We collect the proofs and lemmas which enable linear time algorithms for first-order sentences on graph classes with bounded expansion. Bounded expansion is a property limiting the edge to vertex ratio of a graph and its minors by a function on the permitted amount of edge contraction. The linear runtime is achieved by evaluating quantifier-free first-order sentences on each node after a linear time preprocessing phase. We also demonstrate original linear time solutions developed for interesting non-trivial problems over certain bounded expansion graph classes. We conclude after discussing the implications and potential future applications of this line of research.
Table of Contents
1 Introduction
2 Graph Classes with Bounded Expansion
   2.1 Necessary Definitions
   2.2 Bounded Expansion
3 Implicit Graph Representations
   3.1 Linear Time Calculation of Implicit Representations
4 Transitive Fraternal Augmentation
   4.1 Concept Overview
   4.2 Path Length Reduction
   4.3 Linear Time Calculation
   4.4 Repeatability
5 First-Order Logic and Quantifier Elimination
   5.1 First-Order Logic Properties and the Introduction of Structures
   5.2 Quantifier Elimination
6 Solutions in O(n)
   6.1 Triangle Free on Planar Graphs
   6.2 K-Cycle on Planar Graphs
7 Applications
   7.1 Gaifman Graphs
8 Conclusion
1 Introduction

For all intents and purposes applied research in the algorithms area of computer science is dedicated to solving useful problems at faster rates. Time complexity is one of the most significant areas of study and there is a constant effort to move important problems to faster complexity classes, reducing the time that they take relative to the size of their input. Graph theory is the branch of computer science dealing with sets of interrelated entities visualized using edges and vertices and has essentially infinite applications. Choosing to specialize and focus only on graphs with certain properties, usually considered in collections called classes, can lead to the development of significantly faster algorithms or, at the very least, uncover a fundamental utility that holds the promise of future rewards.

Sparse graph classes refer to collections of graphs whose edge to vertex ratio is asymptotically limited in some way. Designing algorithms exclusively for sparse graph classes lends itself well to improved time complexities because a specific bound is placed on the number of connections between two items. Therefore analyzing all sets of adjacent objects no longer automatically causes the time complexity to be polynomial. Bounded expansion appears to be the most comprehensively studied and ultimately useful delimiter of sparsity that encompasses a significant number of different classes. An in-depth discussion of the properties associated with bounded expansion can be found in section 2 of this paper. Previous work on graphs classes with bounded expansion revealed their potential for developing new, faster algorithms, although it can be difficult to collect this information together and there are integral concepts that lack clear, full explanations in the literature.

Research has mainly been focused on the evaluation of first-order sentences. There exist regimented systems that algorithms use to extract information from graphs. One such system, first-order logic, is a collection of simple symbols that can be arranged to ask almost any yes-or-no question in sentences of increasing complexity. The presence of quantifiers, symbols ∀x and ∃x that signify “for all x” and “there exists an x” respectively, increases the exponent of the polynomial time complexity for running the first-order sentences, going from n to n^2 to n^3 and beyond as each additional quantifier is added. Therefore moving an algorithm to a faster time complexity becomes an issue of eliminating these quantifiers, which bounded expansion can help facilitate.

The concept of bounded expansion was first introduced to the computer science community by Jaroslav Nešetřil and Patrice Ossona de Mendez in 2008 [5]. They went on to extensively explore the first-order properties and algorithmic aspects of graph classes with bounded expansion [3]. They appear to be behind the introduction of transitive fraternal augmentations as well. This work on classes of graphs with bounded expansion had been built upon and was later expanded by many others. Robertson and Seymour extensively explored the idea of minors, while Leiserson and Toledo developed the concept of r-shallow minors that are used to define bounded expansion [9]. Chrobak and Eppstein were the first to note and utilize the fact that if a graph has had its edges oriented such that no node has more than a certain number of edges pointed inward then the adjacency of two nodes can be checked in constant time, an aspect related to the concept of implicit representation discussed in section 3 [9][2]. Eppstein then discovered how to create a linear-time algorithm for finding a fixed subgraph on planar graphs as well as another graph class similar to but distinct from classes with bounded expansion,
minor-closed classes of graphs with locally bounded tree-width. Finding a fixed subgraph is considered the easiest first-order formula lacking any universal quantifiers. Nešetřil and Ossona de Mendez proved one could create a linear-time algorithm for all such first-order formulas without universal quantifiers. Seese proved that all first-order formulas, regardless of their quantifiers, could be run in linear time for any class with bounded maximum degree, and then Dvořák, Král’, and Thomas extended this result to graphs with bounded expansion [3][4]. Quantifier elimination for graph classes with bounded expansion was similar to previous work with graph classes of bounded degree, but ultimately required a different and more complicated series of techniques [5].

Section 2 of this paper explains the nature of bounded expansion and other related terms, and provides proofs for some properties that lead to the ultimate result of linear time first-order queries. Sections 3, 4, and 5 do the same for implicit representations, transitive fraternal augmentations, and quantifier elimination respectively. Section 6 demonstrates how to solve the non-trivial interesting problem known as k-cycle determination on the graph class with bounded expansion known as planar graphs and highlights elements of this method that suggest how other problems and graph classes with bounded expansion could also be handled. Section 7 explores the general applicability of graphs through the Gaifman database conversion process and describes the greatest obstacle to the utilization of this finding, namely large coefficients. The paper then concludes in section 8.

2 Graph Classes with Bounded Expansion
2.1 Necessary Definitions

A series of fundamental concepts must be established and defined before discussing the complexities of bounded expansion.

Computer science uses graphs as models to conceptualize many different types of systems.

**Definition 2.1**: A **graph** is a set of **nodes** or **vertices** that are connected by **edges**. **Graphs** can represent any related objects. The **nodes** may also be **colored** and the **edges oriented** to provide additional information or indicate one-way relationships respectively. A **directed graph** is a graph where some of the **edges** are **oriented**. **Edges** in an **undirected graph** are intuitively understood to be **oriented** in both directions. When referring to a **graph** G, the **size** of the **graph**, or |G|, is the number of **vertices**. The **order** of the **graph**, or ||G||, is the number of **edges**.

We are examining sparse graphs, which have a small number of edges compared to their number of vertices. Sparsity is a relative concept best utilized when discussing classes of graphs.

**Definition 2.2**: A **graph class** is a possibly infinite collection of graphs that all share some property or properties.

A class is generally considered sparse when one of the traits shared by its members is a limit on the edge to vertex ratio [9]. This limit is such that the number of edges has a
linear relationship with the number of vertices, or in other words one may always be able
to construe the number of edges as at most a multiplication of the number of vertices by
some constant. The number of edge-based operations performed when evaluating queries
over sufficiently sparse graph classes is strictly limited by said constant as a result, which
means algorithms can be designed for them with faster time complexities than normal.

Planar graphs are a quintessential example of a sparse graph class. We will use
them throughout this section to help demonstrate important concepts and ultimately
depict the central idea of the entire thesis.

**Definition 2.3:** A graph is **planar** if there exists a way to draw it on a single plane
with no edge crossing another.

Planar graphs are sparse because their number of edges is at most three times the number
of vertices minus six, which constitutes a sufficient basis for a linear relationship between
edges and vertices. We will now show this to be true.

**Theorem 2.1:** Every planar graph satisfies the relationship between the number of
vertices and edges $3|G| - 6 \geq ||G||$.

**Proof:** Euler’s formula states that the number of vertices plus the number of faces
minus the number of edges equals two for all planar graphs, or $|G| + f - ||G|| = 2$. Edges are therefore bound by both vertices and faces. We can minimize the
number of faces to be two-thirds the number of edges because faces require at
least three edges, as is the case when dealing with triangles, and edges can be part
of at most two faces. This statement can be expressed as $3f \geq 2||G||$ which will in
turn be reduced to $f \geq \frac{2}{3}||G||$ when both sides are divided by three. Plugging this
calculation into Euler’s formula returns $|G| + \frac{2}{3}||G|| - ||G|| \geq 2$, which simplifies to
$|G| - \frac{1}{3}||G|| \geq 2$, and when one-third the number of edges is added to both sides and
two is subtracted from both sides transforms into $|G| - 2 \geq \frac{2}{3}||G||$. Multiplying both
sides by three gets us our desired result $3|G| - 6 \geq ||G||$, the maximum number of
edges in terms of vertices.

Planarity is actually a minor-closed property, which is not always the case for properties
delineating the sparsity of a graph class.

**Definition 2.4:** Graph H is a **minor** of graph G if H can be obtained from G
through a combination of vertex deletions, edge deletions, and edge
contractions. An edge contraction is when two adjacent vertices are replaced
with one vertex with all of their edges. A class of graphs is **minor-closed** if any
minor of a graph in that class would still belong to that class. A subgraph is a
possibly smaller graph containing no edges or nodes not in the original, and is
equivalent to a minor of a graph if no edge contractions are needed to obtain the
minor from the original graph.

We will now confirm that planar graphs are minor-closed.

**Theorem 2.2:** The minor of any planar graph is also a planar graph.

**Proof:** By definition if a graph is planar there is a way to draw it so that its edges
do not intersect each other. Removing an edge or a vertex does not change the
position of the remaining edges, so it cannot force an intersection. Every instance
of edge contraction removes the edge that previously connected two nodes and
replaces the nodes at either end with a single node connected to all their edges. There cannot have been any other edges intersecting the prior location of the edge because the graph was planar. Placing the singular new vertex in the middle of that space and bending every edge previously connected to the two old vertices to follow the path of the original edge will thus ensure that the edges do not reside in any occupied space. This process is visualized below:

Therefore planarity is a minor-closed property and planar graphs are minor-closed.

Another important property that can be explored in the context of graph classes is degeneracy, which constitutes a form of sparsity since it limits the edge-vertex ratio.

**Definition 2.5:** A graph is $k$-degenerate if every subgraph has a node with degree, or number of incident edges, at most $k$. This can be expressed formally as for every subgraph $G'$ of a graph $G$, the minimum degree of any node of $G' \leq k$.

Degeneracy can be used to determine several important factors, such as whether the graph admits an orientation with some maximum number of edges pointing to one node or how many colors are needed to ensure no two nodes sharing an edge have the same color. Knowing that a class of graphs has a finite degeneracy will inform us that those values are finite as well. Planar graphs happen to be 5-degenerate, a fact which we will now demonstrate.

**Theorem 2.3:** Every subgraph of a planar graph has a node with degree at most five.

**Proof:** The following is a proof by contradiction of the theorem above. Assume that there exists a planar graph without a node with at most five edges. Therefore all nodes have six or more edges. Every edge is shared by two nodes so the total number of edges is at least three times the number of nodes, which can be expressed as $|G| \geq 3|G|$. A contradiction arises when this attribute is compared with the limit on the number of edges, namely that plugging this value into the place of the number of edges in the formula derived in theorem 2.1 gives us $3|G| - 6 \geq 3|G|$, and subtracting three times the number of vertices from both sides gives us $-6 \geq 0$. This result is impossible, so our assumption must be false. Therefore all planar graphs are 5-degenerate.

2.2 Bounded Expansion

Many sparse graph classes have the property of bounded expansion.

**Definition 2.6:** Graph classes with bounded expansion have a greatest reduced average density, denoted $\nabla_r(G)$, that is always less than or equal to some
consistent function on \( r \), denoted \( f(r) \), that works for any whole number. \( \nabla_r(G) \) is calculated by determining the maximum edge to vertex ratio of the graph \( G \) and all of its **\( r \)-shallow minors**, sometimes abbreviated as **\( r \)-minors**.

**Definition 2.7:** The ‘\( r \)’ in **\( r \)-shallow minors** denotes the number of edges, or **radius**, by which nodes can be separated from a central vertex before they are not allowed to be combined through edge contraction; edge and vertex deletions are always allowed. Any number of central vertexes can be selected and nodes contracted around them as long as the subgraphs formed by the connected nodes are **pairwise disjoint** between centers, meaning that they share no nodes. A series of examples illustrating **\( r \)-shallow minors** follows.

---

**Fig 1:** The original graph used in these examples.

**Fig 2:** Two possible subgraphs or 0-shallow minors.

**Fig 3:** A series of images depicting the transformation of the original graph into a 1-shallow minor. The first image highlights the edges about to be contracted in blue. These contractions are centered around the nodes \( i \) and \( a \). Note that not all edges connected to node \( a \) have been selected for contraction. The second image depicts the result of the contraction and is a perfectly valid 1-minor in and of itself, but the third depicts another 1-minor resulting from further edge and vertex deletions performed on the second image.
Fig 4: A pair of images depicting the transformation of the original graph into a 2-shallow minor. The first image highlights the edges about to be contracted in blue. These contractions are centered around the nodes b and m. The second image depicts the resulting 2-minor.

Let us revisit planar graphs, which in addition to being sparse have the property of bounded expansion. We will now show that there is a bound of three on the $\nabla_r(G)$ of planar graphs.

**Theorem 2.4:** The class of planar graphs has bounded expansion with $f(r) = 3$ for $f(r) \geq \nabla_r(G)$.

**Proof:** The formula $3|G| - 6 \geq ||G||$ implies $||G|| < 3|G|$. Dividing both sides by $|G|$ gives us $||G||/|G| < 3$. Planarity is a minor-closed property and $r$-shallow minors are simply a more limited version of minors, so the limit $||G||/|G| < 3$ is true for all $r$-shallow minors of a planar graph. The $\nabla_r(G)$ of such a graph can never be greater than three. Therefore planar graphs have bounded expansion, with $\nabla_r(G) \leq f(r)$ satisfied by $f(r) = 3$.

Bounded expansion is linked to many properties related to subgraphs or minors, including degeneracy. We will now show that all graphs are actually $2\nabla_0(G)$-degenerate.

**Theorem 2.5:** Let $G$ be any graph with for which $\nabla_0(G)$ has been calculated. Then that graph and all of its subgraphs have at least one node with degree no greater than $2\nabla_0(G)$.

**Proof:** The $\nabla_0(G)$ of a graph is the maximum edge to vertex ratio of all the 0-shallow minors. 0-shallow minors are equivalent to graphs obtained from vertex and edge deletion, which means they merely constitute all the possible subgraphs of the graph. Because each edge is shared by two nodes the average degree of the nodes is $2\nabla_0(G)$. A series of values whose average is $2\nabla_0(G)$ must contain at least one value equal or less than $2\nabla_0(G)$. So every subgraph contains a node with degree at most $2\nabla_0(G)$ and therefore the graph is $2\nabla_0(G)$-degenerate.

For graphs belonging to classes with bounded expansion the variable $\nabla_0(G)$ is bounded by some finite value determined by the corresponding $f(0)$. Therefore we can assume when dealing with those classes that every graph is $2f(0)$-degenerate and can treat them similarly in algorithms and implementations. This and other derivable factors will ultimately enable us to run traditionally polynomial time algorithms at much faster paces over these graphs.

However, remember that we showed planar graphs were all 5-degenerate in theorem 2.3. Plugging in the value for $\nabla_0(G)$ for all planar graphs reveals that planar graphs are also 6-degenerate. This is not a contradiction since by the definition of degeneracy, any graph that is $d$-degenerate for some integer $d$ is also $d'$-degenerate for all integers $d'$ greater than $d$. This demonstrates an important concept, namely that our
calculations will be based on bounds for graph classes and therefore the worst-case scenarios. It is possible to create a specialized solution for a certain graph class or a particular graph, but by always assuming we are in the most difficult scenario we guarantee the correctness of our proofs for all graph classes with bounded expansion and any graph that belongs to them.

3 Implicit Graph Representations
3.1 Linear Time Calculation of Implicit Representations

The concept of linear time means that in the worst-case scenario it takes at most a number of operations proportional to the size of the graph, usually measured as the number of nodes, to calculate the final result.

Definition 3.1: An algorithm’s time complexity is represented as \( O(f(n)) \) if the number of operations is at most \( f(n) \) multiplied by some constant, where \( n \) is the size of the input. An algorithm with constant time complexity has \( f(n) = 1 \), whereas one with linear time complexity has \( f(n) = n \) and one with polynomial time complexity has \( f(n) = n^k \) for some \( k \).

In the context of this senior thesis topic, preprocessing is used to make queries addressed to graphs as easy and quick to solve as possible. The graph in question may be altered in many different ways but the number of steps taken will be proportional only to the size of the graph. This results in a graph primed so that first-order logic queries can be solved in linear time, requiring only a set, finite number of operations regardless of how complex the graph itself gets. This is predicated on the idea, proven in [5] but not reproduced here, that if the transformed graph satisfies a query so would the original graph.

As an example, finding out if a graph is triangle free or equivalently that it contains no three nodes that are mutually connected to one another, is normally an \( O(n^3) \) problem. The brute force solution would involve checking each node, every node connected to that node, and finally every node connected to the third node. Even advanced techniques cannot move triangle-finding algorithms up to or past \( O(n^2) \) on arbitrary graphs [5]. However, the triangle free problem is solvable in \( O(n) \) operations when dealing with graph classes with bounded expansion and using the techniques put forth in this thesis.

A problem will be solvable in linear time if it requires addressing \( n \) individual queries that take a constant time to solve, where \( n \) is the number of nodes in the graph. Converting a graph to its implicit representation is one of the essential preprocessing steps that enable constant time queries.

Definition 3.2: A graph class has an implicit representation if for every graph in the class there is a way to represent the vertices using an amount of space proportional to \( n \) where vertex adjacency can be tested in constant time.

Most but not all sparse graph classes have an implicit representation [11], and it is simple to show that graphs belonging to a class with bounded expansion have an implicit representation.
**Theorem 3.1:** Any graph $G$ belonging to a class of graphs with bounded expansion can be represented in an amount of space proportional to $n$ where vertex adjacency testing takes constant time.

**Proof:** Go through the graph $G$ and record each node’s number of edges. This will take linear time and use space proportional to $n$. Select a node with at most $2\Delta_0(G)$ edges. We know that such a node exists because, as was shown in theorem 2.5, all graphs are $2\Delta_0(G)$-degenerate and the definition of degeneracy ensures the existence of such a node. Since the graph belongs to a class with bounded expansion, $2\Delta_0(G)$ is a finite bounded quantity. We do not even have to examine the graph to calculate its $\Delta_0(G)$; instead we can merely plug in the $f(0)$ for the particular class to which the graph belongs.

Record the adjacent nodes and store that information with the node. Remove that node from the graph and update the adjacent nodes’ recorded edge quantities. The resulting graph $G’$ is a subgraph of the original. Since $\Delta_0(G)$ is determined by the maximum edge-vertex ratio of a graph and all of its subgraphs, it is not possible for the $\Delta_0(G’)$ of this new graph to be greater than the $\Delta_0(G)$ we selected for the original configuration. Therefore this new graph is $2\Delta_0(G)$-degenerate and it must have a node with at most $2\Delta_0(G)$ edges.

Continue selecting, recording and removing until all nodes are gone. Since we only examined each node once we have taken $O(n)$ time. The resulting structure takes up no more than $2\Delta_0(G)n$ space. It is always possible to test whether two nodes are adjacent by examining the information stored with both nodes, since one of those two nodes will have recorded that they are adjacent or neither of them will have. This use of space proportional to $n$ and capacity for constant time vertex adjacency testing fulfills the requirements for being an implicit representation. Therefore we can record an implicit representation of all graphs belonging to classes with bounded expansion in linear time.

4 Transitive Fraternal Augmentation

4.1 Concept Overview

Transitive fraternal augmentations are another way to preprocess a graph that alters the distance between nodes in a regimented, observable manner. A transitive fraternal augmentation inserts edges into the original graph such that fewer edges separate nodes. After enough transitive fraternal augmentations all nodes that were connected to each other in any way are now adjacent or separated only by a single node in-between. This result is guaranteed despite the fact that the process of transitive fraternal augmentation gives a partial choice of which edges to insert. Furthermore, although a transitive fraternal augmentation is implemented on a directed version of the graph and can proceed in different ways depending on that initial orientation, the effect of a single step in this process of increasing adjacency is consistent and reveals specific information about the distance between the nodes on the original graph. Transitive fraternal augmentations can also proceed after recording only a small amount of information about the original graph. This, combined with their linear runtime, allows for
slow and specific gathering of adjacency-related information that eliminates the need for a much more costly preprocessing step

**Definition 4.1:** When a graph undergoes a **transitive fraternal augmentation** all of the edges of the graph are oriented and every pair of nodes \((x, z)\) is examined. In the **transitive** step, if there is an edge pointing from node \(x\) to node \(y\) and from node \(y\) to node \(z\), a new edge is added going from \(x\) to \(z\). In the **fraternal** step, if there is an edge from node \(x\) to node \(y\) and node \(z\) to node \(y\), a new edge is added either from \(x\) to \(z\), \(z\) to \(x\), or in both directions. The undirected version of the resulting graph is called an **augmentation**.

Maximum in-degree is a very important concept in the context of transitive fraternal augmentations.

**Definition 4.2:** The **maximum in-degree** of a directed graph, denoted \(\Delta^-\), is the **in-degree** of the node or nodes with the greatest number of edges pointing inwards in this particular orientation. The number of edges pointing inward equate to the node’s own **in-degree**.

We can establish that it is possible for us to create orientations of any graphs with maximum in-degree \(\Delta^- = 2\Delta_0(G)\) in linear time since, as shown in theorem 2.5, all graphs are \(2\Delta_0(G)\)-degenerate. By repeating the proof of theorem 3.1 but orienting edges instead of removing them, and then ignoring all oriented edges afterwards, it can be shown that all graphs belonging to classes with bounded expansion have such an orientation.

**Theorem 4.1:** An orientation with maximum in-degree \(2\Delta_0(G)\) can be constructed in linear time for all graphs.

**Proof:** Go through the graph and record each node’s number of edges. This will take linear time. Select a node with at most \(2\Delta_0(G)\) edges. All graphs are \(2\Delta_0(G)\)-degenerate and such a node will always be present in a \(2\Delta_0(G)\)-degenerate graph. Orient all of the edges of that node inward and ignore them in further dealings with this representation of the graph, such that the nodes sharing those edges have their edge count reduced by one. The resulting graph is a subgraph of the original, and by the definition of degeneracy also contains a node with at most \(2\Delta_0(G)\) edges. The process can be repeated with such nodes until every node has been visited. Since edges were always oriented inwards and never outwards no node has an in-degree greater than the number of edges at its time of selection, which was always at most \(2\Delta_0(G)\). Since we only examined each node once we have taken \(O(n)\) time.

Therefore we can substitute \(2\Delta_0(G)\) for \(\Delta^-\) when dealing with any graph, assuming a linear time preprocessing step has taken place.

### 4.2 Path Length Reduction

Transitive fraternal augmentations can reduce path length on undirected graphs.

**Definition 4.3:** There is a **path** between two nodes if one node can be reached from the other by travelling along edges in either direction. The number of edges
travelled denotes the **length** of the path. The **distance** between two nodes is equal to the minimum **length** of all **paths** between them.

Suppose the minimum length of a path between two nodes is $r$, with $r \geq 3$. After a single transitive fraternal augmentation the distance between the two nodes will be $\leq r - 1$. We will first examine paths of length three as a model for the effects of a transitive fraternal augmentation on all paths of length greater than three. We will now show that all orientations for paths of length three result in the creation of paths of length two between the same nodes after a transitive fraternal augmentation [6].

**Lemma 4.1:** Two nodes at distance three will be at distance two after a transitive fraternal augmentation, regardless of the orientation of edges in the path that connects them.

**Proof:** Here are half of the possible orientations for paths of length three that would cause two nodes to be at distance three. We envision them as paths from a node $w$ to a node $z$ passing through separate nodes $x$ and $y$.

![Fig 1: Four of the eight possible oriented paths of length three. The remaining orientations would constitute mirror images of the ones diagrammed above, comparable to flipping them around a bisecting vertical line. Alternatively, a reversal of each and every edge orientation would also produce the remaining oriented paths.](image)

Here are those paths, divided into groups based on how they become paths of length three after a transitive fraternal augmentation. In some cases the paths would be transformed transitively and fraternaly, but only one transformation is shown.

**Case (A) and (B):**

![Case (A) and (B):](image)

**Case (D):**

![Case (D):](image)

Fig 2: Two oriented paths of length three, including induced transitive edges. Note that the first path could also induce a fraternal edge in case B and a second transitive edge in case A, although neither is depicted.

In two of the above cases a new transitive edge will be created connecting the first node to the third node, regardless of the orientation of the third edge. In the last
case a new transitive edge will be created connecting the second node to the fourth node.

Case (C):

In the above case at least one of two possible edges will be created to link the first node or the third node to the other, regardless of the orientation of the third edge.

When traveling along the path in the undirected graph, traversing the newly added edge allows the destination to be reached while passing at least one less node than before. Since no edges are ever removed in a transitive fraternal augmentation, the minimal length of a path cannot be increased in any way by the process and the end result is always a length reduction of at least one edge.

When an interior path of length three has been shortened to a path of length two, taking this new shortcut while traveling down the complete path will have the effect of decreasing its length by one edge. We will now show that all paths of length greater than three have their length reduced by at least one after a transitive fraternal augmentation.

**Theorem 4.2:** Two separate nodes at distance three or greater have the distance between them reduced in length by at least one after a transitive fraternal augmentation.

**Proof:** If the distance between two paths is greater than or equal to three it means that the smallest path between those two nodes has length equal to that distance. Since it is the smallest path we know it does not double back onto itself and revisit the same node, or else there would be a smaller path which immediately takes the edges which the original path only took on its second time visiting that node. As such, the path between the two nodes can be conceptualized as a straight line visiting distinct nodes only once.

Since the length of that path is greater than or equal to three, there must be a subsection of that path that consists of four nodes that themselves create a path of length three from the first to the fourth node. Lemma 3.1 shows that all paths of length three become paths of length two after a transitive fraternal augmentation. If the length of this subsection is reduced, the total length of the path will be reduced as well, and therefore the distance between the two nodes will have been reduced.

Multiple transitive fraternal augmentations will continue to reduce the distance between nodes within the graph until there is at least one path of length at most two connecting every pair of nodes.

4.3 Linear Time Calculation

As stated earlier in section 3.1, it is crucial that our preprocessing takes time proportional to the size of the graph such that a result can be obtained in linear time. As a potential candidate for that preprocessing, transitive fraternal augmentations are only
useful if it is possible to perform one on a graph belonging to a class with bounded expansion in linear time. We will first demonstrate one way to add the transitive edges to a graph using a method with a linear time complexity [8].

**Theorem 4.3:** Transitive edges can be added to a representation of a graph in $O(n)$ time, where $n$ is the number of nodes.

**Proof:** Access the representation of the graph and select a node.

Select an incoming edge to that node and examine the node connected to it.

Select the incoming edge to that second node and note the node connected to that one.

Record the existence of a new edge from the last selected node to the first node.

Iterate through all incoming edges to the second node, all choices of the second node with incoming edges to the first, and all choices for the first node. This is the exact method used to determine the need for a transitive edge. Since $\Delta^-$ is the maximum in-degree of every node, there are at most $\Delta^-$ choices for the last node, $\Delta^-$ choices for the second node, and there are $n$ nodes, the maximum number of operations is proportional to $(\Delta^-)^2n$ and the process takes $O(n)$ time.

We will now show that it is also possible to add the fraternal edges in linear time [8].

**Theorem 4.4:** Fraternal edges can be added to a representation of a graph in $O(n)$ time, where $n$ is the number of nodes.

**Proof:** Create an arbitrary linear order for nodes in the graph. Access the representation of the graph and select a node.

Select an incoming edge to that node and note the node connected to it.

Select a second incoming edge to the first node and note the node connected to it as well.

If the first node comes before the second in the arbitrary order, add an edge from the first to the second.
This is the exact method used to determine which of the two potential fraternal edges is added. Iterate through all pairs of incoming edges and all choices for the first node. Since $\Delta^-$ is the maximum in-degree of every node, there are at most $\Delta^-$ choices for the each of the two incoming edges, and there are $n$ nodes, the maximum number of operations is proportional to $(\Delta^-)^2n$ and the process takes $O(n)$ time.

Since a transitive fraternal augmentation consists solely of adding transitive and fraternal edges, the total transformation can be done in time $O(n + n)$ or $O(n)$. We can establish a value for $\Delta^-$ independent of $n$ because theorem 4.1 shows $\Delta^- \leq 2\nabla_0(G)$ if we allow for a linear time preprocessing step. This means our total time to perform a transitive fraternal augmentation is $O(n) + O(2\nabla_0(G)n + 2\nabla_0(G)n)$ or $O(4\nabla_0(G)n)$. Because $\nabla_0(G)$ is bounded by a specific finite number for graphs belonging to a class with bounded expansion the time taken is a product of a constant and $n$, which means it is linear by definition. Therefore we can conclude that there is an algorithm that can add the edges generated by a transitive fraternal augmentation in linear time to a graph with the property of bounded expansion.

4.4 Repeatability

Bounded expansion and maximum in-degree have a complicated, intertwined relationship. We can in fact use the maximum in-degree of some orientation of a graph as a bound on the $\nabla_0(G)$ of that graph, and extend this result to show that graphs belonging to classes with bounded expansion still belong to such a class after transitive fraternal augmentations. We will now establish that all possible maximum in-degrees of any given graph $G$ are greater than or equal to $\nabla_0(G)$.

**Theorem 4.6:** Given some maximum in-degree $\Delta^-$ from any possible orientation of a graph $G$: $\Delta^- \geq \nabla_0(G)$.

**Proof:** Given a directed graph $G$ with maximum in-degree $\Delta^-_G$ for this specific orientation, the maximum in-degree $\Delta^-_{G'}$ of any subgraph $G'$ must be less than or equal to $\Delta^-_G$, as such a subgraph can have fewer nodes and edges but never more. This can be expressed as $\Delta^-_{G'} \leq \Delta^-_G$ for all $G'$. Since it is also possible for some nodes to have in-degree less than the maximum degree, we can say that the product of the maximum in-degree and the number of nodes is less than or equal to the number of edges. We can express this as $|G'| \leq \Delta^-_{G'} \leq \Delta^-_G$ for all $G'$. If we divide both sides by $|G'|$ we get $\|G'|/|G'| \leq \Delta^-_{G'}$ for all $G'$. $\nabla_0(G)$ is defined as the maximum edge-vertex ratio of all the 0-shallow minors (which are equivalent to subgraphs as they only allow edge and vertex deletion, not contraction), or $\|G'|/|G'|$ over all $G'$ of $G$. We know that $\|G'|/|G'|$ is always less than or equal to $\Delta^-_{G'}$, and all $\Delta^-_{G'}$ are less than or equal to $\Delta^-_G$. Therefore $\nabla_0(G) \leq \Delta^-_G$ for any $G$.

This result can also be stated as $\Delta^- \geq \nabla_0(G)$ for all possible $\Delta^-$ and all graphs $G$ [7].
We will proceed by showing the existence of a reasonable bound on the maximum in-degree of a graph after it undergoes the initial transitive fraternal augmentation, which will create a bound on its $\nabla_0$ as a result of the relationships established in theorem 4.6. This will form the basis of a proof that graphs belonging to a class with bounded expansion still belong to some class with bounded expansion after a transitive fraternal augmentation and help us calculate its new $f(r)$. With this in hand it would be possible to perform any number of transitive fraternal augmentations on a graph and still be certain that other proofs throughout this thesis that are reliant on bounded expansion hold for the resulting graph.

Logically the maximum in-degree of a graph after a transitive fraternal augmentation is the sum of the maximum in-degree of a directed version of that graph and the number of edges that are added through the transitive and fraternal steps. We will first show that the number of edges added through the transitive step is limited by the maximum in-degree of the graph.

**Lemma 4.2:** The transitive step can add at most $\Delta^2$ edges to any node in a graph with maximum in-degree $\Delta$.

**Proof:** The transitive step adds an additional edge from node x to node z if there is an edge pointing from x to node y and from y to z. There are at most $\Delta^2$ edges pointing towards each node in the graph. Each collection of nodes that have edges pointing towards a given node can themselves only have at most $\Delta^2$ edges pointing towards them. In other words, there can be at most $\Delta^2$ y-to-z edges for a given node z, and there can be at most $\Delta^2$ x-to-y edges for the set of nodes y. The total number of pairs is equal to the product of the two numbers, so the transitive step can add at most $\Delta^2$ edges.

We will now show that the $\nabla_0$ of the augmentation of a graph with bounded expansion is itself bounded.

**Theorem 4.7:** For a graph G belonging to a class with bounded expansion and a directed version of that graph G’, the augmentation of the graph H has $\nabla_0(H)$ bounded by some finite integer.

**Proof:** The maximum in-degree of a graph after a transitive fraternal augmentation can be derived from the in-degree of the original graph and the number of edges added in the transitive and fraternal steps. We know that the maximum in-degree of the original graph G can be set to $2\nabla_0(G)$ in linear time thanks to theorem 4.1, and if we plug that result into lemma 4.2 it gives us the maximum in-degree increase due to transitive steps, $4\nabla_0(G)^2$. However, we intend to use theorem 4.6 to establish a bound on $\nabla_0(H)$. According to that theorem, such a bound can be established by any maximum in-degree as long as it corresponds to some orientation of the graph. A corollary to theorem 4.6 not proven in this thesis is that every graph G admits an orientation with maximum in-degree at most $\nabla_0(G)$ [3], which means we can set the in-degree increases to $\nabla_0(G)$ and $\nabla_0(G)^2$ respectively. This leaves the fraternal step. It would be trivial to show that the number of edges added in the fraternal step is at most the number of nodes multiplied by the maximum in-degree of the original directed graph squared. However, the number of nodes is not bounded for all graphs belonging to a class with bounded expansion, and we must resort to other means.
The original graph is $2\nabla_0(G)$-degenerate as per theorem 2.5. This also means it is possible to color all the nodes of $G$ with $2\nabla_0(G) + 1$ colors such that no two nodes that share an edge have the same color [3]. In practice this can be achieved through a method nigh-identical to the one used in the proofs of theorem 3.1 or theorem 4.1, whereupon the nodes are colored as they are removed and in addition to the edge count of adjacent nodes being updated, the color used is disallowed for those nodes. None of the removed nodes could have been connected to more than $2\nabla_0(G)$ colored nodes at the time of their removal, and the potential case where the node was connected to exactly $2\nabla_0(G)$ colored nodes adds the single additional color to the sum.

Construct a coloring, and then begin to color all of the edges in $G'$. Each edge should be ‘colored’ with a specific two-item entry describing the color in $G$ of the node the edge points to in $G'$ and a second arbitrarily chosen color, with the limitation that all edges pointing to the same node must have a different second color. Defining the maximum in-degree of $G'$ as $\Delta^-$, there will be at most $(2\nabla_0(G) + 1)\Delta^-$ two-item colors for the edges of $G'$. If we chose to set the maximum in-degree of the original orientation to $2\nabla_0(G)$, we could simplify this to $4\nabla_0(G)^2 + 2\nabla_0(G)$. We can set this to $2\nabla_0(G)^2 + \nabla_0(G)$ if we chose to exploit the concepts discussed earlier when dealing with the original in-degree and transitive step-induced increase in in-degree, and this is indeed what we will do.

Note that the coloring of two edges that would induce a new transitive or fraternal edge must be different from one another since they either point towards two separate but adjacent nodes or to the same end node but from different initial nodes. Pick any pair of edge colorations that share the same initial color in their two-item entry. This is the only case that could possibly induce a fraternal edge in the augmentation, since it means that two of those edges may point to the same node.

Remove every node in $G'$ that is not at one of the ends of one such edge. The remaining graph consists of one or more unconnected groups of nodes with other nodes surrounding them with edges pointing inward, and possibly edges pointing between those outer nodes as well. Contract the edges so that every group of nodes in the graph consists of a single edge from one of the original outer nodes to a composite node. We have only performed edge and vertex deletions, plus edge contractions of radius one. The undirected version of the resulting graph is now a 1-shallow minor of $G$. $\nabla_1(G)$ is by definition the maximum edge-vertex ratio of any 1-shallow minor. Our procedures up to this point have been the equivalent of adding arbitrary fraternal edges and removing any other extant influence on in-degree, and the in-degree of any node in the resulting minor can be set to less than or equal to $\nabla_1(G)$ for some orientation [3]. Since theorem 4.6 shows that any orientation produces a maximum in-degree that is a bound on $\nabla_0(G)$, this value is sufficient for our purposes.

Since there are $2\nabla_0(G)^2 + \nabla_0(G)$ edge colors, if we iterate through every pair of colors we end up with $(2\nabla_0(G)^2 + \nabla_0(G)) \ast (2\nabla_0(G)^2 + \nabla_0(G) - 1) / 2$ combinations. This simplifies to $2\nabla_0(G)^4 + 2\nabla_0(G)^3 - \frac{1}{2}\nabla_0(G)^2 - \frac{1}{2}\nabla_0(G)$. Each combination can add at most $\nabla_1(G)$ edges to the maximum in-degree of the orientation of the augmentation. Because we are looking for a bound on $\nabla_0(G)$
which can be provided by the in-degree of any orientation, we can use this number as a consistent value for the increase in in-degree created in this fashion even though it is possible that the resulting maximum in-degree is different in specific cases. This leaves us with \((2\Delta_0(G)^4 + 2\Delta_0(G)^3 - \frac{1}{2}\Delta_0(G)^2 - \frac{1}{2}\Delta_0(G))\Delta_1(G)\) as a usable value for the increase in in-degree due to the fraternal step.

Combining this with our original findings, we have a value for the possible in-degree of the graph after a transitive fraternal augmentation:

\[
\Delta_0(G) + \Delta_0(G)^2 + (2\Delta_0(G)^4 + 2\Delta_0(G)^3 - \frac{1}{2}\Delta_0(G)^2 - \frac{1}{2}\Delta_0(G))\Delta_1(G)
\]

Since a graph belonging to a class with bounded expansion has a bound on \(\Delta_r(G)\) for all \(r\) in the form of \(f(r)\) and this formula consists only of that variable \(\Delta_r(G)\), the formula itself is bounded by:

\[
f(0) + f(0)^2 + (2f(0)^4 + 2f(0)^3 - \frac{1}{2}f(0)^2 - \frac{1}{2}f(0))f(1)
\]

Theorem 4.6 shows that this also acts as a bound on \(\Delta_0\) of the graph resulting from the transitive fraternal augmentation.

This result has demonstrated our ability to conclude important information about and extract bounded quantities from the augmentation of a graph. A bound on the \(\Delta_r\) of a graph belonging to a class with bounded expansion after a transitive fraternal augmentation can be calculated by a function on the \(\Delta_0\) of the graph and the \(\Delta_0\) of several sequential augmentations [4]. Such a bound could theoretically be derived from the proof above and independent confirmation that the augmentations belong to classes with bounded expansion, ensuring finite bounded values for their \(\Delta_0\) and \(\Delta_1\). However, the polynomial used as the basis of the bound within the literature has a different origin.

Rather than iterating through multiple transitive fraternal augmentations, the proof in question is based on adding edges to induce the adjacency of k-reachable nodes [3].

**Definition 4.4:** In a directed graph, two distinct nodes are said to be **k-reachable** if there are oriented paths from both nodes to a third node and the sum of the length of those paths is less than or equal to \(k\).

We will now show that there exists a polynomial bounding the \(\Delta_0\) of a graph after the induced adjacency of k-reachable nodes for any \(k\).

**Theorem 4.8:** The \(\Delta_0\) of the graph created when adjacency is induced between k-reachable nodes of a directed graph is less than or equal to the iterative polynomial \(P_k(x, y)\) which returns \(y\) when \(k\) is set to one and \(4(k-1)((P_{k-1}(x, y)+1)(x+1))^ky\) otherwise, with inputs \(\Delta^-\) and \(\Delta_{k-1}(G)\).

**Proof:** The following is a proof by induction that the polynomial \(P_k(\Delta^-, \Delta_{k-1}(G))\) is greater or equal to \(\Delta_0(H_k)\) for all \(k\) greater than zero, where \(H_k\) represents the graph with a new edge between any two k-reachable nodes that were not already adjacent in \(G\), derived from a specific initial \(G\) and specific initial oriented version of \(G\), \(G'\), with maximum in-degree \(\Delta^-\).

**Base Case:** \(H_1\) would be identical to \(G\), since the only possible combination of distinct nodes who have paths of length one and length zero respectively to the same node are already adjacent. Therefore \(\Delta_0(H_1)\) is equal to \(\Delta_0(G)\). \(P_1(\Delta^-, \Delta_{1-1}(G))\) returns \(\Delta_0(G)\), so \(P_k(\Delta^-, \Delta_{k-1}(G)) \geq \Delta_0(H_k)\) when \(k\) equals one.

**Inductive Hypothesis:** Assume that the polynomial \(P_i(x, y)\) is greater than or equal to \(\Delta_0(H_i)\) for all \(i\) greater than or equal to one and less than \(k\).
**Inductive Step:** As per theorem 4.6, the maximum in-degree of any orientation of a graph must be greater than its $\Delta_0$. The maximum in-degree of some orientation of $H_k$ is equal to the maximum in-degree of $G'$ plus the increase in in-degree due to the induced adjacency of k-reachable nodes.

Let us divide the nodes that qualify for k-reachability into three categories. The first consists of k-reachable nodes for which the sum of the lengths of the paths to the third node, as per the definition, is less than k. The edges added in this fashion, and the edges present in $G$, result in a graph with $\Delta_0$ equal to $\Delta_0(H_{k-1})$ according to the inductive hypothesis. An aforementioned corollary to theorem 4.6 not proven in this thesis is that every graph $G$ admits an orientation with maximum in-degree at most $\Delta_0(G)$ [3]. Thus, these nodes contribute $\Delta_0(H_{k-1})$ to our theoretical maximum in-degree.

The second category consists of nodes for which the length of one path is zero and the length of the other path is exactly k. Lemma 4.2 shows that the increase in in-degree from a transitive step is bounded by the product of the number of possible edges that induce the new transitive edge, which in that case was two. Since the two nodes in this category consist of one node with a path of oriented edges of length k pointing towards the other node, and the requirement of induced adjacency creates a new edge linking them directly, this situation is definitively comparable to the state of affairs that induces a transitive edge. It is trivial to extend that proof to show that adding a new edge to the set of edges that induce a new connection means that the previous product of in-degrees must be multiplied by the number of additional possible edges, which is still the in-degree of the new node. Therefore these pairs of nodes can add at most $(\Delta^-)^k$ edges to the maximum in-degree of the graph.

The final remaining category of nodes consists of nodes for which the length of both paths is greater than zero and the sum of the lengths is exactly equal to k. In order to derive the contribution from the induced adjacency of these nodes to the maximum in-degree of some orientation of $H_k$ and do so only in terms of bounded or fixed variables, we must examine certain features of the graph $H_{k-1}$.

For the sake of brevity let us refer to the formula $P_{k-1}(\Delta^-, \Delta_{k-2}(G))$, which we know is a bound on $\Delta_0(H_{k-1})$ from our inductive hypothesis, as simply P. $H_{k-1}$ is 2P-degenerate since its $\Delta_0$ is equal to 2P. This also means it is possible to color all the nodes of $H_{k-1}$ with $2P + 1$ colors such that no two nodes that share an edge have the same color, via principles demonstrated in the proof for theorem 4.7. Construct a coloring and then begin to color all of the edges in $G'$. Each edge should be ‘colored’ with a specific two-item entry describing the color of the node that the edge points to in $H_{k-1}$ and a second arbitrarily chosen color, with the limitation that all edges pointing to the same node must have a different second color. Since the maximum in-degree of $G'$ is $\Delta^-$, there will be at most $(2P + 1)\Delta^-$ two-item colors for the edges of $G'$.

Begin examining lists of the edge colorings that appear in a particular pair of paths responsible for the k-reachability of nodes in this category, ordered from the start to end node. It is inherently impossible for the lists of colors in these two paths to be identical. Either they represent two initial distinct paths that come
together at one or more node, in which case the color of the edges pointing to the conjoined node must be different, or they represent one larger path and a second shorter internal path with the starting node contained within the first and the same end node, in which case their lengths are different and their contents are irrelevant.

Select one combination of lengths and a specific pair of paths of those lengths. Focus on the longer path, or pick between them arbitrarily if they are of the same length. Collect all the individual edges that could be found in paths of the same length as the selected path in $G'$ that also contain the same edge colors in the same order. These paths have length less than or equal to $k-1$, since they are one of two paths the sum of whose length is $k$ and both of which have lengths at least one. Therefore all of their nodes are adjacent in $H_{k-1}$ and they must be mutually distinct. Because node color contributes to edge color and no node has two incoming edges of the same color, every pair of paths must be entirely different, containing none of the same nodes and edges but instead similar ones with the same colors, or share the same initial sequence of nodes but eventually split at some point into coincidentally identically colored sections that never recombine, or the paths in the pair must be completely identical. Repeat this process with the remaining path, but this time do not collect any edges that would constitute the last edge in the path. A combination of any two specific paths from the two collections could contain none of the same edges and nodes or they could contain the same initial node, since that is the only node for which the second edge-coloring influence would allow it to have the same color but then not disrupt the mutual distinctiveness of the two path colors. They cannot meet at the last node because such edges were removed during the collection of the second set.

Remove every node in $G'$ that is not at one of the ends of an edge from the two collections, and contract the edges from the collections so that every group of nodes in the graph connected by edges from the collection is now a single node. The distance from the start node of one of the two selected paths to the end node is either the length of the longer of the two involved paths or the length of the shorter path minus one. Those two lengths are both less than or equal to $k - 1$. Therefore the undirected version of the resulting graph is a $(k-1)$-shallow minor of $G$. If the two paths would have caused their starting nodes to achieve $k$-reachability, those nodes are now adjacent due to the uncollected last edge in the shorter path. $\nu_{k-1}(G)$ is by definition the maximum edge-vertex ratio of any $(k-1)$-shallow minor. [3] states that this ensures the existence of an orientation limiting maximum in-degree to $\nu_{k-1}(G)$ for this minor.

There are $2(k-1)$ pairs of integers greater than zero that add up to $k$. There are $(2P + 1)\Delta$ edge colors, and if every combination of colors is possible, every pair of integers that add up to $k$ is examined each time, and there exists at least one node that is featured in every one of these combinations, there is a total possible increase in the maximum in-degree of $2(k-1)((2P + 1)\Delta)^k \nu_{k-1}(G)$ from the process of ensuring adjacency of nodes in this category.

We have now finally derived a value for the influence on the maximum in-degree of some orientation from each of the three categories of nodes that are subject to induced adjacency due to their $k$-reachability. The sum of the three
derived values is \( \nabla_0(H_{k-1}) + (\Delta^-)^k + 2(k-1)((2P + 1)\Delta^-)^{k-1}(G) \). Replace \( P \) with \( P_{k-1}(\Delta^-, \nabla_{k-2}(G)) \) once more to get \( \nabla_0(H_{k-1}) + (\Delta^-)^k + 2(k-1)((2P_{k-1}(\Delta^-, \nabla_{k-2}(G)) + 1)\Delta^-)^{k-1}(G) \). We now have a value for a maximum in-degree of some orientation of the graph \( H_k \) that is composed of only bounded integers, including \( P_{k-1}(\Delta^-, \nabla_{k-2}(G)) \) which eventually returns the bounded integer result of the base case and only includes additional bounded integers before that, and is thus itself bounded. As theorem 4.6 states that \( \Delta^- \geq \nabla_0(G) \) for all graphs and any choice of maximum in-degree this value is a bound on \( \nabla_0(H_k) \). Through algebraic manipulation of the formula we derived that will not be repeated here but which is necessary to reach the conclusion in [3], we can set the formula for \( P_k \) where \( k \) is greater than one at the more compact \( P_k(x, y) = 4(k-1)((P_{k-1}(x, y)+1)(x+1))y \).

A proof that all graphs in a class with bounded expansion retain bounded expansion against a new \( f(r) \) after a transitive fraternal augmentation based on the polynomial derived above is contained in [3]. The \( f(r) \) in question is equal to \( P_{2(r+1)}(2\Delta^-, \nabla_{2r+1}(G)) \).

As a result of all of these proofs, we can state with certainty that if we perform any number of transitive fraternal augmentations on a graph belonging to a class with bounded expansion, it remains a member of some class of graphs with bounded expansion.

5 First-Order Logic and Quantifier Elimination
5.1 First-Order Logic Properties and the Introduction of Structures

Previously we investigated the preprocessing needed to have graphs ready to accept and quickly run through our queries. Now it is time to address the queries themselves. First-order logic is a system that allows us to ask almost any yes-or-no question about objects and their properties. Sentences in first-order logic can be composed of the traditional Boolean operators \( \land \), \( \lor \), and \( \neg \), as well as functions, predicates, and the universal and existential quantifiers. Queries using only the first five symbols take constant time to evaluate.

Functions, symbols of the form \( f(x) \) that take one or more elements and return another element, and predicates, symbols of the form \( P(x) \) that are true if element \( x \) has some property, allow graph-specific questions such as whether nodes \( x \) and \( y \) are connected to be expressed as \( f(x) = y \) or \( E(x, y) \).

**Definition 5.1:** A **functional symbol** such as \( f \) is a letter or other representation for a **function**. The **arity** of the functional symbol determines its inputs, or the number of free variables related to the function. For example, \( f(x) \) takes one input and is a unary predicate with an **arity** of one. A **predicate** is a specific kind of function that returns true if the input belongs to a corresponding subset of items. A **unary predicate** is a **predicate** that accepts a single input.

This can be used to build more complicated structures, like \( E(x, y) \land E(y, z) \), to check for the presence or absence of a path or another kind of connection. To illustrate the expression of an actual query using this format we will revisit the triangle free problem mentioned in section 3. It could be expressed using formal symbols as \( \forall x, y, z \neg(E(x, z) \land E(z, y) \land E(x, y)) \). This is a statement which returns true if and only if every node \( x \) in the
The graph is not part of a triangle. The preceding quantifier $\forall x, y, z$ is equivalent to $\forall x \forall y \forall z$ and indicates the presence of three quantifiers rather than a single quantifier.

We want to create a consistent terminology for the representation of graphs belonging to classes with bounded expansion and queries on them. We have shown that it is possible to create an orientation for a graph belonging to a class with bounded expansion with in-degree at most $2\delta(G)$. We therefore will represent oriented edges in such graphs with functional symbols $f_i(x)$, with $f_i(x)$ returning the $i$th node from which an edge points to node $x$, and $i$ being a number between 1 and $2f(0)$. In other words, we represent an oriented edge from node $a$ to node $b$ as $a = f_i(b)$ for some $i$. We can limit the total number of functions to $2f(0)$, since there will never be more than $2f(0)$ incoming edges for each node, placing a strict limit on the number of functional symbols used. We will also represent colors of nodes on the graph with unary predicates in the form $P_a(x)$, where $x$ represents a node and $a$ indicates the subset against which the predicate checks its input [5].

5.2 Quantifier Elimination

Quantifier elimination is a structured process through which a query expressible in first-order logic has its time complexity pared down. This logically necessitates the removal of the universal and existential quantifiers, which normally cause the equation to run through a list of all possible values. Since in our case all variables represent nodes, it takes $O(n)$ operations to run through a list of all possible nodes to determine some property of the entire graph rather than the state of specific individual nodes, as is the case when dealing with a $\forall x$. As a result the exponent on the time complexity is usually equal to the number of quantifiers, such that a first-order formula with two quantifiers has time complexity $O(n^2)$, one with three quantifiers $O(n^3)$, and so on. Even the existential quantifier $\exists x$ increases the time complexity because in the worst-case scenario, where the last iteration investigated is the only one that is true or all items return false, the number of operations is the same as for the universal quantifier. Quantifier elimination is thus an essential part of reducing first-order queries to linear time.

Quantifier elimination to ensure linear time complexity over classes of graphs with bounded expansion takes the form of specific simplifications performed on the problem until the formula is simple and only one variable is subject to any functional symbols or unary predicates.

Definition 5.2: A formula is simple if it contains no functional symbols themselves used as the input to another functional symbol, as in the case of $f(g(x))$.

The techniques involved in this transformation can be organized into three categories. The first are the simplifications that merely involve algebraic manipulation, such as combining the elements of the formula $f(x) = y \land g(y) = z$ into $g(f(x)) = x$. The second are the simplifications that rely on the results of transitive fraternal augmentations on the original graph. We can convert $g(f(x)) = x$ into $h(x) = x \land P_{gH}(x)$ where $h(x)$ represents a oriented edge to node $x$ present in the augmentation of the graph and $P_{gH}(x)$ is a unary predicate that returns true if $x$ has a oriented edge pointing directly to it in the
augmentation and also had a path of length two of oriented edges pointing to it in the original graph [5]. Finally, there are the alterations that divide the parts of the formula into specific sections, such as formulas containing a not sign and those without one. This serves to make the iterative process of quantifier elimination simpler and provide a sensibly sorted final product.

A full breakdown of quantifier elimination as it relates to bounded expansion and the necessary proofs involved is beyond the scope of this thesis and can be found in [5].

6 Solutions in O(n)
6.1 Triangle Free on Planar Graphs

Up to this point in the thesis we have analyzed the graphs, algorithms, and techniques applied to said graphs and algorithms that are needed in order to run traditionally polynomial time first-order algorithms in linear time over classes of graphs with bounded expansion. Let us illustrate their true potential via specific non-trivial examples that will depict these concepts and processes.

Section 2 noted the existence of a class of graphs called planar graphs which were shown to have bounded expansion with \( f(r) = 3 \) for all \( r \) in theorem 2.4 and to be 5-degenerate in theorem 2.3. Sections 3 and 5 bought up a problem known as triangle free which checked for the nonexistence of three interconnected nodes anywhere in the graph and which had time complexity \( O(n^3) \) and first-order formula \( \forall x, y, z \neg (E(x, z) \land E(z, y)) \). Since planar graphs have bounded expansion and the triangle free problem is expressible in first-order logic it is the premise of this thesis that there is a way to run the triangle free problem on planar graphs in linear time.

We will begin by examining how the most basic aspects of graph preprocessing proceed in this case and referencing the portions of the thesis most directly relevant to their operation. According to theorem 4.1, it is possible to construct an orientation with maximum in-degree \( \Delta_0(G) \) on any graph belonging to a class with bounded expansion in linear time. This means that for any planar graph it is possible to construct an orientation with maximum in-degree six in linear time, since \( \Delta_0(G) \) is at most three for any planar graph. However, that proof was predicated on the fact that all graphs are \( 2\Delta_0(G) \)-degenerate as established in theorem 2.5. We know planar graphs to be 5-degenerate and thus in their case we can proceed through the instructions given in the proof of theorem 4.1 while selecting nodes with at most five edges each time instead of nodes with at most \( 2\Delta_0(G) \) edges and be sure that we create an orientation with maximum in-degree five on any planar graph. Furthermore, theorem 4.1 was an adaptation of theorem 3.1, which depicted the creation of an implicit representation. If we proceed through the instructions given in the proof of theorem 3.1 as well, we can create an implicit representation in which adjacency can be tested in constant time. Our method of recording adjacency through storing the information about nodes adjacent to node \( x \) at the time of its removal could do so through the use of functions \( f_1(x) \) through \( f_n(x) \), which return \( y \) if the \( n \)th edge connected node \( x \) to \( y \), as was suggested in section 5.1. Since at most five edges are recorded there will not be more than five functions per node, and these functions constitute a virtual directed version of the original graph, with edges oriented inwards from \( y \) to \( x \).
For the sake of clarity, we will now explicitly state what steps would be taken to create the graph representation described above while dealing with a planar graph. We would iterate through the nodes of the graph and record the degree of each node, which was shown in the proof of theorem 3.1 to take linear time. We will then select a node with at most five edges and record functions $f_1(x)$ through $f_5(x)$ that return the nodes on the other end of the five edges of $x$. Even if that information was not already known it would take constant time to check, as there are only five edges to investigate. The existence of the function $f_1(x) = y$ is understood to mean that the edge from $x$ to $y$ would be oriented inwards, pointing towards $x$. The node and edges are then removed from the graph and another node with at most five edges, whose existence is ensured due to the 5-degeneracy of planar graphs established in theorem 2.3, would be selected. This process would continue until all nodes have been removed, and theorem 3.1 establishes this still takes linear time. Throughout this process no more than five functions have been created for any node, and given that the functions represent inwardly oriented edges the maximum in-degree of this graph is at most five. Having no more than five functions per node also helps ensure we are using space proportional to $n$ for our representation. Therefore we have created a directed implicit representation of the planar graph in linear time.

This approach can be generalized for any graph belonging to a class with a known degeneracy that applies to the entire class, with the sole deviation of picking a node with edges numbering at most the value of that applicable degeneracy. That value would likewise propagate through the instructions and replace five as the maximum number of functions and maximum in-degree. This generalizability is a given for graphs belonging to classes with bounded expansion, which are guaranteed $2\nabla_0$-degeneracy for the value that is a bound on $\nabla_0$ for the entire class, namely $f(0)$.

We proceed by breaking down the triangle free problem to work in this context. The triangle free problem has the first-order formula $\forall x, y, z \neg(E(x, z) \land E(z, y) \land E(x, y))$. The symbol $E(x, y)$ signifies the existence of an edge between two nodes $x$ and $y$. The existence of an edge connecting $x$ and $y$ will have been recorded in the form of one and only one of the two possible equivalences, $f_i(x) = y$ or $f_j(y) = x$, for some value of $i$ or $j$, because each edge has been oriented in exactly one direction. Therefore we could replace $E(x, y)$ with $(f_i(x) = y \lor f_j(y) = x)$ and do the same for every edge mentioned in the formula to produce the new, equivalent first-order formula:

$$\forall x, y, z \neg(\forall i \leq 5 (f_i(x) = z \lor f_i(z) = x) \land (f_k(x) = y \lor f_l(y) = z) \land (f_m(x) = y \lor f_n(y) = x)),$$

The statement at the end of this formula is shorthand indicating that when the formula is run, it should be tested for every value and combination of values for the variables $i$, $j$, $k$, $l$, $m$, and $n$ between and including one and five. We could achieve the same effect by replacing any or all individual equivalences, such as $f_i(x) = z$, with five specific equivalences signifying each value, which in that case would be $(f_1(x) = z \lor f_2(x) = z \lor f_3(x) = z \lor f_4(x) = z \lor f_5(x) = z)$. Regardless, we now see exactly how to phrase the first-order formula so that it works on the established representation of the graph.

The particular problem of whether or not a graph is triangle free has specific attributes we can exploit to simplify our task, namely our need to check every node in the graph and the symmetry of a triangular structure. Because function-based equivalences of the form $f(x) = y$ takes constant time to check in our implicit representation, and the new
first-order formula for the triangle free problem consists only of functions and Boolean symbols which take constant time to run, any single iteration of the first-order formula for a specific $x$, $y$, and $z$ takes constant time not proportional to the size of the graph. We are currently checking every set of nodes $x$, $y$, and $z$ and every combination of orientations between those nodes. Examining each set of nodes is in fact what gives our algorithm time complexity $O(n^3)$, as was described in section 5.1. We will now show that if we check every node $x$, all the possible orientations for edges of triangles that $x$ is a part of will be represented in one of two orientations when examining one of the three nodes as $x$.

**Lemma 6.1:** When determining whether a graph contains any triangles, checking for the existence of triangles with the specific two orientations in figure 1 below is equivalent to checking for the existence of triangles with any possible orientation if all nodes of the graph are examined.

**Proof:** A triangle has three edges that can be oriented in two ways, so there must be $2^3$ or eight different orientations. Here are two of the eight possible orientations of edges in a triangle with nodes $x$, $y$, and $z$:

![Fig 1: Two of the eight possible triangle orientations in a directed graph.](image)

Two of the other orientations can be flipped around a line intersecting node $x$ and the edge between nodes $z$ and $y$ so that their edges point in the same relative direction as orientations in the figure above.

![Fig 2: Two sets of triangle presented in the order of the particular member of the remaining six orientations, that triangle after it has been flipped, and the corresponding triangle whose edge orientations they now match, which are the first and second orientation in Fig 1 respectively.](image)

The remaining four orientations can be rotated and flipped so that their edges point in the same relative direction as orientations in the first figure.
Fig 3: Four sets of triangle presented in the order of the member of the remaining four orientations, that triangle after it has been rotated and if necessary flipped, and the corresponding triangle whose edge orientations they now match, which is always the second orientation in Fig 1. The last two sets require both rotating the graph and flipping it, or else they would merely correspond to the orientation in the first column of the second set in Fig 2.

In the case of the items in the first column of figure 2, an algorithm based around the corresponding orientation in figure 1 would still perceive the same triangle because the orientations are indistinguishable from the perspective of node x. In the case of the first and fourth item in the first column of figure 3, the algorithm would be able to perceive a triangle at that orientation when node z is examined and treated like node x in the original set of triangle orientations, even if a triangle would not be detected when looking at the original node x. The same is true in the case of the second and third item when treating node y as node x. Thus, only the first two orientations are needed if we examine every node.

We want to run our algorithm in linear time, which could be achieved if we only checked every individual node instead of every set of three nodes. We will now show that it is possible to construct an equivalent first-order formula that only uses a single variable.

**Theorem 6.1:** The first-order formula that takes only one variable \( \forall z \sim (f_i(f_j(f_k(z))) = z) \vee (f_l(f_j(z)) = f_k(z)) \), \( 1 \leq i, j, k, l \leq 5 \) returns true if and only if there are no triangles in the graph on which it is run.

**Proof:** Lemma 1 shows that we only need to deal with two orientations of a triangle’s edges when detecting the existence of any triangles if we iterate through all the nodes in a graph. We could therefore alter our formula to only examine those two orientations.

We can cover the first orientation in figure 1 of lemma 6.1 with the formula \( f_i(y) = x \land f_j(z) = y \land f_k(x) = z \). Likewise, we can cover the second orientation with the formula \( f_l(y) = x \land f_m(z) = y \land f_n(z) = x \). These formulas would be separated by or statements as they do not all have to be true to indicate a triangle, but instead only a single one has to be true. The full formula would look like:

\[
\forall x, y, z \sim (f_i(y) = x \land f_j(z) = y \land f_k(x) = z) \lor (f_l(y) = x \land f_m(z) = y \land f_n(z) = x), 1 \leq i, j, k, l, m, n \leq 5
\]
Looking at the portion of the formula corresponding to the first orientation, we see that we can simplify the statement \( f_i(y) = x \land f_j(z) = y \) to \( f_i(f_j(z)) = x \), and then do likewise to the first and second function in the other portion. This results in formulas \( f_i(f_j(z)) = x \land f_i(x) = z \) for the first orientation and \( f_i(f_m(z)) = x \land f_i(z) = x \) for the second. These formulas can be further reduced to \( f_i(f_i(f_j(z))) = z \) and \( f_i(f_m(z)) = f_i(z) \) respectively. The full formula now looks like this:

\[
\forall x, y, z \neg (f_k(f_i(f_j(z))) = z) \lor (f_i(f_j(z)) = f_l(z)), \ 1 \leq i, j, k, l, m, n \leq 5
\]

Notice that nodes \( x \) and \( y \) no longer appear in the formula. This means we can remove the quantifiers handling \( x \) and \( y \).

Remember that these formulas represented the first and second item in figure 1. Those orientations share the orientation of edges pointing from \( x \) to \( y \) and from \( y \) to \( z \). This means we can use the same functions for both halves of the remaining formula. The resulting formula looks like:

\[
\forall z \neg (f_i(f_i(f_j(z))) = z) \lor (f_i(f_j(z)) = f_i(z)), \ 1 \leq i, j, k, n \leq 5
\]

Since our choices of variable names were arbitrary and merely needed to be consistent within any one iteration of the formula, we can clean up the variables used in the functions into a set of sequential letters, replacing \( q \) with \( l \). The final formula looks like:

\[
\forall z \neg (f_i(f_i(f_j(z))) = z) \lor (f_i(f_j(z)) = f_l(z)), \ 1 \leq i, j, k, l \leq 5
\]

The middle of this formula perfectly matches the two orientations of a triangle shown in figure 1 of lemma 6.1, and lemma 6.1 states that we need only those two orientations if we iterate through all nodes. We can also easily visualize exactly how the portions of the formula work; \( f_i(f_i(f_j(z))) = z \) asks whether upon traveling across three edges we end back where we started, while \( f_i(f_j(z)) = f_i(z) \) asks if there is a path of length two from node \( z \) to some destination that can also be reached by a path of length one. The formula \( (f_i(f_i(f_j(z))) = z) \lor (f_i(f_j(z)) = f_i(z)) \) is guaranteed to return true for some node \( z \) if and only if a triangle is in the graph regardless of the graph’s orientation, and the not statement surrounding that formula ensures that the full formula returns true if and only if there are no triangles in the graph on which it is run.

The process of running the new algorithm derived in theorem 6.1 consists of iterating through all nodes and putting them through certain specific tests. Having selected a single node, we check the functions \( f_j(z) \) and \( f_l(z) \) associated with it and what node they return, then do the same process to the node that was returned by \( f_j(z) \) to find the output of nested functions \( f_i(f_j(z)) \). We check if that output is the same as \( f_i(z) \), and repeat the node investigation process on the output of \( f_i(f_j(z)) \) to find the value of \( f_k(f_i(f_j(z))) \) and check if it returns the same node we started with. If either of our checks were correct, this particular iteration of the formula returns true, meaning the formula as a whole returns false and we conclude that the graph is not triangle free. If we iterate through every node and find our formula always returned true, the graph is triangle free. Because for each node we need to check all values of \( i, j, k, \) and \( l \), and the planarity of our investigated graph ensures that there are no more than five functions, there are at most
625 specific checks for each node. However this is clearly a constant value, and the formula takes constant time when checking a single node and $O(n)$ time overall since all nodes will be checked.

We have therefore demonstrated how to run the triangle free problem, normally an $O(n^3)$ problem, in linear time over a planar graph. We could do the same thing on a graph belonging to any class of graphs with bounded expansion, although we would no longer be checking at most five variations per function but instead $2\n泳$ variations and doing $16\n泳^4$ specific checks per nodes.

6.2 K-Cycle on Planar Graphs

The triangle free problem is in fact an iteration of a much more difficult problem known as the k-cycle problem. The k-cycle problem consists of finding whether or not a graph contains a cycle of length k. The triangle free problem would be termed the 3-cycle problem.

**Definition 6.1:** A cycle is a path that never visits the same node twice, with the exception of the single node at which it both starts and finishes.

The triangle free problem was represented by the first-order formula $\forall x,y,z \neg(E(x, z) \land E(z, y) \land E(x, y))$. The 4-cycle problem would be presented by the formula $\forall x,y,z,w \neg(E(x, y) \land E(y, z) \land E(z, w) \land E(w, x))$, and it should be intuitively clear that all other k-cycle problems with a greater k can be derived from previous formulas through the act of changing one edge to point to a new node, adding an additional edge from that node to the node the edge used to point to, and adding a new quantifier to iterate through the additional node. Therefore a k-cycle problem involves k edges and k quantifiers, and has a brute force solution with time complexity $O(n^k)$.

Rather than demonstrating how to run a single iteration of the k-cycle problem in linear time over a graph belonging to a class with bounded expansion, with planar graphs used as a key example, we will demonstrate a general method that applies to every version of the k-cycle problem and which highlights the ultimate utility of transitive fraternal augmentations. Section 6.1 has already summarized the related proofs and demonstrated the capacity to create a directed implicit representation with oriented edges embodied by functions for planar graphs or any other class of graph with bounded expansion, and thus we can move directly to the task of reducing the number of quantifiers in any iteration of the k-cycle problem to a single quantifier.

**Theorem 6.2:** It is possible to derive a first-order logic algorithm to solve any k-cycle problem that has only one quantifier and takes only one variable as an input for all k greater than or equal to three.

**Proof:** The following is a proof by induction on k that any k-cycle problem can be expressed in first-order logic with a single variable and quantifier.

Let us define the natural state of the k-cycle problem as $\forall x_1,x_2,x_3\ldots x_\text{k-1},x_\text{k} \neg(E(x_1, x_2) \land E(x_2, x_3) \land \ldots \land E(x_\text{k-1}, x_\text{k}) \land E(x_\text{k}, x_1))$. This converts to $\forall x_1,x_2,x_3\ldots x_\text{k-1},x_\text{k} \neg(f_1(x_1) = x_2 \lor f_2(x_2) = x_1) \land (f_1(x_2) = x_3 \lor f_3(x_3) = x_2) \land \ldots \land (f_\text{m}(x_\text{k}) = x_1 \lor f_\text{m}(x_1) = x_\text{k}))$ when dealing with the directed implicit representation discussed in section 6.1.
Base Case: The 3-cycle problem also answers to the name of the triangle free problem. Theorem 6.1 shows that the formula $\forall z \neg((f_i(f_j(f_k(z)))) = z) \lor (f_i(f_j(z)) = f_l(z))$, $1 \leq i, j, k, l \leq 5$ is a solution to that problem, and said formula takes only one variable as an input.

Inductive Hypothesis: Assume that there exists a formula with a single quantifier taking a single variable as input that solves the k-cycle problem for all k greater than or equal to three and less than n.

Inductive Step: It is possible to reduce this n-cycle problem to the (n-1)-cycle problem via the use of a transitive fraternal augmentation. A cycle is also a path by definition, and theorem 4.2 shows that all paths of length greater than three have their length reduced by one after a transitive fraternal augmentation. Lemma 4.1 also establishes that this happens because a set of three edges, regardless or orientation, induce the creation of a shortcut edge between two nodes. Select a section of the formula that represents four sequential nodes, such as $(f_i(x_1) = x_2 \lor f_k(x_2) = x_3 \lor f_m(x_3) = x_4) \land f_m(x_3) = x_4 \lor f_n(x_4) = x_3)$. This section represents every orientation of edges connecting node $x_1$ to node $x_4$ and consists of four pairs of $or$ statements separated by three and $and$ statements. It can therefore be replaced by a series of formulas that represent every combination of orientations and come in the form of eight sets of three $and$ statements separated by seven $or$ statements. For example, one such statement would be $f_i(x_1) = x_2 \land f_k(x_2) = x_3 \land f_m(x_3) = x_4$, while another could be $f_j(x_2) = x_1 \land f_k(x_2) = x_3 \land f_m(x_3) = x_4 \lor f_n(x_4) = x_3$. Alter each of these statements to represent the state of the graph after a transitive fraternal augmentation such that one variable is removed. $f_i(x_1) = x_2 \land f_k(x_2) = x_3 \land f_m(x_3) = x_4$ would become $(P_{ikh}(x_1) \land f_h(x_1) = x_3 \land f_m(x_3) = x_4) \lor (f_i(x_1) = x_2 \land P_{ikh}(x_1) \land f_h(x_2) = x_4)$, where, as mentioned in section 5, statements such as $P_{ikh}(x_1)$ return true if and only if edges $f_i$ and $f_k$ would induce transitive edge $f_h$, meaning when $f_h(f_i(x_1)) = f_6(x_1)$ over the augmented graph. The statement $f_j(x_2) = x_1 \land f_k(x_2) = x_3 \land f_m(x_4) = x_3$ would transform based on the two possible fraternal edges in a method covered by [5]. In both examples, and indeed in every actual case, we have removed one variable from the equation by using a transitive fraternal augmentation to essentially skip over two edges that induce a transitive or fraternal edge. The resulting formula contains one less variable and is now asking if the augmented graph contains a (n-1)-cycle. In fact, were we to ignore the predicates it would be identical to the formula provided for a (n-1)-cycle. It is worth noting that one could also progress at a faster rate by selecting mutually distinct sets of four sequential nodes and noting the transitive fraternal augmentation-induced transformation of all of them, moving the problem from the n-cycle to the (n - ceiling(n/3))-cycle.

However, there is one caveat to this result. A cycle can never return to one of its nodes more than once, and by removing a node from the formula we have left ourselves open to the possibility of mistaking a mere path of length n that returns to the original node for a cycle of length n. We can solve this problem by appending the predicate $A(x_{all})$ to the formula. This predicate, which does not appear in the literature, returns true if and only if all the nodes visited while the algorithm was run were distinct. Since the presence of predicates such as $P_{ikh}(x_1)$ means that all nodes in the original cycle will be visited at some point, it will be
possible to incorporate this predicate into the formula and receive an accurate result.

Therefore it is possible to reduce the formula $\forall x_1, x_2, x_3, \ldots x_n, x_1 \sim (E(x_1, x_2) \land E(x_2, x_3) \land \ldots \land E(x_n, x_1))$ so that it uses one less edge and runs a quantifier on one less variable while retaining the correctness of the formula by performing a transitive fraternal augmentation. Such a function is identical to the (n-1)-cycle, which we know to have a single variable version via the inductive hypothesis.

Let us now explain how exactly the theorem and proof above can be used to run a k-cycle problem on a class of graphs with bounded expansion. We have shown in section 4.3 that it takes linear time to run a transitive fraternal augmentation on a class of graphs with bounded expansion. Performing k-3 transitive fraternal augmentations ensures the reduction of the problem to the triangle free problem, now with the additional predicate $A(x_{all})$. Comparing nodes to ensure there are no duplicates other than the node actually iterated through by the quantifier takes constant time as it can be done by comparing the most recently visited node to a set of all previously visited nodes and adding it to the set if the node was not present. Therefore we can otherwise abide by the instructions for running the triangle free problem in section 6.1. However, there has been one significant change. We will no longer be checking at most five variations for each function, as is the case for planar graphs, or 2f(0) variations in the general case. As we have shown in section 4.4, after a transitive fraternal augmentation a graph still belongs to a class with bounded expansion, but the f(r) for that class is different. We must now investigate at most 2f'(0) edges, where f'(0) corresponds to the bounded expansion formula for the original class of graphs after it has undergone k-3 transitive fraternal augmentations. As an example, running the 4-cycle problem over a planar graph requires one transitive fraternal augmentation. Theorem 4.7 shows that the f'(0) of the new graph’s class would be bounded by f(0) + f(0)^2 + (2f(0)^4 + 2f(0)^3 - \frac{1}{2}f(0)^2 - \frac{1}{2}f(0))f(1) for the original class of graphs. In the case of a planar graph, both f(0) and f(1) are equal to three so we end up having to investigate at most 1284 variations for each edge, reflecting a massive increase in the potential maximum in-degree of the graph. Planar graphs do have a small advantage here over other classes of graphs with bounded expansion, and reexamining theorem 4.7 and tracking down a reference within to degeneracy’s influence on coloration allows the value to be recalculated as 942, which is still very sizable when one considers it is the number of variations per edge and the checks per node will be even greater. Regardless, the new graph class would also no longer be minor-closed. Therefore, calculating the f'(0) for the class of graphs corresponding to a planar graph that underwent more than one transitive fraternal augmentation would require calculating the f'(1) of the new class, necessitating the use of the polynomial reached at the end of section 4.4.

We have shown how the techniques explained in this thesis can be used to discover whether or not a graph belonging to a class with bounded expansion has a cycle of size k. Similar or nigh-identical approaches could be taken to solve other problems expressible in first-order logic. However, it is possible to run every first-order problem in linear time over a graph belonging to a class with bounded expansion [6] and to use the same techniques every time. We have shown that the creation of a directed implicit representation that uses functions to represent orientation is generalizable for any graph belonging to a class with a known bounded expansion function in section 6.1. Once this
has been done, the first-order logic problem can be subjected to the process known as quantifier elimination, as discussed in section 5.2, which relies on the use of transitive fraternal augmentations. No further changes to deal with specific attributes of the problem will be needed, although they will remain possible as a method of simplifying the process. The alterations to the formula performed in the process of quantifier elimination proceed according to a regimented series of steps that can be applied consistently to any input [5].

7 Applications
7.1 Gaifman Graphs

Although we stated the importance in the field of computer science of discovering faster algorithms, the utility of this project may still remain unclear. The time has come to demonstrate exactly when and how our results can be put to efficient, effective use.

Databases are the bedrock of modern computing. Almost every industry uses databases for some vital function. Relational databases are the most common type of database. They consist of a domain of individual items and relations, sets that contain items that share some established relationship. A Gaifman graph is a direct adaptation of a relational database into a graph and conversion between the two forms is a completely elementary process. Every node in the graph corresponds to one item in the database’s domain, and there are edges from any node corresponding to an item in a relation to every other node corresponding to an item located in the same relation. Using this abstraction, every statement we have proven for graphs will also hold true for databases.

Imagine that there exists an arbitrarily large database. This database is under constraints such that the Gaifman graphs corresponding to any iteration of that database created by the input, removal, and alteration of entries belong to a class with bounded expansion. These constraints could be as simple as a maximum number of allowable relations per item, or maximum total relations and items per relation, or any one of a number of genuine considerations. In such a case any complex operation that can be phrased in first-order logic can be performed using the techniques explored throughout this thesis and achieve an explicitly faster time complexity.

There is a caveat to this finding. As we have seen, the total time complexity is proportional only to the size of the graph. Since most complicated operations involve polynomial time algorithms, and these findings work equally well on problems for which the best natural solution is in time complexity $O(n^3)$ as $O(n^{50})$, one would assume this means these techniques always surpass the normal methods. However, all of the complex preprocessing necessary to achieve the linear time performance actually causes a large amount of additional operations to occur as well, which for some graphs may mean that these techniques have a net slowdown effect on the time it takes to run the algorithm. We can clearly see this in the discussion of increased in-degree after a transitive fraternal augmentation in section 6.2. Yet even this is not a significant setback, as this amount is constant as the graph’s size increases, and past a certain point these techniques become objectively, definitively faster than the polynomial time originals. In addition, in recent years there has been an increased focus on so-called ‘big data,’ quantities of information that involve discrete entries numbering in the billions or greater. For such a database the
fixed preprocessing time becomes insignificant compared to factors arising purely from the size of the database, making these techniques even more useful.

8 Conclusion

This paper has led us on a journey from the basic definition of bounded expansion to a dissection of its interactions with complementary ideas in graph theory and formal logic and finally to an explicit outline of how it can be used to solve real-world problems at faster speeds. In doing so we have defined and combined those concepts into one whole that explicitly outlines the utility of bounded expansion. Although we were unable to include every detail within this paper such that it and it alone could serve as a guide to the use of bounded expansion for running first-order algorithms in linear time, we have identified the best available sources one could use to get the full picture.

The linear time solutions with their potential to speed up difficult tasks evolved organically out of a combination of diverse mathematical concepts. We have also shown that they come at a cost, namely that the fixed but large preprocessing time means that they are only faster than normal algorithms when the size of the data passes some threshold. This limits their true utility and seems to consign them to a mere mathematical curiosity. Yet the benefits of implementation remain, due in part to an increased predilection for extremely sizable datasets and the extraordinarily slow natural time complexity of certain useful algorithms. Already there are attempts to extend this finding to the nowhere-dense classes of graphs, which include classes with bounded expansion and more besides. Perhaps soon someone will find a better method of limiting in-degree and $\nabla_r(G)$ after a transitive fraternal augmentation as well, and either discovery could mean that this work’s implementation becomes an absolute necessity rather than a limited means of advancement.

Bounded expansion has only recently begun to be explored. Its inclusion of concepts from various combinatorics subdisciplines and the scattering of necessary proofs in various locations make it naturally difficult to comprehend. It touches on so many different concepts that studying it becomes valuable in and of itself, forcing you to learn new fields. Ultimately its true value exists on both a conceptual and implementable level, demonstrating how divergent concepts can be combined and offering the potential for better, faster code.
References


