Searching for and Classifying the Finite Set of Minor-Minimal Non-Apex Graphs

Mike Pierce, California State University, Chico
Supervised by Dr. Thomas Mattman

June 17, 2014

Abstract

We say that a graph is non-apex if the removal of any vertex results in a non-planar graph. We say that a graph is minor-minimal non-apex (MMNA) if it is non-apex but all of its proper minors are apex. As a consequence of Robertson and Seymour's Graph Minor Theorem [1], we know that the set of MMNA graphs is finite. In this paper we present an argument for the uniqueness of graph simplification. We show that there are no MMNA graphs on 14 or fewer edges. We develop algorithms to do a brute-force search for MMNA graphs on graph order and size, finding all MMNA graphs with 10 or fewer vertices and with 21 or fewer edges. We also develop algorithms to construct the ΔY family of a given graph and find any new MMNA graphs in that family. In total, we have found 157 examples of MMNA graphs.

Contents

1 Introduction

The research presented in this paper looks at the apex property of graphs in view of a recent result by Neil Robertson and P. D. Seymour regarding graph minors [1]. A graph is apex if it has a vertex that can be removed to make a planar graph. According to Robertson and Seymour's Graph Minor Theorem, there must exists some finite set of graphs, the minor-minimal non-apex (MMNA) graphs, that can be used to completely characterize the apex graph property. The purpose of this paper is to attempt to find and classify the members of this finite set of MMNA graphs.

In Section 2 of this paper we will first define some common graph theory terminology, then move on to terminology and background more specific to this research, and then give an overview of the work that has been done towards characterizing the set of MMNA graphs prior to the results presented in this paper.

Section 3 presents original results towards classifying the set of MMNA graphs. We define the simplification of a graph and show that it is unique. We prove that there are no MMNA graphs with 14 or fewer edges. Algorithms are presented that allow us to quickly determine if a graph is MMNA, and a procedure using these methods to do a brute-force search on graph size and order is outlined. Algorithms to construct the ΔY-family of a given graph are also presented and used to construct the ΔY-families of our known MMNA graphs in search of new MMNA graphs.

In Section 4 we discuss open questions and topics of interest with regards to future efforts to classify the set of MMNA graphs.

Also presented in this paper in Section 5 are the algorithms, written primarily in Mathematica, that are discussed in Section 3.

2 Definitions & Background

General Graph Terminology

We define a **graph** G to consist of a finite set of **vertices** denoted as $\mathrm{V}(G)$ and a finite set of **edges** denoted as $\mathrm{E}(G)$, where $\mathrm{E}(G) \subset \big(\mathrm{V}(G) \times \mathrm{V}(G)\big)$. Graph edges are **undirected**, so an edge on vertices a and b may be denoted as (a, b) or equivalently as (b, a). We will refer to the count of the vertices of G as the **order** of G and will refer to the count of the edges of G as the **size** of G.

We say that a graph is **loopless** if no edge is defined on a single vertex ($\forall a \in \mathrm{V}(G)$, $(a, a) \notin \mathrm{E}(G)$). We say that a graph has no **double-edges** if no two vertices have more than a single edge between them (the elements of $\mathrm{E}(G)$ are distinct). Graphs that are both loopless and have no double-edges are called **simple** graphs (also called **strict** graphs). For the purposes of this research we require that all graphs be simple.

We will denote a graph with a vertex removed as $G - v$. Similarly, we will denote a graph with a edge removed as $G - e$. (Later the symbols v and e will also be used to denote the order and size of a graph respectively, but this change in use will be clear from the context).

Given a graph G such that $a, b \in \mathrm{V}(G)$ and $(a, b) \in \mathrm{E}(G)$, we say that the vertices a and b are **adjacent** (or that they are **neighbors**). Given a graph G with $a, b, c \in \mathrm{V}(G)$ and $(a, b), (b, c) \in \mathrm{E}(G)$, we say that edges (a, b) and (b, c) are **incident**. We also say that vertex b is incident to both edge (a, b) and to edge (b, c). The **degree** of a vertex is the count of the edges incident to it.

A graph is **complete** if every two distinct vertices are adjacent (i.e. $\forall a, b \in \mathrm{V}(G)$, $(a, b) \in \mathrm{E}(G)$). We denote the complete graph on n vertices as K_n. A graph G is **bipartite** if there exists a partitioning of its vertices into two non-empty sets A and B such that $A \cup B = \mathrm{V}(G)$ and such that every edge of G connects a vertex in A to a vertex in B. A graph is **complete bipartite** if every vertex of A is adjacent to every vertex of B. We denote a complete bipartite graph as $K_{n,m}$, where $n = |A|$ and $m = |B|$.

We define a **path** from vertex a to vertex b to be a finite ordered set of edges P such that a is only incident to the first edge of P, b is only incident to the last edge of P, consecutive edges of P are incident to each other, and a vertex is incident to *at most* two edges of P. We will say a graph G is **connected** if there exists a path between any two vertices of G. Otherwise is a graph is **disconnected**. We will denote a disconnected graph with components $\{G_1, G_2, \ldots, G_n\}$ as $G_1 \sqcup G_2 \sqcup \cdots \sqcup G_n$.

A non-empty path P from a vertex to itself is called a **cycle** and will be referred to as an n-cycle, where $n = |P|$. A graph with no cycles is called a **tree**. The degree-1 vertices of a tree are called **leaves**. A disconnected graph with only tree components is called a **forest**.

We say that two graphs G and H are **isomorphic** if there exists a bijection $f\colon \mathrm{V}(G) \leftrightarrow \mathrm{V}(H)$ that preserves vertex adjacency (i.e. $(a, b) \in \mathrm{E}(G) \iff (f(a), f(b)) \in \mathrm{E}(H)$). That is to say that two isomorphic graphs have the exact same structure.

Research Terminology and Background

Before we introduce the theorem that is the basis for this research, we need to define a subgraph and a graph minor. Given a graph G, we can take a **subgraph** of G by deleting any non-empty collection of vertices or edges from G. The idea of a graph minor is a generalization of the idea of a subgraph that includes edge contraction. We can **contract** an edge (a,b) in a graph by deleting vertices a and b and replacing them with a new vertex c that is adjacent to all the vertices that either a or b were adjacent to. Given a graph G, we can take a **minor** of G by performing any non-empty sequence of vertex deletions, edge deletions, and edge contractions on G. If a graph is the result of a single one of these actions, we call it a **simple minor** of G.

Now that we have defined a graph minor, we can introduce Robertson and Seymour's Graph Minor Theorem.

Theorem 2.1 *Consider some property of graphs P such that every graph either does or does not have P and that is closed under taking minors. Consider the set of all graphs G such that G does* not *have property P, but such that every minor of G has property P. This set of graphs, which are called the forbidden minors of P, is guaranteed to be finite. [1]*

In this paper, we will refer to the forbidden minors of P as the **minor-minimal non-P** graphs.

It is useful to introduce some concrete graph properties to understand the implications of the above theorem. We will say that a graph is **planar** if there exists some way to embed the graph in the plane such that no edges overlap. Note that every graph must either be planar or non-planar, and that if we take a minor of a planar graph, that minor must also be planar. Then according to Theorem 2.1 there must exist some finite set of graphs, the minor-minimal non-planar (MMNP) graphs, that are non-planar, but such that every minor of each one of them is planar.

Theorem 2.2 *There are exactly two minor-minimal non-planar graphs: the complete graph K_5, and the complete bipartite graph $K_{3,3}$ (see Figure 1). [2]*

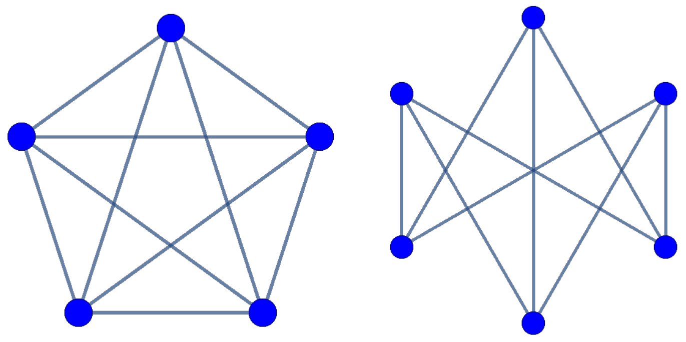


Figure 1: Graphs K_5 (left) and $K_{3,3}$ (right).

This theorem says that if we have a non-planar graph, then this graph must either be K_5 or $K_{3,3}$, or have a K_5 or $K_{3,3}$ as a minor. In this way, these two graphs completely characterize the property of graph planarity.

But of course Robertson and Seymour's Graph Minor Theorem holds for *any* graph property that meets the given criteria. We say that a graph is **intrinsically linked** if it contains a pair of disjoint linked cycles regardless of its embedding in $\mathbb{R}^3$. For example, the complete graph K_6 is intrinsically linked because it must contain at least a pair of linked 3-cycles regardless of how it is embedded in space. We note that the property of a graph being *not* intrinsically linked is closed under taking minors, so our property P in terms of Theorem 2.1 is the property of *non*-intrinsic linkedness. By Theorem 2.1 we are guaranteed that there is a finite set of minor-minimal intrinsically linked (MMIL) graphs.

THEOREM 2.3 *The MMIL graphs are the seven graphs of the Petersen Family (see Figure 2). [3]*

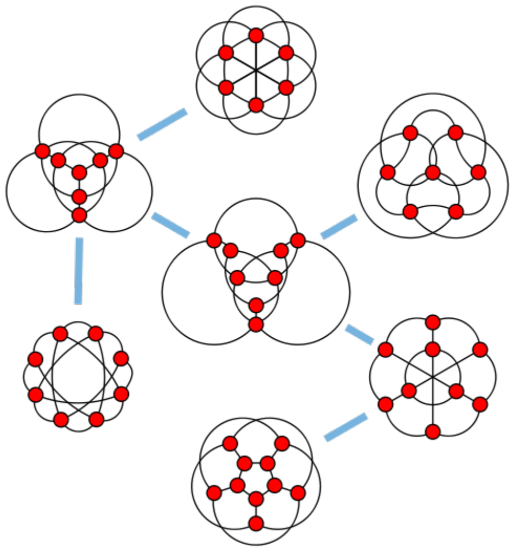


Figure 2: The Petersen Family (Source: Wikimedia Commons).

The Petersen Family is a well-known set of graphs that is characterized as being generated by ΔY and YΔ transforms on K_6.

Given a graph G with a 3-cycle $\{(a,b),(b,c),(c,a)\}$, we can perform a ΔY (also called a **triangle-wye**) transform on G by deleting the edges of the 3-cycle and replacing them with a new vertex that is adjacent to the three vertices of the 3-cycle. We refer to the set of all distinct graphs that are the result of performing a single ΔY transform on G as the ΔY-**children** of G. We refer to the set of all distinct graphs that are the result of performing any non-empty sequence of ΔY transform on G as the ΔY-**family** of G.

Given a graph G with degree-3 vertex v, we can perform a YΔ (also called a **wye-triangle**) transform on G by deleting vertex v and adding edges connecting the vertices that were adjacent to v. We define the YΔ-**parents** and YΔ-**family** of a graph similarly to ΔY-children and ΔY-family above.

The YΔY-**family** of a graph G is the set of all graphs that are the result of some sequence of ΔY or YΔ transforms on G.

Results Preceding this Research

The focus of this research are graphs that are non-apex. A graph G is **apex** if there exists some vertex v such that $G - v$ is planar. By extension, G is non-apex if $G - v$ is non-planar $\forall v \in \mathrm{V}(G)$. The apex property of graphs is closed under taking minors, so by Theorem 2.1 there must be a finite set of minor-minimal non-apex (MMNA) graphs.

Some work towards finding and classifying this set of MMNA graphs has already been done.

Theorem 2.4 *The graphs of the Petersen Family are MMNA. [4]*

Theorem 2.5 *The Jørgensen graph is MMNA (see Figure 3). [4]*

Theorem 2.6 *The MMNA graphs on eight or fewer vertices are exactly the five graphs of the Petersen Family and the Jørgensen Graph. [4]*

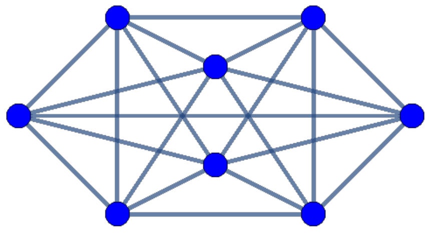


Figure 3: The Jørgensen Graph.

Results have been proven regarding the effects of ΔY and $Y\Delta$ transforms on non-apex graphs.

Lemma 2.7 *Let G be a non-apex graph with triangle t, let G' be the result of performing ΔY on t in G and let v be the new vertex added in G' as a result of performing ΔY on t. Graph G' is non-apex if and only if $G' - v$ is non-planar. [5]*

Proof: If G' is non-apex, then $G' - v$ is non-planar by the definition of an apex graph. Conversely say that $G' - v$ is non-planar. Perhaps G' is apex, so there is some vertex a of G' such that $G' - a$ is planar. Note that $a \neq v$ so either a is adjacent to v or a is not adjacent to v. If a is adjacent to v then since $G' - a$ is planar and $G - a$ is a minor of $G' - a$, graph $G - a$ is planar which contradicts G being non-apex. Else if a is not adjacent to v, then vertex a is not part of our triangle t. Then performing ΔY on $G - a$ will be the same graph as $G' - a$, so ΔY on $G - a$ is planar. Note that undoing the ΔY move on $G - a$ will preserve is planarity. Graph $G - a$ being planar contradicts G being non-apex. $\square$

So basically, unless the vertex that gets added when we perform a ΔY move on G causes the graph to become apex, then G will remain non-apex. The idea then is that since ΔY often preserves non-apexness in a graph, the ΔY families of MMNA graphs may contain more MMNA graphs.

This turns out to be true in the case of the Jørgensen graph. The ΔY family of the Jørgensen graph contains several examples of MMNA graphs.

Work has been done at classifying MMNA graphs based on connectivity. We say that a graph is n-**connected** if the removal of $n-1$ vertices results in a connected graph. By convention a complete graph K_n is $(n-1)$-connected. In general when we say a graph is n-connected, n is the maximum value for which the graph is n-connected.

Theorem 2.8 *There are exactly three 0-connected MMNA graphs, namely $K_5 \sqcup K_5$, $K_5 \sqcup K_{3,3}$, and $K_{3,3} \sqcup K_{3,3}$. [6]*

Theorem 2.9 *There are no 1-connected MMNA graphs. [6]*

We also have several examples of 2-connected MMNA graphs, but classifying *every* 2-connected MMNA graph is a difficult task.

Preceding the results presented in this paper, we have a total of 61 known MMNA graphs.

3 Theorems and Results

Graph Simplification

We note that a graph must be sufficiently dense to be MMNA. For example, a graph with a tree component cannot be MMNA because the deletion of any leaf would preserve the non-apex property of the graph. This prompts a definition of what it means for a graph to be sufficiently dense.

We say that a graph is **simplified** if it contains no vertices that are of degree 0, 1, or 2. Furthermore, given a graph G, we define the **simplification** of G to be the result of the following process on G:

(1) Delete all vertices of degree 0.

(2) Delete all vertices of degree 1 and the incident edge of each one.

(3) If there are any vertices of degree 0 or 1 remaining, go to step (1).

(4) Let v be a vertex of degree 2. Delete vertex v and its incident edges.
If the neighbors of v have no edge connecting them, add an edge to connect them.
If there are any vertices of degree 2 left, repeat this step.

(5) If there are any vertices of degree 0 or 1 remaining, go to step (1).

In this definition when a degree-2 vertex is deleted, an edge is only added between its neighbors if there was no edge between them before, so no double-edges can form in our graph. Notice that a loop could occur during simplification only if a degree-2 vertex were deleted and its neighbors happened to be the same vertex. So since double-edges don't occur, loops don't occur.

It is important to notice that the simplification of a graph will be unique under isomorphism. This fact, though, is not trivial to prove. The process of simplification above is strictly defined with the exception of step (4) where we must consider one vertex at a time. Therefore showing that simplification is unique comes down to showing that performing step (4) on a graph can only result in a single outcome independently of the order in which degree 2 vertices are chosen. For this proof we will require some new terminology and a few lemmas.

An **A-chain** of a graph is a maximal collection of 3 or more adjacent degree-2 vertices in the graph such that no vertex has a non-degree-2 neighbor. (Imagine a cycle consisting only of degree-2 vertices). A **B-chain** of a graph is a maximal collection of 2 or more adjacent degree-2 vertices in the graph such that there is exactly one non-degree-2 vertex that is a neighbor to some of the vertices in the collection. This non-degree-2 neighbor we will refer to as the **terminal vertex**. A **C-chain** of a graph is a maximal collection of 1 or more adjacent degree-2 vertices in the graph such that there are exactly two non-degree-2 vertices (terminal vertices) that are neighbors of some of the vertices in the collection.

Note that every degree-2 vertex in a graph must exist either in an A-chain, B-chain, or C-chain. If we consider some graph and some arbitrary degree-2 vertex v in our graph, we can imagine tracing two paths through our graph starting at v and headed in different directions. If in both directions we hit a vertex that is not of degree two, we must have either hit the same vertex from both directions, or distinct vertices in each direction, meaning that v was part of a B-chain or C-chain respectively. If we don't hit a non-degree-2 vertex in either direction, the v is part of an A-chain. If our path terminates in one direction but does not terminate in the other direction, then our graph is evidently not finite.

LEMMA 3.1 *Performing step (4) on an A-chain in a graph can result in only a single possible graph (under isomorphism), namely the graph where the A-chain is replaced by K_2.*

PROOF: First we note that if an A-chain exists in a graph, it must either make up the entire graph, or must be a component of a disconnected graph. Either way, suppose that our A-chain consists of n vertices. By performing step (4) on this chain, we will single out a vertex, delete it, and connect its neighbors, resulting in an A-chain of $n-1$ vertices. We continue performing step (4) until we have an A-chain of 3 vertices. Then we single out a vertex and delete it and note that its neighbors are already connected, resulting in K_2. So we see that regardless of the vertices of the A-chain that were singled out for deletion, the resulting graph is a disjoint union of K_2 and the component of the graph disjoint from the original A-chain. □

LEMMA 3.2 *Performing step (4) on a B-chain in a graph can result in only a single possible graph (under isomorphism), namely the graph where the B-chain is replaced by a single degree-*1 *vertex adjacent to the terminal vertex.*

PROOF: Consider a B-chain of a graph and its single terminal vertex v. By performing step (4) on this B-chain we single out a vertex arbitrarily, delete it, and connect its neighbors, resulting in a B-chain of one fewer vertices and with the same terminal vertex v. Much like above we can continue performing step (4) until our B-chain consists of two vertices, both neighbors to v. Performing step (4) again, we delete one of the degree-2 vertices and note that its neighbors, the other degree-2 vertex and v are already connected. So we see that regardless of the vertices of the B-chain that were singled out for deletion, the resulting graph will be the same graph we started with, only having a single degree-1 vertex connected to v in place of its B-chain. □

LEMMA 3.3 *Performing step (4) on a C-chain in a graph can result in only a single possible graph (under isomorphism), namely the graph where the C-chain is replaced by an edge between the two terminal vertices.*

PROOF: Consider a C-chain of a graph and its terminal vertices v_1 and v_2. By performing step (4) on this C-chain we single out a vertex arbitrarily, delete it, and connect its neighbors, resulting in a C-chain of one fewer vertices and with the same terminal vertices v_1 and v_2. We can continue performing step (4) until our C-chain consists of a single vertex, neighboring both v_1 and v_2. Performing step (4) again, we delete our final degree-2 vertex and if v_1 and v_2 are not neighbors, we connect them. So we see that regardless of the vertices of the C-chain that were singled out for deletion, the resulting graph will be the same graph we started with, only having had all the degree-2 vertices of the C-chain deleted and the vertices v_1 and v_2 connected. □

THEOREM 3.4 *The simplification of a graph is unique under isomorphism.*

PROOF: We note that every step of the simplification process defined above, with the exception of step (4), is strictly defined and can result in only a single possible graph. So the uniqueness of graph simplification is contingent on the uniqueness of the result of step (4).

Take some arbitrary graph G. We've already mentioned above that every degree-2 vertex of G must exist in exactly one A-chain, B-chain, or C-chain. Since G can only have a finite number of degree-2 vertices, there must be a finite number of these chains. Since no degree-2 vertex can be in more than one chain, the deletion of a vertex in a chain cannot affect the existence of other degree-2 vertices in other chains. Because of this, if we perform step (4) on G, all the chains of G will be reduced uniquely in the ways described in Lemmas 3.1, 3.2, and 3.3 regardless of the order in which we select the degree-2 vertices. Since all of these chains must be simplified in the ways described in these preceding lemmas, any new degree-2 vertices that can emerge in this process (the degree of a terminal vertex of a B-chain will lose a degree when its chain is simplified), *must* emerge in this process, and so must be selected during the performance of step (4).

So because every degree-2 vertex that can be considered in step (4) must be considered in step (4), and the degree-2 vertices in the chains of G are disjoint, and every iteration of step (4) is a step in reducing a chain as described in Lemmas 3.1, 3.2, and 3.3 and the result of these processes are unique, the end result of the process of graph simplification is also unique. □

Now that we have established a rigid definition of a simplified graph and of the simplification of a graph, we will go ahead and prove the idea that inspired the consideration of graph simplification in the first place.

THEOREM 3.5 *A MMNA graph must be a simplified graph.*

PROOF: Perhaps we have a graph G that is MMNA, but is not simplified.

Since G is MMNA, $G - v$ is apex and not planar. There must exist some vertex w of $G - v$ such that $G - v - w$ is planar. But consider $(G - v - w) + v = G - w$.

First, if the degree of v is 0, then it adds no vertices to our graph, and so $G - w$ must also be planar. But $G - w$ is a minor of G so it cannot be planar. So G cannot have a degree-0 vertex.

Next suppose the degree of v is 1. If w is the neighbor of v, then v is degree-0 in $G - w$ and, like above, we must have $G - w$ planar. Else w is not the neighbor of v. Independent of the embedding of $G - w$, we can shrink the edge incident to v such that it can have no effect on the planarity of $G - w$. So since $G - v - w$ is planar, $G - w$ must also be planar. But $G - w$ is a minor of G so it cannot be planar. In either case, G cannot have a degree-1 vertex.

Lastly suppose the degree of v is 2. If w is either neighbor of v, then v is degree-1 in $G - w$ and, like above, we must have $G - w$ planar. Else w is not the neighbor of v. Independent of the embedding of $G - w$, if we were to consider the edge that would replace v and its incident edges if either of those incident edges were to be contracted, this edge can be made to occupy the same space as v and the incident edges of v. In this way, it can have no effect on the planarity of $G - w$. So since $G - v - w$ is planar, $G - w$ must also be planar. But $G - w$ is a minor of G so it cannot be planar. In either case, G cannot have a degree-2 vertex.

So the minimum degree of graph G must be 3, and our MMNA graph must be simplified. □

Classifying MMNA Graphs by Size and Order

Before characterizing the set of MMNA graphs in terms of their size and their order, we establish a few characteristics of the set of graphs in general.

LEMMA 3.6 *Let G be a graph with minimum vertex degree δ, and order v such that $v > \delta$. Then, denoting the size of G as e, e is at least $\lceil \frac{\delta v}{2} \rceil$ and at most $\frac{v(v-1)}{2}$.*

PROOF: For the minimum value of e, we recall that by the handshaking lemma, the sum of the degrees of the vertices in G must be twice the number of edges. So since $v\delta$ is less than the sum of the actual degrees of the vertices of G, and since each of v, δ, and e are whole numbers, we have:

$$v\delta \leq \sum_{i \in V(G)} |N(i)| = 2e \quad \implies \quad v\delta \leq 2e \quad \implies \quad \left\lceil \frac{v\delta}{2} \right\rceil \leq e \tag{3.6.1}$$

So G has a minimum of $\lceil \frac{v\delta}{2} \rceil$ edges.

Now for the maximum value of e, if a graph has order v, since we are not permitting loops or double-edges, the largest size of this graph is for the complete graph K_v. The size of K_v is given by the sum:

$$\sum_{i=1}^{v-1} i = \frac{v(v-1)}{2} \tag{3.6.2}$$

So G has a maximum of $\frac{v(v-1)}{2}$ edges. □

We require that $v > \delta$ in the above lemma because a vertex in a graph has degree at most $v-1$. Since an MMNA graph must be simplified, we will only be considering graphs with a minimum vertex degree $\delta \geq 3$.

As a consequence of Lemma 3.6, we can also talk about the order of the graph given the size.

COROLLARY 3.7 *Given a graph with size e and minimum degree δ, the order of the graph is at least $\lceil \frac{1+\sqrt{8e+1}}{2} \rceil$ and at most $\lfloor \frac{2e}{\delta} \rfloor$.*

PROOF: We can rearrange the expression $e \leq \frac{v(v-1)}{2}$ in Lemma 3.6 to get the minimum of v in terms of e:

$$\begin{aligned} \frac{v(v-1)}{2} \geq e \quad &\implies \quad (v-\frac{1}{2})^2 - \frac{1}{4} \geq 2e \\ &\implies \quad v \geq \left\lceil \sqrt{\left(2e+\frac{1}{4}\right)} + \frac{1}{2} \right\rceil = \left\lceil \frac{1+\sqrt{8e+1}}{2} \right\rceil \end{aligned} \tag{3.7.1}$$

Similarly, if we look at the middle inequality of 3.6.1, we could very well solve for v instead. So if we have a graph with size e, then the order of this graph must be at most $\lfloor \frac{2e}{\delta} \rfloor$. □

Using these relations between graph order and graph size, we can talk about restrictions on the set of MMNA graphs in terms of graph order and graph size.

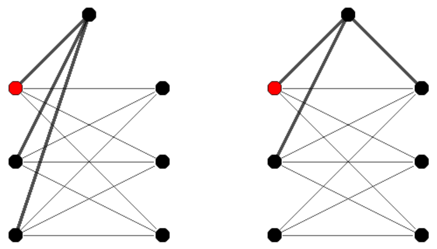
THEOREM 3.8 *No graph of size less than or equal to 14 is MMNA.*

PROOF: First we note that if a graph is MMNA, then it must be simplified, and so must have a minimum vertex degree of $\delta \geq 3$. If our graph G has size less than or equal to 10, by Corollary 3.7, the maximum order is $\lfloor \frac{2\times 10}{3} \rfloor = 6$. Then G must be a proper minor of K_6. Since K_6 is MMNA, we know that G cannot be MMNA.

SIZE 11: Take a graph G of size 11, and suppose G is MMNA. By Corollary 3.7, the maximum order of our graph is $\lfloor \frac{2\times 11}{3} \rfloor = 7$. Since G is MMNA, $G - v$ is non-planar $\forall v \in V(G)$. Note that the maximum order of $G - v$ is 6. So we ask which graphs are both non-planar and have maximum order 6? The complete graph K_5 fulfills these requirements, but then we would have $G = K_5 + v$ and is a proper minor of K_6. A graph with order 6 and a $K_{3,3}$ minor also fulfills these requirements, but since the size of $K_{3,3}$ is 9, if we were to consider adding a vertex of degree at least 3 to this graph, the result would have size 12 or more, and so it could not be G.

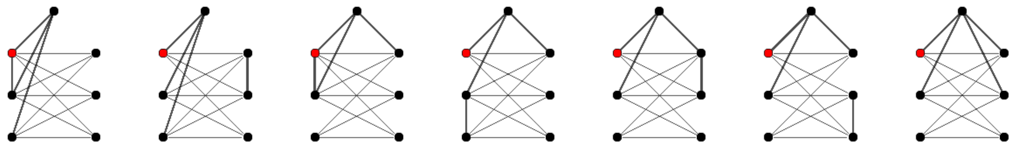
SIZE 12 OR 13: Take a graph G of size 12 or 13, and suppose G is MMNA. By Corollary 3.7, the maximum order of our graph is $\lfloor \frac{2\times 13}{3} \rfloor = 8$. Since G is MMNA, $G - v$ is non-planar $\forall v \in V(G)$. Then $G - v$ has a K_5 or $K_{3,3}$ minor. First, if $G - v$ has a K_5 minor, then $G - v$ must be K_5 since this graph has size at most 10 and the addition of a degree-3 vertex would put its size at the limit of 13. But then $G = K_5 + v$ where v is a degree-3 vertex. Then G has order 6 and is a proper minor of K_6, so G cannot be MMNA.

Second we must consider if $G - v$ has a $K_{3,3}$ minor. If G has size 12, then since the size of $K_{3,3}$ is 9, it must be that $G = K_{3,3} + v$ where v is a vertex of degree three. This leads to two possible graphs:



We note, though, that if we remove a single degree-4 vertex in each of these (indicated with red), the resultant graphs are planar, so G must be apex.

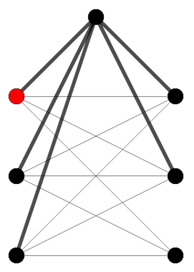
Then if G has size 13, it must either be that $G = K_{3,3} + e + v$ where v is a degree-3 vertex, or $G = K_{3,3} + v$ where v is a degree-4 vertex. In either of these cases, we can simply consider the distinct ways to add an edge to the two graphs that we made in the case of G having size 12. There are 7 such possibilities:



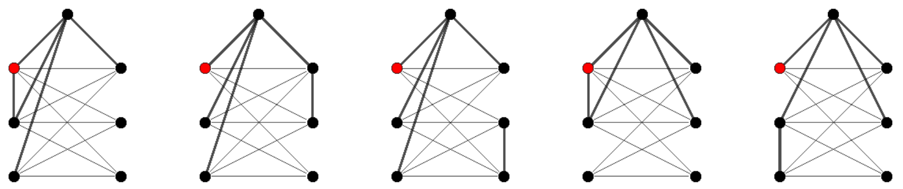
In each of these cases too, we see that there is a vertex that can be removed (indicated with red) that will result in a planar graph, so G must be apex.

SIZE 14: Take a graph G of size 14, and suppose G is MMNA. By Corollary 3.7, the maximum order of our graph is $\lfloor \frac{2\times 14}{3} \rfloor = 9$. Since G is MMNA, $G - v$ is non-planar $\forall v \in V(G)$. Then $G - v$ has a K_5 or $K_{3,3}$ minor. First, if $G - v$ has a K_5 minor, then since K_5 has size 10 and is complete it must be that $G = K_5 + v$ where v is a degree-4 vertex. There is a single possible graph that G may be, namely $K_6 - e$. But since this is a proper minor of K_6 G must be apex.

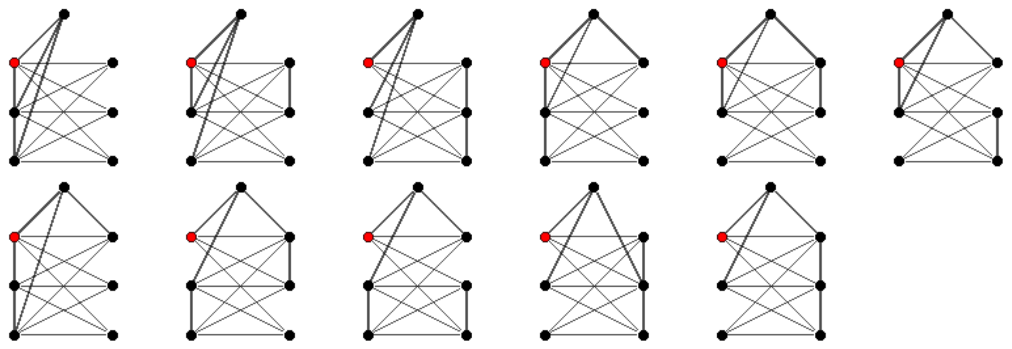
Second, if $G - v$ has a $K_{3,3}$ minor, since G has size 14 and $K_{3,3}$ has size 9, there are three possible cases we must consider for G: First, we may have $G = K_{3,3} + v$ where v is a degree-5 vertex. There is a single possible graph of this type:



Second, we may have $G = K_{3,3} + e + v$ where v is a degree-4 vertex. There are five possible graphs of this type:



Third, we may have $G = K_{3,3} + 2e + v$ where v is a degree-3 vertex. There are eleven possible graphs of this type:



In all of these cases though, there is a vertex that can be removed (indicated with red) that will result in a planar graph. □

Brute-Force Search for MMNA Graphs

For graphs of size greater than 14, searching for MMNA graphs based on size and order becomes markedly more difficult. At this point it becomes more practical to simply do a computer search for MMNA graphs.

First we need to generate all the graphs that need to be searched. For this we can use the gtools that come with the `nauty` & `Traces` graph package[7]. These tools can be used to generate all graphs of a given order with bounds on both the graph size and the minimum and maximum vertex degree. We can also use these gtools to remove all the planar graphs. I made a bash script `runner.sh` to automate this.

Next we need to convert the output of `runner.sh` to a format that Mathematica can more easily work with. For this I made the C++ script `geng2mathematica`.

Now we must have a way in Mathematica to check if a graph is MMNA. For the sake of algorithmic efficiency, I wrote the following three functions:

> `DeleteGraphDuplicates` - Deletes duplicates (under isomorphism) from a graph list.
>
> `DistinctEdges` - Given a graph, returns a maximal set of non-equivalent edges.
>
> `DistinctVertices` - Given a graph, returns a maximal set of non-equivalent vertices.

Next we need to be able to take a minor of a graph. The capability to delete vertices and edges is built into Mathematica, but we need to create a function to contract a given edge:

> `EdgeContract` - Given a graph and edge, contracts that edge in the graph.
>
> `BuildFirstMinors` - Given a graph, returns a list of all the graphs that result from removing one vertex, removing one edge, or contracting one edge.

Using the `BuildFirstMinors` function and Mathematica's built-in capability to check if a graph is planar, writing functions to check if a graph is apex and check if a graph is MMNA is straightforward.

> `ApexGraphQ` - Given a graph, returns true if that graph is apex.
>
> `MMNAGraphQ` - Given a graph, returns true if that graph is MMNA.

These can then be used to check large sets of graphs for any graphs that are MMNA. When I started work on this project, there were a total of 61 MMNA graphs known. These 61 graphs, in addition to the information from Lemma 3.6 and Theorem 3.8 are represented in the table below.

Number of MMNA Graphs by Size and Order (61 Total)

Number of Vertices (Order)

0	←	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
17	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨						
16	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨								
15	▨	▨	▨	▨	▨	▨	▨	▨	▨					4				
14	▨	▨	▨	▨	▨	▨	▨				1	1			3			
13	▨	▨	▨	▨	▨	▨			1			2	1			2		
12	▨	▨	▨	▨	1					1			2	2	1		1	
11	▨	▨	▨			2	1	1			1	1		1				
10	▨	1			2	3	7	7										
9	▨	1			1		2	1										
8	▨	2						1								▨	▨	▨
7	▨	2	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
6	▨	1	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
↓	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨

Number of Edges (Size)

Table squares in which MMNA graphs exist are indicated with blue. Squares in which we know that no MMNA graphs exists are indicated with gray. The squares that remain white are yet to be searched.

We can do a brute-force search on many of the white squares. First we generate the list of all graphs to be searched based on graph order. We generate all the graphs of order v with minimum vertex degree 3, maximum vertex degree $v-1$, and with at least 16 edges and at most $\frac{v(v-1)}{2}$ edges for $v \in \{8, 9, 10\}$.

```
./runner.sh 3 7 8 16 28
./runner.sh 3 8 9 16 36
./runner.sh 3 9 10 16 45
```

These three queries generate 2104, 83927, and 5203091 graphs respectively, for a total of 5289122 graphs to be checked in Mathematica. Of these graphs, 11 are MMNA graphs that were not previously known of.

The next sensible thing would be to do a similar search on graphs of order 11. Unfortunately, this is not feasible. Running the computation in Mathematica to search 5289122 graphs for MMNA graphs took more than a week to finish. Since there are 577076528 graphs on 11 vertices that would have to checked, continuing this line of brute-force search will not complete in a reasonable amount of time with Mathematica.

Instead we can pursue a similar brute-force search on graph size. We can generate all graphs of size e with minimum vertex degree 3, and with at least 11 vertices and at most $\lfloor \frac{2e}{\delta} \rfloor$ vertices. for $e \in \{17, 18, 19, 20, 21\}$.

```
./runner.sh 3 10 11 17 21
./runner.sh 3 11 12 18 21
./runner.sh 3 12 13 20 21
./runner.sh 3 13 14 21 21
```

These four queries result in a total of 2658795 graphs. Of these graphs, 37 were new MMNA graphs.

Number of MMNA Graphs by Size and Order (109 Total)

Number of Vertices (Order)

17																		
16																		
15													4					
14										1	1			3				
13								1			2	1			2			
12					1		1	12		1			2	2	1		1	
11						2	12	14			1	1		1				
10		1			2	3	7	7	4	5	2							
9		1			1		2	1										
8		2						1										
7		2																
6		1																
↓																		
0	←	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Number of Edges (Size)

Like before, since there are 11247888 graphs that would have to be checked on 22 vertices, continuation of this line of brute-force search is not computationally feasible with Mathematica.

Generating the ΔY-Families of MMNA Graphs

In addition to a simple brute-force search for MMNA graphs on graph size and order, we can do a more intelligent search for MMNA graphs.

From Lemma 2.7 we know that performing ΔY on a non-apex graph will preserve the non-apex property unless the removal of the vertex that gets added to the graph when we perform ΔY results in a non-planar graph. It is also true that ΔY preserves the size of the graph, so performing ΔY does not produce a graph minor. These two facts together strongly indicate that we may find new MMNA graphs in the ΔY families of the graphs that are in our list of known MMNA graphs.

We develop functions in Mathematica to generate the ΔY families of our list of known MMNA graphs and then iterate through the graphs in these families checking if they are new MMNA graphs. First we have two functions:

> `TriangleList` - Returns a list of all triangles in a graph.
>
> `TriangleWye` - Given a graph and a triangle in that graph, returns the result of performing ΔY on that triangle in that graph.

Then we have the functions that do the work of producing the ΔY family of a graph and of iterating over those families searching for new MMNA graphs:

> `TriangleWyeChildren` - Given a graph, return the distinct ΔY children of that graph.
>
> `TriangleWyeFamily` - Given a graph, return the ΔY family of that graph.
>
> `TriangleWyeMMNAGraphMine` - Given a list of graphs, returns a list of any new MMNA graphs that exist in the ΔY families of the graphs in the list.

Running `TriangleWyeMMNAGraphMine` on our list of 109 known MMNA graphs results in 48 new MMNA graphs. This brings the total number of known MMNA graphs to 157.

Number of MMNA Graphs by Size and Order (157 Total)

Number of Vertices (Order)

17	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	⬜	⬜	⬜	⬜	⬜	⬜
16	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	⬜	⬜	⬜	🟧 **1**	🟧 **6**	⬜	⬜	⬜
15	▨	▨	▨	▨	▨	▨	▨	▨	▨	⬜	⬜	⬜	⬜	🟧 **4**	🟧 **8**	⬜	⬜	⬜
14	▨	▨	▨	▨	▨	▨	▨	▨	⬜	⬜	🟧 **1**	🟧 **2**	🟧 **2**	🟧 **1**	🟧 **7**	🟧 **1**	⬜	⬜
13	▨	▨	▨	▨	▨	▨	▨	▨	🟧 **1**	🟧 **3**	⬜	🟧 **3**	🟧 **3**	🟧 **2**	🟧 **1**	🟧 **2**	🟧 **1**	⬜
12	▨	▨	▨	▨	🟦 **1**	▨	🟦 **1**	🟦 **12**	🟧 **1**	🟧 **6**	⬜	🟧 **2**	🟧 **2**	🟧 **2**	🟧 **1**	⬜	🟧 **1**	⬜
11	▨	▨	▨	▨	▨	🟦 **2**	🟦 **12**	🟦 **14**	🟧 **2**	🟧 **4**	🟧 **1**	🟧 **1**	⬜	🟧 **1**	⬜	⬜	⬜	⬜
10	▨	🟦 **1**	▨	▨	🟦 **2**	🟦 **3**	🟦 **7**	🟦 **7**	🟦 **4**	🟦 **5**	🟦 **2**	▨	▨	▨	▨	▨	▨	▨
9	▨	🟦 **1**	▨	▨	🟦 **1**	▨	🟦 **2**	🟦 **1**	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
8	▨	🟦 **2**	▨	▨	▨	▨	▨	🟦 **1**	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
7	▨	🟦 **2**	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
6	▨	🟦 **1**	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
↓	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
0	←	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Number of Edges (Size)

Figure 4: The 157 known MMNA graphs

Note that in the above table, orange squares indicate that there exist MMNA graphs of that size and order but that there may be more, blue squares indicate that all MMNA graphs of that size and order have been found, gray squares indicate that there are no MMNA graphs of that size and order, and white squares indicate that no MMNA graphs of that size and order have yet been found.

4 Questions for Further Research

While the results in this paper do well to find more examples of MMNA graphs, nothing is really said regarding how close we are getting to finding the *entire* set of MMNA graphs. Robertson and Seymour's Graph Minor Theorem [1] tells us that this set of MMNA graphs must be finite, but gives us no indication of an upper bound on its size.

QUESTION 4.1 *What is a finite upper bound on the number of MMNA graphs?*

QUESTION 4.2 *How many MMNA graphs are there?*

The hope during this research was that by doing a computer search for MMNA graphs, we would find a significant portion of the entire set. Then using these graphs that were found, notice some pattern inherent to MMNA graphs to help classify the entire set of MMNA graphs. Noting the results presented in Figure 4, though, there exist many MMNA graphs beyond the limit of feasibly doing a brute-force search with Mathematica.

QUESTION 4.3 *How much further can we expand the limit of feasibly doing a brute-force search by using a more efficient computing technology?*

There is also something to be said about the lack of MMNA graphs on 16 or 17 edges. Referencing Figure 4, we have the Petersen family as our only MMNA graphs on 15 edges, then there is a evident gap between the Petersen family and the rest of the known MMNA graphs on 18 and greater edges. We have done a brute-force check to confirm that there are no MMNA graphs on 16 or 17 edges, but we have no intuitive idea as to why this is the case.

QUESTION 4.4 *Why are there no MMNA graphs on* 16 *or* 17 *edges?*

Having an answer to this question may give us some insight into the structure and content of the entire set of MMNA graphs.

One idea that lends to answering the above question (in that it partially explains why the Petersen family is isolated from the greater collection of MMNA graphs) is that the Petersen family is closed under ΔY and YΔ transforms. That is, the Petersen family is a complete YΔY family. The content of this paper did some work to explore how MMNA graphs can be related in terms of ΔY and YΔ transforms, but there is a lot more that can be done. For example, when calculating the ΔY families of each of our known MMNA graphs, we only calculating the successive ΔY children of each starting graph and did not search for any graphs that have one of our known MMNA graphs as a child.

While there is no evidence to suggest that a YΔ transform has any tendency to preserve the non-apex property of a graph, there still may be something to exploring the YΔ families of our list of known MMNA graphs. Furthermore, though it would be more computationally extensive, we could explore the full YΔY families of our list of known MMNA graphs, splitting them up into equivalence classes.

QUESTION 4.5 *How many* YΔY *families exist in our list of known MMNA graphs?*

QUESTION 4.6 *How many* YΔY *families exist in the entire set of MMNA graphs?*

The hope is that we will see some structure within the YΔY equivalence classes of the known MMNA graphs to get some insight into how we can answer the second of these questions.

5 Source Code

runner.sh

This script relies on the gtools `geng`, `planarg`, and `showg` distributed in the `nauty` & `Traces` software package [7].

```bash
#!/bin/bash

###############################################################################
# runner.sh
#
# Generates a file that Mathematica can read in easily
# for the purpose of finding MMNA graphs.
#
# Run it as follows:
#   ./runner.sh MinDegree MaxDegree Vertices MinEdges MaxEdges
#     MinDegree - the minimum degree of a vertex
#     MaxDegree - the maximum degree of a vertex (setting to 100 is fine)
#     Vertices - the number of vertices
#     MinEdges - the minimum number of edges
#     MaxEdges - the maximum number of edges (can be same as MinEdges)
###############################################################################

if [ "$5" = "" ]
then
   printf "USAGE:\nrunner.sh d D n e E
   d - the minimum degree of a vertex
   D - the maximum degree of a vertex (setting to 100 is fine)
   n - the number of vertices
   e - the minimum number of edges
   E - the maximum number of edges (can be same as MinEdges)\n"
   exit 1
fi

if [ "$4" = "$5" ]
then
   NAME="${3}cd${1}-p(${4}).txt"
else
   NAME="${3}cd${1}-p(${4}-${5}).txt"
fi

geng -c -d${1} -D${2} ${3} ${4}:${5} | planarg -v | showg -e | ./geng2mathematica > ./${NAME}

exit 0
```

geng2mathematica

```cpp
///////////////////////////////////////////////////////////////////////////////
// geng2mathematica.cpp
//
// Converts text generated by
//   geng -c -d# -D# n min:max | planarg -v | showg -e
// into a list of ordered pairs (edges) that can be read by Mathematica.
//
// This also increments each numeral by 1,
// so vertex enumeration starts at 1.
///////////////////////////////////////////////////////////////////////////////

#include <iostream>
#include <string.h>
#include <limits>
using namespace std;

int main() {

    string graph_header = "";  // If it equals "Graph", read an edgelist.
    unsigned long int count = 0; // Number of edges for an edgelist.
    unsigned long int v1 = 0;  // First vertex of edge.
    unsigned long int v2 = 0;  // Second vertex of edge.
    bool first_run = true;     // First Run.

    while(cin.peek() != -1) {
        if(isspace(cin.peek())) {
            cin.ignore();
        }
        else if(isalpha(cin.peek())) {
            cin >> graph_header;
            if(graph_header == "Graph") {
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                cin >> count;
                cin >> count;
                if(!first_run) cout << " }" << endl;
                first_run = false;
                cout << "{ ";
                for(unsigned int i = 0; i < count; ++i) {
                    cin >> v1;
                    cin >> v2;
                    v1++;
                    v2++;
                    cout << "{" << v1 << "," << v2 << "}";
                    if (i < count - 1) {cout << ",";}
                }
            }
        }
        else{
            cerr << "The input was not formatted as expected." << endl;
            return 1;
        }
    }
    cout << " }" << endl;
    return 0;
}
```

MMNAGraphQ and Related Mathematica Functions

These functions were written and executed in Mathematica 9.0.0.0.

```
(* Given a list of graphs GList, remove any graphs
   that are duplicates under isomorphism. *)
DeleteGraphDuplicates[GList_List] := Module[{},
  Return[
    DeleteDuplicates[GList, IsomorphicGraphQ[#1, #2] &]
  ];
];

(* Returns a list of the distinct (non-equivalent)
   edges in graph G, where two edges are equivalent
   if the graphs that are the result of the removal
   of those edges are isomorphic. *)
DistinctEdges[G_Graph] := Module[
  {edges = EdgeList[G]},
  Return[
    DeleteDuplicates[
      edges,
      IsomorphicGraphQ[EdgeDelete[G, #1], EdgeDelete[G, #2]] &
    ]
  ];
];

(* Returns a list of the distinct (non-equivalent)
   vertices in graph G, where two vertices are equivalent
   if the graphs that are the result of the removal
   of those vertices are isomorphic. *)
DistinctVertices[G_Graph] := Module[
  {vertices = VertexList[G]},
  Return[
    DeleteDuplicates[
      vertices,
      IsomorphicGraphQ[VertexDelete[G, #1], VertexDelete[G, #2]] &
    ]
  ];
];
```

```
(* Given a graph G and an edge E,
   returns the result of contracting E in G. *)
EdgeContract[G_Graph, E_UndirectedEdge] := Module[
  {v1 = Min[{E[[1]], E[[2]]}], v2 = Max[{E[[1]], E[[2]]}], graph, edges},

  (* Delete the edge to be contracted. *)
  graph = EdgeDelete[G, E];

  (* Get edges of v2 that need to be transfered. *)
  edges = UndirectedEdge[Min[v1, #], Max[v1, #]] & /@ AdjacencyList[graph, v2]

  (* Remove any of those edges that are already on v1. *)
  edges = Complement[edges, EdgeList[graph, UndirectedEdge[v1, _]]];
  (* Add edges to v1. *)
  graph = EdgeAdd[graph, edges];
  (* Delete v2. *)
  graph = VertexDelete[graph, v2];
  Return[graph];
];

(* Returns the first minors of graph G by
   taking the union of the results of removing all
   non-equivalent vertices and of removing
   and contracting all non-equivalent edges. *)
BuildFirstMinors[G_Graph] := Module[
{vertices, edges, vertexremovals, edgeremovals, edgecontractions},

  vertices = DistinctVertices[G];
  edges = DistinctEdges[G];

  vertexremovals = Table[VertexDelete[G, i], {i, vertices}];
  edgeremovals = Table[EdgeDelete[G, i], {i, edges}];
  edgecontractions = Table[EdgeContract[G, i], {i, edges}];

  Return[
    Union[edgeremovals,
      DeleteGraphDuplicates[Union[vertexremovals, edgecontractions]]
    ]
  ];
];

(* Take a graph G and each graph that is the result
   of removing a vertex from G, and map PlanarGraphQ over them all.
   If any member of this set is True, then G is apex. *)
ApexGraphQ[G_Graph] := Module[{},
  Return[
    MemberQ[
      PlanarGraphQ /@ Union[{G}, Table[VertexDelete[G, i], {i,VertexList[G]}]]
    , True]
  ];
];
```

```
(* Returns True if graph G is not apex
   and all the first minors of G are apex. *)
MMNAGraphQ[G_Graph] := Module[{},
  Return[
    !(ApexGraphQ[G] || MemberQ[ApexGraphQ /@ BuildFirstMinors[G], False])
  ];
];
```

TriangleWyeMMNAGraphMine and Related Mathematica Functions

These functions were written and executed in Mathematica 9.0.0.0.

```
(* Returns a list of all 3-cycles(triangles) in G.
   The list is of the form {{1,2,3},{3,4,6}, ...}
   where the numerals are the vertices of the triangles in G. *)
TriangleList[G_Graph] := Module[{},
  Return[
    Union[
      Sort /@ Flatten[Table[
        List[e[[1]], e[[2]], #] & /@
        Intersection[AdjacencyList[G, e[[1]]], AdjacencyList[G, e[[2]]]]
      , {e, EdgeList[G]}], 1]
    ]
  ];
];

(* Performs a triangle-wye move on triangle Triangle in G. *)
TriangleWye[G_Graph, Triangle_List] := Module[
  {v, deledges, addedges},
  If[Length[Triangle] != 3, Return[$Failed];];

  (* The vertex to be added. *)
  v = Min[Complement[Range[VertexCount[G] + 1],VertexList[G]]];

  (* The edges of Triangle to be deleted. *)
  deledges = Intersection[Flatten[Table[
    {UndirectedEdge[e[[1]], e[[2]]], UndirectedEdge[e[[2]], e[[1]]]}
  , {e, Subsets[Triangle, {2}]}], 1], EdgeList[G]];

  (* The edges of our Y to be added. *)
  addedges = UndirectedEdge[Min[v, #], Max[v, #]] & /@ Triangle;

  Return[EdgeAdd[VertexAdd[EdgeDelete[G, deledges], v], addedges]];
];
```

```
(* Returns the triangle-wye children of G
   that are distinct under isomorphism. *)
TriangleWyeChildren[G_Graph] := Module[{},
  Return[
    DeleteGraphDuplicates[Table[TriangleWye[G, t], {t, TriangleList[G]}]]
  ];
];

(* Returns the entire triangle-wye family of G.
   Returns a nested list that keeps track of the family heritage.
   Just use Flatten[] to get a normal list. *)
TriangleWyeFamily[G_Graph] := Module[{},
  Return[
    List[#, TriangleWyeFamily[#]] & /@ TriangleWyeChildren[G]
  ];
];

(* Takes a list of graphs {G} and creates the triangle-wye family for all of them.
   Returns any new MMNA graphs (graphs not already in {G}) found in these families. *)
TriangleWyeMMNAGraphMine[GraphList_List] := Module[
  {results = {}},

  (* Iterate through GraphList building the triangle-wye family for each
     and append the found MMNA graphs to results. *)
  Do[
    AppendTo[results,
      DeleteCases[
        DeleteGraphDuplicates[Flatten[TriangleWyeFamily[i]]], G_ /; \[Not] MMNAGraphQ[G]
      ]
    ];
  , {i, GraphList}];

  results = DeleteGraphDuplicates[Flatten[results]];
  Return[
    (* Remove MMNA graphs that are already in GraphList. *)
    DeleteCases[results, G_ /; MemberQ[GraphList, H_ /; IsomorphicGraphQ[G, H]]]
  ];
];
```

References

[1] N. Robertson and P. D. Seymour, "Graph Minors. XX. Wagner's Conjecture," *Journal of Combinatorial Theory, Series B*, vol. 92, no. 2, pp. 325–357, 2004. `http://www.sciencedirect.com/science/article/pii/S0095895604000784`.

[2] K. Wagner, "Über eine Eigenschaft der ebenen Komplexe," *Mathematische Annalen*, vol. 114, no. 1, pp. 570–590, 1937.

[3] N. Robertson, P. D. Seymour, and R. Thomas, "Linkless Embeddings of Graphs in 3-Space," *Bulletin of the American Mathematical Society*, vol. 28, pp. 84–89, 1993. `http://www.ams.org/journals/bull/1993-28-01/S0273-0979-1993-00335-5/home.html`.

[4] H. E. Ayala, "MMNA Graphs on Eight Vertices or Fewer," 2014.

[5] J. Barsotti. Private Communication, 2013.

[6] J. Thomas, "Properties of 2-Connected MMNA Graphs," *Preprint*, 2014.

[7] B. D. McKay and A. Piperno, "Practical Graph Isomorphism, {II}," *Journal of Symbolic Computation*, vol. 60, no. 0, pp. 94 – 112, 2014. `http://www.sciencedirect.com/science/article/pii/S0747717113001193`.