_S \cong P_2$. Note that the Total Domination Number is undefined for any smaller graphs. Therefore the worst set to condense a Path on is $\{0\}$. □

Proposition 5.1.6. *The best case for Path condensation is achieved by using the set* $\{\frac{2}{3}\}$.

Proof. Given a Path P_n where $n \geq 5$, let $S = \{\frac{2}{3}\}$. Let the leaf vertices be labeled v_1, v_n. Note any vertex pair whose degree sum is three or a multiple of

three must include a leaf vertex. The only vertices which can share neighbors with a leaf vertex are a distance of two away from them. So let v_3 and v_{n-2} be the vertices which share a neighbor with v_1 and v_n, respectively. Hence $|V(P_n/\sigma_S)| = n-2$. Consider that the vertices which v_1, v_3 and v_{n-2}, v_n share in common are only adjacent to the vertices of their respective pair. So those vertices become leaves in P_n/σ_S, while v_1, v_3 and v_{n-2}, v_n become adjacent to two vertices. Thus, the quotient graph of P_n by the relation σ_S is isomorphic to P_{n-2}. Note $\gamma_t(P_n) - \gamma_t(P_{n-2}) \in \{0, 1\}$ by Theorem 2.1.3. Therefore the best set to condense on a Path is $\{\frac{2}{3}\}$. □

Chapter 6

Conclusion and Future Work

Conclusion

In this work, we investigated the complexity of Total Domination in graphs and developed an algorithm to make the problem more tractable through preprocessing.

We began by observing that, in a minimum total dominating set, at most one vertex from any set of vertices with the same neighborhood is chosen to be in the set. From that, we created a measure of neighborhood similarity $\hat{\sigma}$. Using our measure of neighborhood similarity, we defined the ordered quotient and ordered quotient graph. The ordered quotient was used to group vertices by their neighborhood similarities and construct an ordered quotient graph with similar features to the original. In particular, ordered quotient graphs preserve exactly or approximately the Total Domination Number, while using a number of vertices that is no greater than the original.

Ordered quotient graphs and quotient graphs are tools to use when finding a total dominating set for a less researched graph. As we showed with complete k-partite graphs, large graph classes can be reduced down to a simpler, already solved forms via quotient graphs.

Our algorithm to compute ordered quotient graphs was shown to have a worst-case time complexity of $O(|V|^2\Delta)$. We analyzed effectiveness of our algorithm using competitive analysis on specific graph classes such as Wheels, Paths, and Cycles. When applied before a Total Domination algorithm, our preprocessing algorithm has the potential to substantially improve practical performance.

Future Work

We are interested in exploring the broader applications of our condensation algorithm. Future research could investigate applying the algorithm to subgraphs rather than entire graphs; this approach may yield a graph of lower order while simplifying the selection of neighborhood similarities for condensation. Additionally, extending our condensation algorithm to other domination variants could reveal further benefits, with identifying appropriate similarity measures for each variant being a promising direction.

Experimentally, we still have significant work to do. Running trials with real-world data will help us evaluate our algorithm's performance. We also plan to test our algorithm as a preprocessing step for various existing Total Domination algorithms. By comparing speed improvements, we aim to identify

which algorithms benefit the most from our preprocessing.

Bibliography

[1] Sharareh Alipour and Mohammadhadi Salari. *On distributed algorithms for minimum dominating set problem and beyond*. 2021. arXiv: 2103.08061 [cs.DC].

[2] Eduardo Álvarez-Miranda and Markus Sinnl. *Exact and heuristic algorithms for the weighted total domination problem*. 2019. arXiv: 1910.03363 [math.OC].

[3] Thomas H Cormen et al. *Introduction to Algorithms*. 3rd. Cambridge, MA: MIT Press, 2009.

[4] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1979. ISBN: 0716710447.

[5] Darij Grinberg. *An introduction to graph theory*. 2024. arXiv: 2308.04512 [math.HO].

[6] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. "Exploring network structure, dynamics, and function using NetworkX". In: *Proceedings of the 7th Python in Science Conference (SciPy2008)*. Ed. by Gäel

Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, Aug. 2008, pp. 11–15.

[7] Michael A. Henning and Anders Yeo. *Total domination in graphs*. Springer Monographs in Mathematics. Springer, New York, 2013, pp. xiv+178. ISBN: 978-1-4614-6525-6. DOI: 10.1007/978-1-4614-6525-6.

[8] Thore Husfeldt. *Graph colouring algorithms*. 2015. arXiv: 1505.05825 [cs.DS].

[9] Adel P Kazemi, Farshad Kazemnejad, and Somayeh Moradi. "Total mixed domination in graphs". In: *AKCE International Journal of Graphs and Combinatorics* 19.3 (2022), pp. 229–237.

[10] Garrison Koch and Nathan Shank. *On the k-attack Roman Dominating Number of a Graph*. 2024. arXiv: 2305.16256 [math.CO].

[11] Oscar Levin. *Discrete Mathematics: An Open Introduction*. 4th ed. CRC Press, 2024. URL: https://discrete.openmathbooks.org/dmoi4/.

[12] Robert E. Tarjan and Jan van Leeuwen. "Worst-case Analysis of Set Union Algorithms". In: *J. ACM* 31.2 (Mar. 1984), pp. 245–281. ISSN: 0004-5411. DOI: 10.1145/62.2160. URL: https://doi.org/10.1145/62.2160.

[13] Johan M.M. van Rooij and Hans L. Bodlaender. "Exact algorithms for dominating set". In: *Discrete Applied Mathematics* 159.17 (2011), pp. 2147–2164. ISSN: 0166-218X. DOI: 10.1016/j.dam.2011.07.001.

[14] Jianrui Xie. *On Symmetric Invertible Binary Pairing Functions*. 2021. arXiv: 2105.10752 [math.CO].

Appendix A

Notation

$G = (V, E)$:= graph

$V(G)$:= vertex set of graph

$E(G)$:= edge set of graph

$\deg(v)$:= degree of a vertex v

$\delta(G)$:= minimum degree of G

$\Delta(G)$:= maximum degree of G

$G/\sim$:= quotient graph of G under the relation $\sim$

$N(v)$:= open neighborhood of a vertex v

$N_G(v)$:= open neighborhood of a vertex v in the graph G

$N(S) = \bigcup_{v \in S} N(v)$:= open neighborhood of all vertices in the set S

$\gamma_t(G)$:= Total Domination Number of G

$\mathbb{N} := \{1, 2, \dots\}$

$G \cong H$:= graph isomorphism

$\overline{K_n}$:= a graph of n not connected vertices