2013-12-17

Counting Threshold Graphs and Finding Inertia Sets

Christopher Abraham Guzman
Brigham Young University - Provo

Counting Threshold Graphs and Finding Inertia Sets

Christopher Guzman

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Wayne Barrett, Chair
Jeffrey Humpherys
Rodney Forcade

Department of Mathematics

Brigham Young University

December 2013

ABSTRACT

Counting Threshold Graphs and Finding Inertia Sets

Christopher Guzman
Department of Mathematics, BYU
Master of Science

This thesis is separated into two parts: threshold graphs and inertia sets. First we present an algorithmic approach to finding the minimum rank of threshold graphs and then progress to counting the number of threshold graphs with a specific minimum rank.

Second, we find an algorithmic and more automated way of determining the inertia set of graphs with seven or fewer vertices using theorems and lemmata found in previous papers. Inertia sets are a relaxation of the inverse eigenvalue problem. Instead of determining all the possible eigenvalues that can be obtained by matrices with a specific zero/nonzero pattern we restrict to counting the number of positive and negative eigenvalues.

Keywords: Threshold graphs, minimum rank, inertia sets.

Acknowledgments

I want to thank my advisor Professor Barrett for the long hours that he put in to help me complete this thesis. I would also like to thank my family and friends for helping me to remain motivated. Thank you to all of the professors who have aided me in my accomplishments both as an undergraduate and as a graduate student at BYU.

Contents

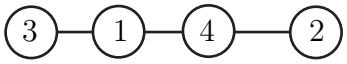
Chapter 1. Introduction

From linear algebra, we learn that symmetric matrices with real entries have many nice properties. One property is that they have all real eigenvalues. Another property, is that they are diagonalizable, or equivalently we can also always find a basis for $\mathbb{R}^n$ consisting of its eigenvectors. Due to the convenient properties of symmetric matrices, they have been studied extensively. One such branch of study is determining what we can know about a symmetric matrix that has a specific zero/nonzero pattern. In other words, if we decide which entries of the symmetric matrix must be zero, and which entries must be nonzero, then what kind of conclusions may we draw about the matrix? Can we determine whether the eigenvalues of the matrix will be positive, negative, or both? Can we determine if it is possible for there to be repeated eigenvalues?

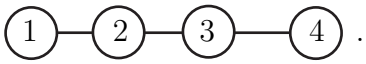
Let $A = \begin{pmatrix} d_1 & a & b & c \\ a & d_2 & e & f \\ b & e & d_3 & g \\ c & f & g & d_4 \end{pmatrix}$. Then A is a general 4×4 symmetric matrix. If we say that the diagonal entries are arbitrary, and only care about whether the entries in the off-diagonal positions are zero or nonzero, then there are $2^{\binom{4}{2}} = 64$ possible zero/nonzero patterns for this matrix.

For an $n \times n$ matrix, there are $2^{\binom{4}{2}}$ possible zero/non-zero patterns. However, we can reduce the number of possibilities if we express the zero/nonzero pattern in terms of a graph. If we number the vertices the same as the row and column numbers, then we say that there is an edge between vertex i and j, where $i \neq j$ if and only if there is a non-zero number in the ij^{th} and ji^{th} entries of the symmetric matrix. Here are two matrices and their corresponding graphs. We can see that both of these matrices will yield the same graph.

$$\begin{pmatrix} d_1 & 0 & a & b \\ 0 & d_2 & 0 & c \\ a & 0 & d_3 & 0 \\ b & c & 0 & d_4 \end{pmatrix}$$



$$\begin{pmatrix} d_1 & a & 0 & 0 \\ a & d_2 & b & 0 \\ 0 & b & d_3 & c \\ 0 & 0 & c & d_4 \end{pmatrix}$$



.

Disregarding the unspecified diagonal entries, these matrices are permutation similar.

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} d_1 & 0 & a & b \\ 0 & d_2 & 0 & c \\ a & 0 & d_3 & 0 \\ b & c & 0 & d_4 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} d_3 & a & 0 & 0 \\ a & d_1 & b & 0 \\ 0 & b & d_4 & c \\ 0 & 0 & c & d_2 \end{pmatrix}$$

Therefore, they share many matrix theoretic properties, such as characteristic polynomial, determinant, trace, rank, and eigenvalues. For the problems we consider, it suffices to consider just one of the possible matrix classes corresponding to each graph. It turns out, that out of the 64 possible zero/nonzero symmetric patterns on 4×4 matrices, there are only 11 graphs.

One of the main sections studied in combinatorial matrix theory is finding out what we can know about a symmetric matrix that has a specific zero/non-zero pattern. Looking at the graphs that correspond to these symmetric matrices is a good way to do this.

1.1 Definitions

Definition 1.1. Given a graph G on n vertices, let $S(G)$ be the set of all real symmetric matrices $A = [a_{ij}]$ such that $a_{ij} \neq 0, i \neq j$, if and only if ij is an edge of G.

Definition 1.2. Given two graphs G and H with $V(G) \cap V(H) = \emptyset$, the *union* of G and H is the graph $(V(G) \cup V(H), E(G) \cup E(H))$ and is written $G \cup H$

Definition 1.3. Given two graphs G and H with $V(G) \cap V(H) = \emptyset$, the *join* of G and H, written $G \vee H$, is the graph with vertex set $V(G) \cup V(H)$ and edge set

$$E(G) \cup E(H) \cup \{uv | u \in V(G) \text{ and } v \in V(H)\}$$

Definition 1.4. Given a graph $G = (V, E)$, the *complement* of the G, is the graph $G^c = (V, E^c)$, where E^c consists of all two element sets from V that are not in E.

Definition 1.5. A *complete graph* on n vertices, denoted K_n, is the graph whose edge set consists of all possible two element sets from the vertex set.

Definition 1.6. We abbreviate the disjoint union $K_1 \cup \cdots \cup K_1$ (n times) to nK_1. So $nK_1 = K_n^c$, the graph consisting of n isolated vertices.

Definition 1.7. The *minimum rank* of a graph G is defined as

$$\text{mr}(G) = \min_{A \in S(G)} \{\text{rank}(A)\}$$

We illustrate this definition with two easy examples of n vertex graphs. Consider the graph nK_1. The minimum rank of this graph may be attained using the $n \times n$ all zeros matrix. Hence the minimum rank of $nK_1 = 0$.

Now consider the complete graph K_n for $n > 1$ We notice that the $n \times n$ all ones matrix is found in $S(K_n)$. Therefore $\text{mr}(K_n) \leq 1$. We also know that for $n > 1$, there must be at least one nonzero entry in an off-diagonal position of the matrix. Hence $\text{mr}(K_n) \geq 1$ (since

any matrix that has at least one nonzero entry must have a rank of at least one). Therefore, $\mathrm{mr}(K_n) = 1$.

1.2 Previous Results

Proposition 1.8. *[1, Proposition 3.6] Given any graph G,*

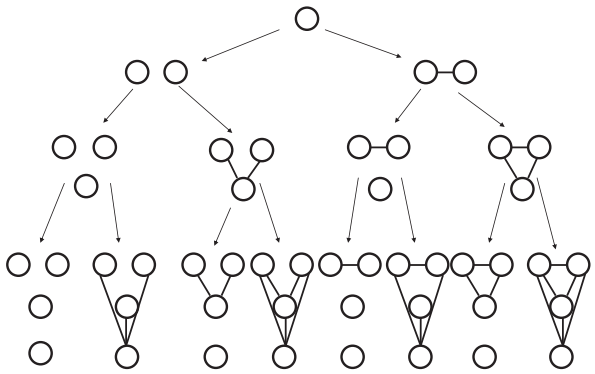
$$\mathrm{mr}(G \vee K_1) = \begin{cases} \mathrm{mr}(G) & \text{if } G \text{ has no isolated vertices} \\ \mathrm{mr}(G) + 1 & \text{if } G \text{ has one isolated vertex} \\ \mathrm{mr}(G) + 2 & \text{if } G \text{ has two or more isolated vertices} \end{cases}$$

Chapter 2. Threshold Graphs

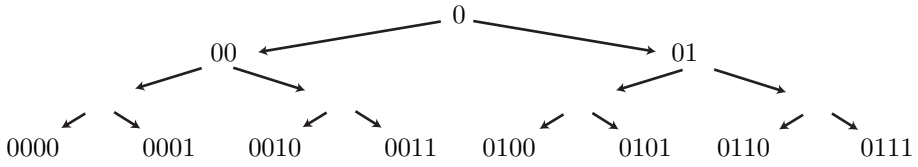
Just as was talked about before, trying to find out what we can know about graphs with specific patterns is one of the main sections studied in combinatorial matrix theory. In this chapter we are going to look at a specific type of graph called a threshold graph. Due to the properties of threshold graphs and Proposition 1.8, we will be able to determine the minimum rank with relative ease.

2.1 What are Threshold Graphs

Threshold graphs are graphs that can be constructed recursively by starting with K_1. We say that K_1 is a threshold graph, and if G is a threshold graph, then so are $G \cup K_1$ and $G \vee K_1$. The following flow chart shows the first few threshold graphs. A move to the left in the flow chart represents the union of the previous graph and K_1. A move to the right represents the join of the previous graph and K_1. It may also be noticed from the construction of threshold graphs and from the flow chart, that there are 2^{n-1} threshold graphs on n vertices.



For convenience, I represent each threshold graph on n vertices with a 0,1 sequence of length n. The 0 represents taking the union of the previous graph with K_1, and a 1 represents a join with K_1. The first entry in this vector is always a 0, because to generate the graph, you always need to start with K_1. We may also think of it as the union of the empty graph with a K_1. There is a one-to-one correspondence between the threshold graphs and these 0,1 sequences.



2.2 Minimum Rank of Threshold Graphs

When calculating the minimum rank of threshold graphs, it is easier to use these 0,1 sequences as opposed to looking at the graph directly. This comes essentially from Proposition 1.8, which tells us how the minimum rank of a graph G will change when you take the join of G and K_1. It says that when you join K_1 to a graph the minimum rank will increase by 0, 1 or 2. We also know that when you take the union of a graph with K_1, the minimum rank will stay the same.

Therefore the algorithmic way to count the minimum rank is:

(i) We check the 0,1 sequence to see if there are places where there are more than two consecutive 0's anywhere. If there are we get rid of 0's until there are only 2 consecutive

0's there.

(ii) If there are any 0's at the end, we just truncate them.

(iii) To then determine the minimum rank, we just count the number of 0's. That is why we made the convention that the first entry is always a 0.

Consequently one can always find the minimum rank of a threshold graph, because as you go forward along the sequence, when you join K_1, the minimum rank will go up by 0,1 or 2 depending on how many isolated vertices there are. When you take the union, the minimum rank will not change. So if $\{t_n\}$ is a sequence corresponding to a threshold graph G, for convenience, let $w(G) = \#1$'s, and $z(G) = \#0'$s, and $\{t_{n_k}\}$ be the subsequence corresponding to the 1's in the sequence. Then:

Lemma 2.1. *The minimum rank of a threshold graph G can be found as follows:*

$$\mathrm{mr}(G) = \sum_{k=1}^{w(G)} \min\{2, n_k - n_{k-1} - 1\}$$

where $n_0 = 0$

Proof. Note that $n_k - n_{k-1} - 1$ will count the number of 0's between the ones in positions n_{k-1} and n_k. Then if there are no zeros between two consecutive ones, the minimum rank will not increase, if there is one zero in between consecutive ones, then the minimum rank will increase by 1, if there are 2 or more zeros in between consecutive 1's, then the minimum rank will increase by 2. By summing all of these increases together, we obtain the minimum rank of the threshold graph. □

Lemma 2.2. *Given a threshold graph G, and the (0,1) sequence corresponding to G, then*

$$w(G) \geq \left\lceil \frac{\mathrm{mr}(G)}{2} \right\rceil$$

Proof.

$$\begin{aligned} \mathrm{mr}(G) &= \sum_{k=1}^{w(G)} \min\{2, n_k - n_{k-1} - 1\} \\ &\leq \sum_{k=1}^{w(G)} 2 \\ &= 2w(G) \implies \\ \frac{\mathrm{mr}(G)}{2} &\leq w(G) \end{aligned}$$

but the number of ones is a whole number $\implies$

$$\left\lceil \frac{\mathrm{mr}(G)}{2} \right\rceil \leq w(G).$$

□

Lemma 2.3. *Given a threshold graph G, and the $(0,1)$ sequence corresponding to G, then*

$$z(G) \geq \mathrm{mr}(G).$$

Proof.

$$\begin{aligned} \mathrm{mr}(G) &= \sum_{k=1}^{w(G)} \min\{2, n_k - n_{k-1} - 1\} \\ &\leq \sum_{k=1}^{w(G)} (n_k - n_{k-1} - 1) \\ &\leq z(G). \end{aligned}$$

□

Theorem 2.4. *Given a threshold graph G on n vertices,*

$$\mathrm{mr}(G) \leq \left\lfloor \frac{2}{3} n \right\rfloor.$$

Furthermore, this bound is attainable for each positive integer n.

Proof.

$$\frac{3}{2}\operatorname{mr}(G) \leq \operatorname{mr}(G) + \left\lceil \frac{\operatorname{mr}(G)}{2} \right\rceil \leq z(G) + w(G) = n \implies \operatorname{mr}(G) \leq \frac{2}{3}n.$$

Since $\operatorname{mr}(G)$ is a whole number, that means that

$$\operatorname{mr}(G) \leq \left\lfloor \frac{2}{3}n \right\rfloor.$$

To show that this bound is attainable, we must consider three different cases.

Case 1: $n = 3k$. Then $\left\lfloor \frac{2}{3}n \right\rfloor = 2k$. Consider the sequence

$$\underbrace{\underbrace{001}\,\underbrace{001}\cdots\underbrace{001}}_{k \text{ times}}.$$

Then using the algorithmic way of calculating the minimum rank of threshold graphs, we find that we don't need to delete any zeros because there are never more than two consecutive zeros. We also don't need to truncate any zeros because there are none at the end. Then we need only count the zeros. There are two for every three numbers in the sequence. Hence there are $2k$ zeros and therefore, the minimum rank is $2k$.

Case 2: $n = 3k + 1$. Then $\left\lfloor \frac{2}{3}n \right\rfloor = 2k$. Consider the sequence

$$\underbrace{\underbrace{001}\,\underbrace{001}\cdots\underbrace{001}}_{k \text{ times}}0.$$

Then using the algorithmic way of calculating the minimum rank of threshold graphs, we find that we don't need to delete any zeros because there are never more than two consecutive zeros. We do however need to truncate the terminal zero. Then we need only count

the remaining zeros. This is simple however, because it has turned into the same sequence as above. Hence there are $2k$ zeros and therefore, the minimum rank is $2k$.

Case 3: $n = 3k + 2$. Then $\left\lfloor \frac{2}{3}n \right\rfloor = 2k + 1$. Consider the sequence

$$\underbrace{001\ 001 \cdots 001}_{k \text{ times}}\ 01.$$

Then using the algorithmic way of calculating the minimum rank of threshold graphs, we find that we don't need to delete any zeros because there are never more than two consecutive zeros. We also don't need to truncate any zeros because there are none at the end. Then we need only count the zeros. There are two for every three numbers in the sequence when we take out the last two elements of the sequence. Hence there are $2k + 1$ zeros and therefore, the minimum rank is $2k + 1$.

Therefore, this bound is attainable for each positive integer n. □

2.3 Counting Threshold Graphs

In combinatorics, there are two types of problems that occur repeatedly. One is the existence of something. For example the existence of a formula for the maximum minimum rank. Many times this may be difficult to find. In the case of threshold graphs, it was fairly simple to find the formula for the maximum minimum rank. When the existence of a formula or arrangement is fairly simple, then sometimes we move onto the second type of combinatorial problem. That is, the enumeration of said arrangement. That is what the next three theorems are doing.

Theorem 2.5. *For each positive integer k, there is one threshold graph on $3k$ vertices that attains the maximum minimum rank.*

Proof. Given a threshold graph G on $n = 3k$ vertices, we know that the maximum possible

minimum rank of G is

$$\left\lfloor \frac{2}{3}3k \right\rfloor = \lfloor 2k \rfloor = 2k$$

By previous lemmata, we know that

$$w(G) \geq \left\lceil \frac{\text{mr}(G)}{2} \right\rceil = \left\lceil \frac{2k}{2} \right\rceil = k$$

$$z(G) \geq \text{mr}(G) = 2k.$$

By using these two inequalities, we get that

$$n = z(G) + w(G) \geq 2k + k = 3k = n.$$

Therefore we have all equalities and $z(G) = 2k$ and $w(G) = k$. That means that each 1, or each time we join K_1, has to increase the minimum rank by 2, and each 0, or each time we take the union with K_1, has to count towards the minimum rank. In other words, we can't take the union more than two times in a row or else at least one of those unions won't help to increase the minimum rank when a join occurs. When we consider our minimum rank equation from Lemma 5.1, the only way for this to happen is to have the pattern 001 repeated k times which accounts for $w = k$ and $z = 2k$ and yields the correct minimum rank. □

Theorem 2.6. *For each positive integer k, there are $k + 1$ threshold graphs on $n = 3k + 2$ vertices that attain the maximum minimum rank.*

Proof. Given a threshold graph G on $3k + 2$ vertices that reaches the maximum minimum rank, that means that

$$\text{mr}(G) = \left\lfloor \frac{2}{3}(3k + 2) \right\rfloor = \left\lfloor 2k + \frac{4}{3} \right\rfloor = 2k + 1$$

By previous lemmata, we know that

$$w(G) \geq \left\lceil \frac{\mathrm{mr}(G)}{2} \right\rceil = \left\lceil \frac{2k+1}{2} \right\rceil = k+1$$

$$z(G) \geq \mathrm{mr}(G) = 2k+1.$$

Using these two inequalities, we get

$$n = z(G) + w(G) \geq 2k+1+k+1 = 3k+2 = n$$

Therefore we have all equalities. Each 0 needs to contribute to the minimum rank, and each 1 needs to increase the minimum rank by 1 or 2. Therefore, we need to be able to partition the $2k+1$ zeros into $k+1$ parts using 1's and 2's. The only option for this is k 2's, and one 1. That means that we need k copies of 001 and one instance of 01. The 01 can go before or after any of the 001 and that will generate a different threshold graph. There are $k+1$ places to put the 01, therefore, there are $k+1$ threshold graphs on $3k+2$ vertices that attain the maximum minimum rank of $2k+1$. □

Theorem 2.7. *For each positive integer k, there are $\binom{k+2}{2}+k$ threshold graphs on $n = 3k+1$ vertices that attain the maximum minimum rank.*

Proof. Given a threshold graph G on $3k+1$ vertices that attains the maximum minimum rank, then

$$\mathrm{mr}(G) = \left\lfloor \frac{2}{3}(3k+1) \right\rfloor = \left\lfloor 2k + \frac{2}{3}k \right\rfloor = 2k$$

Therefore by previous lemmata, we know that

$$w(G) \geq \left\lceil \frac{\mathrm{mr}(G)}{2} \right\rceil = \left\lceil \frac{2k}{2} \right\rceil = k$$

$$z(G) \geq \mathrm{mr}(G) = 2k$$

Therefore, we have one extra character that can either be a 0 or a 1.

Case a: $z(G) = 2k + 1$ and $w(G) = k$. That means that one of the 0's cannot count towards the minimum rank, because that would give a minimum rank of $2k+1$. That means that there would either need to be a 0 at the end, or one set of three 0's in a row. However, since we have $w(G) = k$ they would all need to increase the minimum rank by 2. That means that we would have to have at least 2 zeros between every 1. So the pattern needs to be k copies of 001 which accounts for $3k$ of the elements, and then we can place a 0 before any of these repetitions (creating three consecutive zeros), or we can place the 0 at the end of the sequence. Each of these constructions will generate a different threshold graph. There are $k + 1$ places to place this 0. Hence there are $\binom{k+1}{1}$ ways to do this.

Case b: $z(G) = 2k$, $w(G) = k + 1$, and the extra 1 does not increase the minimum rank. That means that it had to come after another 1, for only when we join K_1 to a connected graph, does the minimum rank not increase. All of the 0's must count towards the minimum rank, but the minimum rank can only be increased k times, therefore, we must also have the pattern of k copies of 001. The only places that we can put the extra 1 is following one of these copies which will each generate a different threshold graph. There are k other 1's, therefore there are k threshold graphs like this.

Case c: $z(G) = 2k$, $w(G) = k + 1$, and every instance of the number 1 increases the minimum rank. Since $z(G) = 2k$, every 0 needs to add to the minimum rank. Then $2k$ partitioned into $k+1$ parts using 1's and 2's yields $k-1$ copies of 2 and two 1's. That means that there needs to be $k - 1$ copies of 001 and two copies of 01. There are k places to place these 01's which will yield different threshold graphs. There are $\binom{k}{1}$ ways to put the 01's in if we place them together and $\binom{k}{2}$ ways to put the 01's in separately.

Therefore, in total there are

$$\begin{aligned}\binom{k}{1} + \binom{k}{2} + \binom{k+1}{1} + k &= \binom{k+1}{2} + \binom{k+1}{1} + k \\ &= \binom{k+2}{2} + k\end{aligned}$$

□

Now that we have counted the number of threshold graphs that reach the maximum minimum rank, we move on to counting the number of threshold graphs that have a given minimum rank. Specifically, how many threshold graphs on n vertices have minimum rank r.

Definition 2.8. Let $T_{n,r}$ be the number of threshold graphs on n vertices with minimum rank r.

Theorem 2.9. *$T_{n,0} = 1$ for all $n > 0$.*

Proof. Given a threshold graph G where $\text{mr}(G) = 0$,

$$0 = \sum_{k=1}^{w} \min\{2, n_k - n_{k-1} - 1\},$$

so one of two things must occur. Either

$$n_k - n_{k-1} - 1 = 0 \ \forall k, \text{ or}$$

$$w(G) = 0$$

Suppose that $w(G) \neq 0$. We know however that, $n_1 \geq 2$ since $t_1 = 0$, but $n_0 = 0$. That means that $n_1 - n_0 - 1 \geq 2 - 0 - 1 = 1$, which would imply that $\text{mr}(G) \geq 1$. Therefore $w(G) = 0$ and $z(G) = n$. There is only one threshold graph with an all zero sequence, nK_1. □

Theorem 2.10. *$T_{n,1} = n - 1$ for all $n > 1$*

Proof. Suppose that

$$1 = \sum_{k=1}^{w(G)} \min\{2, n_k - n_{k-1} - 1\}$$

That means that $w(G) \neq 0$ and $n_k - n_{k-1} - 1 = 1$ for some k, and for all other k, $n_k - n_{k-1} - 1 = 0$. However, we know from the proof of the previous theorem that $n_1 \geq 2$. If $n_1 > 2$ that

would imply that $\mathrm{mr}(G) > 1$. Therefore, we need $n_1 = 2$. Then, as long as we have $n_k - n_{k-1} - 1 = 0$ for all $1 < k \leq w(G)$, or equivalently, all 1's are consecutive, the minimum rank will be 1. Suppose that we have k ones, then the corresponding sequence is

$$0\underbrace{11\cdots1}_{k \text{ times}}\ \underbrace{00\cdots0}_{n-k-1 \text{ times}}$$

and we will obtain the desired minimum rank. Since the number of ones determines the graph and there are $n-1$ choices for $w(G)$, there are $n-1$ threshold graphs on n vertices with minimum rank 1. □

Theorem 2.11. *There are $(n-2)^2$ threshold graphs on n vertices with minimum rank 2.*

Proof. Case 1a One way to have a threshold graph with minimum rank 2 is to have 2 or more zeros in the beginning of the sequence, and then a 1, and then have all the remaining be zeros. This is the same as having a string of $n-3$ zeros and one 1 with the position of the 1 arbitrary. There are n-2 places to place the 1 and hence $\binom{n-2}{1}$ ways for that to happen.

Case 1b Another way to have a threshold graph with minimum rank 2 is to start of the same way as the previous case, but instead of only having one 1, you have multiple ones. For the threshold graph to still have minimum rank 2, all of the ones would have to be consecutive. Otherwise with at least one zero somewhere in between, the minimum rank would increase. To determine how many sequences fit this case, you can choose at which position the consecutive ones will start and at which position they will end. There are $n-2$ positions, and you are choosing two of them. Hence there are $\binom{n-2}{2}$ ways for that to happen.

Case 2 Now suppose that instead of having two zeros in a row followed by a 1, we have two sets of 01. That will also correspond to threshold graphs that have minimum rank 2.

These sequences are completely determined once we figure out where we want to put the second 0 and the final 1. After that is decided, then we will place 1's in between the first one and the second 0 so that we don't increase the minimum rank, then if there are positions between the second 0 and the final 1 we fill them with 1's, then we will place 0's after the

final 1. Since the first two entries are already determined for us, there are $n-2$ other places to choose from. We want to choose two of those positions, where the first one will tell us where the second zero goes, and the second position tells us where the last 1 goes. Therefore, we have $\binom{n-2}{2}$ ways for that to happen.

Therefore in total we have

$$\begin{aligned} T_{n,2} &= \binom{n-2}{1} + \binom{n-2}{2} + \binom{n-2}{2} \\ &= \binom{n-1}{2} + \binom{n-2}{2} \\ &= \frac{1}{2}(n-1)(n-2) + \frac{1}{2}(n-2)(n-3) \\ &= \frac{1}{2}(n-2)(2n-4) \\ &= (n-2)^2 \end{aligned}$$

□

We now consider $T_{n,r}$ for $r > 2$. $T_{n,r}$ can be determined by considering how to get to those graphs by using smaller threshold graphs.

Let $T_{n,r}|_1$ be the number of threshold graphs on n vertices with minimum rank r whose sequence ends in a 1. In other words, this is the number of connected threshold graphs on n vertices with minimum rank r. Similarly we let $T_{n,r}|_{10}$ be the number of threshold graphs on n vertices with minimum rank r whose sequence ends in a 10.

To obtain a formula for $T_{n,r}$, we note that here are multiple ways of obtaining a threshold graph on n vertices with minimum rank r. You could start with a graph that already had minimum rank r and add something to it that doesn't increase the minimum rank. That means that you could take all the threshold graphs on $n-1$ vertices with minimum rank r and just union a K_1, or you could take all the connected graphs on $n-1$ vertices with minimum rank r and join K_1.

You could also start with a graph that has minimum rank $r-1$ and make sure that when

you join K_1 the minimum rank increases by 1. That means that there was only one isolated vertex. That is the same as taking all of the connected threshold graphs on $n-2$ vertices with minimum rank $r-1$ and then taking the union with a K_1, then taking the join with a K_1.

You could also take the graphs that have minimum rank $r-2$ and make sure that when you join with a K_1, the minimum rank increases by 2. That means that there had to be at least 2 isolated vertices. If we take all of the threshold graphs on $n-3$ vertices with minimum rank $r-2$ and take the union with $2K_1$, then we know for sure that when we take the join with a K_1 the minimum rank will increase by 2.

So from the explanation above, we arrive at the following formula.

$$T_{n,r} = T_{n-1,r} + T_{n-1,r}|_1 + T_{n-2,r-1}|_1 + T_{n-3,r-2} \tag{2.1}$$

Also note that to get the connected threshold graphs on n vertices with minimum rank r, we can either take all the connected threshold graphs on $n-1$ vertices with minimum rank r and join them with K_1, or we can take the threshold graphs on $n-1$ vertices that have minimum rank $r-1$ that we are sure will have the minimum rank increase by 1 when we take the join with K_1. This is the same as taking all of the connected threshold graphs on $n-2$ vertices with minimum rank $r-1$, because then we can take the union followed by the join with K_1. The last way to obtain these graphs, would be to take graphs on $n-1$ vertices with minimum rank $r-2$ that we are sure will have the minimum rank increase by two when we join K_1. Just as before, that means that there had to be at least two isolated vertices. Just as before, if we take all of the threshold graphs on $n-3$ vertices with minimum rank $r-2$ and take the union with $2K_1$ and then the join with K_1 we know that the resulting graph will be a connected threshold graph on n vertices with minimum rank r.

That gives us the following equation:

$$T_{n,r}|_1 = T_{n-1,r}|_1 + T_{n-2,r-1}|_1 + T_{n-3,r-2} \tag{2.2}$$

By rewriting (2.1) and (2.2) we get the following two equations

$$T_{n,r} - T_{n-1,r} - T_{n-3,r-2} = T_{n-1,r}|_1 + T_{n-2,r-1}|_1 \tag{2.3}$$

$$T_{n-1,r}|_1 + T_{n-2,r-1}|_1 = T_{n,r}|_1 - T_{n-3,r-2} \tag{2.4}$$

It follows then from (2.3) and (2.4) that

$$T_{n,r}|_1 = T_{n,r} - T_{n-1,r} \tag{2.5}$$

Using (2.5), and the fact that n and r are arbitrary, we get the following two equations

$$T_{n-1,r}|_1 = T_{n-1,r} - T_{n-2,r} \tag{2.6}$$

$$T_{n-2,r-1}|_1 = T_{n-2,r-1} - T_{n-3,r-1} \tag{2.7}$$

Now we plug (2.6) and (2.7) into (2.1) and we find our overall formula

$$T_{n,r} = 2T_{n-1,r} - T_{n-2,r} + T_{n-2,r-1} - T_{n-3,r-1} + T_{n-3,r-2} \tag{2.8}$$

The following is a program in MATLAB that will find the number of threshold graphs on n vertices with minimum rank r. It takes as inputs n and r and outputs $T_{n,r}$.

One of the things we can do with this recursive equation, is to give a short alternative verification for the equation $T_{n,2} = (n-2)^2$.

Proof. We proceed by way of induction. First we consider $T_{2,2}$. Note that for $n = 2$, and by Lemma 2.4 we know that the maximum minimum rank is 1. Therefore, there are no threshold graphs on 2 vertices that have minimum rank 2. Hence we have $T_{2,2} = 0 = (2-2)^2$. Now suppose that the formula $T_{k,2} = (k-2)^2$ is correct for all $k < n$, Then consider

$$T_{n,2} = 2T_{n-1,2} - T_{n-2,2} + T_{n-2,1} - T_{n-3,1} + T_{n-3,0}.$$

Then by our inductive hypothesis, Theorem 2.9, and Theorem 2.10 we get

$$\begin{aligned} T_{n,2} &= 2(n-3)^2 - (n-4)^2 + n - 3 - (n-4) + 1 \\ &= 2n^2 - 12n + 18 - n^2 + 8n - 16 + 2 \\ &= n^2 - 4n - 4 \\ &= (n-2)^2 \end{aligned}$$

verifying Theorem 2.11. □

The formula from equation (2.8) may also be used to generate the formulas for the number of threshold graphs on n vertices with a small constant minimum rank r. Since we were able to find the formulas for $T_{n,0}, T_{n,1}$, and $T_{n,2}$, we can use them to successively determine formulas for $T_{n,3}$, $T_{n,4}$, etc.

Consider

$$T_{n,3} = 2T_{n-1,3} - T_{n-2,3} + T_{n-2,2} - T_{n-3,2} + T_{n-3,1}.$$

We use the already known formulas from Theorem 2.10 and 2.11 and find that.

$$T_{n,3} = 2T_{n-1,3} - T_{n-2,3} + (n-4)^2 - (n-5)^2 + (n-4).$$

$$T_{n,3} - 2T_{n-1,3} + T_{n-2,3} = 3n - 13$$

There are multiple ways to solve this recurrence relation that is now only in one variable. I chose to use the method of generating functions. We note first of all, that by using Theorem 2.4 we know that $T_{n,3} = 0$ for $n \leq 4$. Then if we start at $n = 5$, applying the MATLAB program above, we get that the first few nonzero terms of the sequence are 2, 9, 24 and 50.

To use the method of generating functions we take our equation and multiply everything

by x^n and then sum from $n = 5$ to ∞.

$$\sum_{n=5}^{\infty} T_{n,3}x^n - 2\sum_{n=5}^{\infty} T_{n-1,3}x^n + \sum_{n=5}^{\infty} T_{n-2,3}x^n = \sum_{n=5}^{\infty} (3n-13)x^n$$

$$\sum_{n=5}^{\infty} T_{n,3}x^n - 2x\sum_{n=5}^{\infty} T_{n,3}x^n + x^2\sum_{n=5}^{\infty} T_{n,3}x^n = \sum_{n=5}^{\infty} (3n-13)x^n$$

$$(1-2x+x^2)\sum_{n=5}^{\infty} T_{n,3}x^n = \sum_{n=5}^{\infty} (3n-13)x^n$$

Through further computation, we find that

$$(1-x)^2\sum_{n=5}^{\infty} T_{n,3}x^n = \frac{x^6+2x^5}{(1-x)^2},$$

which implies that

$$\sum_{n=5}^{\infty} T_{n,3}x^n = \frac{x^6+2x^5}{(1-x)^4}.$$

This gives us the generating function. Now we can use this generating function to find the formula for $T_{n,3}$. We find that

$$\begin{aligned} \frac{1}{(1-x)^4} &= \frac{1}{6}\sum_{n=3}^{\infty} n(n-1)(n-2)x^{n-3} \\ &= \sum_{n=3}^{\infty} \binom{n}{3} x^{n-3} \end{aligned}$$

Finally we multiply in the last part and solve for the coefficients.

$$\begin{aligned} \frac{x^6+2x^5}{(1-x)^4} &= \sum_{n=3}^{\infty} \binom{n}{3} x^{n+3} + \sum_{n=3}^{\infty} 2\binom{n}{3} x^{n+2} \\ &= \sum_{n=6}^{\infty} \binom{n-3}{3} x^n + \sum_{n=5}^{\infty} 2\binom{n-2}{3} x^n \\ &= 2x^5 + \sum_{n=6}^{\infty} \left[\binom{n-3}{3} + 2\binom{n-2}{3} \right] x^n \end{aligned}$$

The coefficient of the generating function gives us the formula for $T_{n,3}$. Therefore, we have

found that

$$T_{n,3} = \binom{n-3}{3} + 2\binom{n-2}{3},\ n \geq 5 \tag{2.9}$$

We may find the formula for $T_{n,4}$ in a similar fashion. Theorem 2.4 tells us that for $n \leq 5$, $T_{n,4} = 0$. So, the first few nonzero terms of this sequence starting at $n = 6$ are 1, 8, 31, and 85. Then

$$T_{n,4} = 2T_{n-1,4} - T_{n-2,4} + T_{n-2,3} - T_{n-3,3} + T_{n-3,2}.$$

Then we can use Equation (2.9) and Theorem 2.11 we find that

$$T_{n,4} - 2T_{n-1,4} + T_{n-2,4} = \binom{n-5}{3} + 2\binom{n-4}{3} - \binom{n-6}{3} - 2\binom{n-5}{3} + (n-5)^2.$$

$$T_{n,4} - 2T_{n-1,4} + T_{n-2,4} = \frac{5}{2}n^2 - \frac{55}{2}n + 76.$$

Following the same steps as in the first part, except this time starting the sum at $n = 6$

$$(1 - 2x + x^2)\sum_{n=6}^{\infty} T_{n,4}x^n = \sum_{n=6}^{\infty}\left(\frac{5}{2}n^2 - \frac{55}{2}n + 76\right)x^n$$

Therefore,

$$\sum_{n=6}^{\infty} T_{n,4}x^n = \frac{x^6 + 3x^7 + x^8}{(1-x)^5}$$

Through similar calculations as in the previous part, we get

$$\sum_{n=6}^{\infty} T_{n,4}x^n = x^6 + 8x^7 + \sum_{n=8}^{\infty}\left[\binom{n-2}{4} + 3\binom{n-3}{4} + \binom{n-4}{4}\right]x^n.$$

Therefore, in this case we have found that

$$T_{n,4} = \binom{n-2}{4} + 3\binom{n-3}{4} + \binom{n-4}{4},\ n \geq 8 \tag{2.10}$$

Chapter 3. Inertia Sets

In this chapter we will investigate the well known concept of the inertia of a symmetric matrix and its combinatorial extension to the set of all possible inertias of matrices in $S(G)$ for a given undirected graph G.

I used a previous paper [2], in which it is outlined by various theorems and lemmata how to find the inertia sets of graphs on 7 or fewer vertices. It is complex and time consuming however to carry out the necessary calculations on paper, and I therefore, set up a more algorithmic way and implemented it through MATLAB.

I wrote a recursive program that will output these inertia sets. One difficult aspect of writing a program like this in MATLAB is that there is no nice way of representing the graph except by a matrix. I chose to use the adjacency matrix to represent these graphs. However, there are many different adjacency matrices corresponding to different vertex labelings that represent the same graph. All of these matrices are permutation similar, so they have many of the same properties. My program had to be able to determine properties about a graph given any of the adjacency matrices that represent it.

The program is also able to find the inertia set for many other graphs that have more than seven vertices.

3.1 More Definitions

Definition 3.1. Given a graph G on n vertices, the *adjacency matrix* $A(G) = [a_{ij}]$ such that

$$a_{ij} = \begin{cases} 1 & i \neq j,\ i \sim j \\ 0 & \text{otherwise} \end{cases}$$

Here $i \sim j$ means that i is adjacent to j.

Definition 3.2. The *row sum vector* is found by multiplying the adjacency matrix by the all ones vector. In other words it is the column vector formed by taking all of the row sums.

Definition 3.3. Given a graph G on n vertices, the *degree* of a vertex v_i is the number of other vertices that are adjacent to it. One way to find the degree of each vertex using the adjacency matrix is by looking at the row sum vector.

Definition 3.4. Given a graph G on n vertices, the *Laplacian Matrix* is defined as

$$L(G) = \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \sim j \text{ and } i \neq j \\ 0 & \text{otherwise} \end{cases}$$

where $\deg(v_i)$ is the degree of vertex v_i.

Definition 3.5. Given a graph G on n vertices, H is called an *induced subgraph* if you can get from G to H by removing a number of vertices and their corresponding edges.

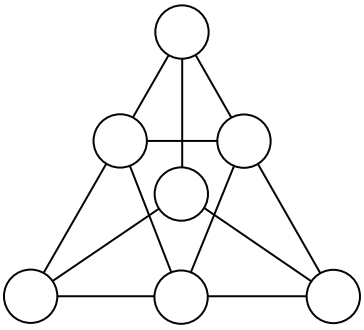
Definition 3.6. A set of vertices in a graph G is an *independent set* if its vertices are pairwise non-adjacent. The *independence number* of G, denoted $\alpha(G)$, is the size of the largest independent set in G.

Definition 3.7. A *bipartite* graph is a graph whose vertices can be partitioned into two disjoint (possibly empty) independent sets. Given $m, n \in \mathbb{N}$, the graph $mK_1 \vee nK_1$ is called a *complete bipartite graph* and is written $K_{m,n}$.

Definition 3.8. A complete *tripartite* graph is a graph whose vertices can be partitioned three independent sets, where each vertex is adjacent to every vertex in the two set to which it does not belong. It is denoted $K_{p,q,r}$ where p, q, and r are the sizes of the independent sets.

Definition 3.9. A *k-connected* graph is one in which there does not exist a set of $k-1$ vertices whose removal results in either a disconnected or trivial graph.

Definition 3.10. The graph $Q_3Y\Delta$ is defined to be



Definition 3.11. Let G_1 and G_2 be graphs with at least two vertices, each with a vertex labeled v. The *vertex sum* of G_1 and G_2 is found by identifying the vertices labeled v in both graphs. In other words, you create a new graph by leaving all the edges from both graphs and taking the two vertices labeled v and making them one vertex.

Definition 3.12. Given a graph $G = (V, E)$, a *2-separation* (if it exists) is a pair of subgraphs (G_1, G_2) satisfying the following:

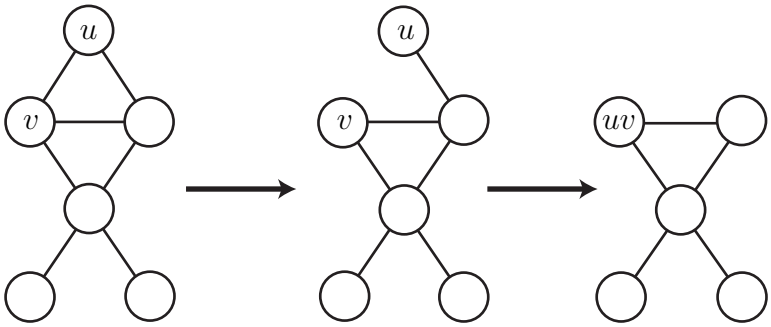
(i) $V(G_1) \cup V(G_2) = V$

(ii) $|V(G_1) \cap V(G_2)| = 2$

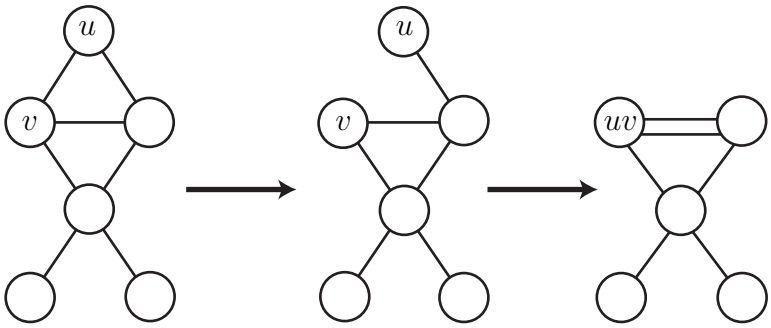
(iii) $E(G_1) \cup E(G_2) = E$

(iv) $E(G_1) \cap E(G_2) = \emptyset$.

Definition 3.13. Let G be a graph on at least two vertices and let uv be an edge of G. An *edge contraction* on the edge uv is found by deleting the edge between u and v, and then coalescing u and v into one vertex.

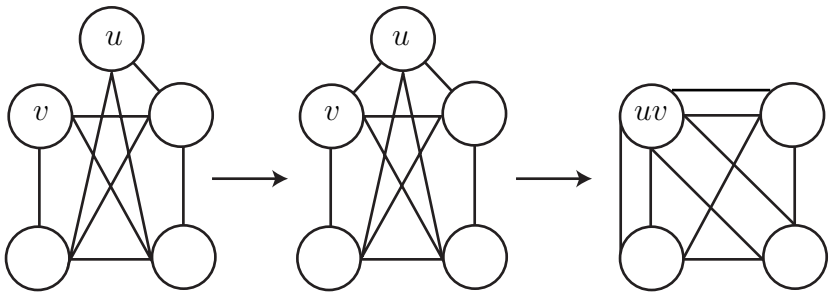


Sometimes this creates multiple edges between two vertices called *parallel edges*.



We will also need an extension of the previous definition to include the case in which u and v are not adjacent.

Definition 3.14. Let G be a graph on at least two vertices, and let u and v be vertices of G. We construct a new graph G/uv by inserting the edge uv if it doesn't exist and then taking the edge contraction on the edge uv. If the edge uv already exists, then G/uv is merely the edge contraction on the edge uv.



For convenience in working with parallel edges, we consider an expanded definition of $S(G)$.

Definition 3.15. [6, Definition 2.4] Let $G = (V, E)$ be a graph with $V = \{1, 2, \cdots, n\}$ which we allow to have parallel edges. We denote by F_2 the field with only two elements. If F is a field unequal to F_2, we define $S^F(G)$ as the set of all F-valued symmetric $n \times n$ matrices $A = [a_{i,j}]$ with

(i) $a_{i,j} = 0$, if $i \neq j$ and i and j are not adjacent,

(ii) $a_{i,j} \neq 0$ if $i \neq j$and i and j are connected by exactly one edge,

(iii) $a_{i,j} \in F$ if $i \neq j$ and i and j are connected by multiple edges, and

(iv) $a_{i,i} \in F$ for all $i \in V$

For our purposes, we need only consider graphs that will have at most two edges joining a given pair of vertices. Therefore, two vertices could be joined by no edges, one edge, or a pair of parallel edges. [2]

Definition 3.16. Given a graph G, consider the following ways of reducing it:

(i) delete an edge,

(ii) contract an edge,

(iii) delete an isolated vertex.

A *minor* of G is any graph H that can be produced from G by successive application of these reductions.

Definition 3.17. Given an $n \times n$ real symmetric matrix A, the *inertia* of A is the ordered triple $(\pi(A), \nu(A), \delta(A))$, where $\pi(A)$ is the number of positive eigenvalues of A, $\nu(A)$ is the number of negative eigenvalues of A, and $\delta(A)$ is the multiplicity of 0 as an eigenvalue of A. Then $\pi(A) + \nu(A) + \delta(A) = n$ and $\pi(A) + \nu(A) = \text{rank}(A)$.

If the order of A is known, then we lose no information by discarding the third number of the triple.

Definition 3.18. Given a real symmetric matrix A, the *partial inertia* of A is the ordered pair $(\pi(A), \nu(A))$, written $\text{pin}(A)$.

Definition 3.19. Given a graph G, the *inertia set* $\mathcal{I}(G)$ is the set of all possible partial inertias of matrices in $S(G)$. That is,

$$\mathcal{I}(G) = \{(r, s) | \, \text{pin}(A) = (r, s) \text{ for some } A \in S(G)\}$$

Definition 3.20. [5] The *minimum rank line* of a graph G consists of all points $(r, s) \in \mathcal{I}(G)$ such that $r + s = \text{mr}(G)$.

Definition 3.21. Let m and n be non-negative integers with $m \leq n$. When plotted as points in $\mathbb{R}^2$, the set

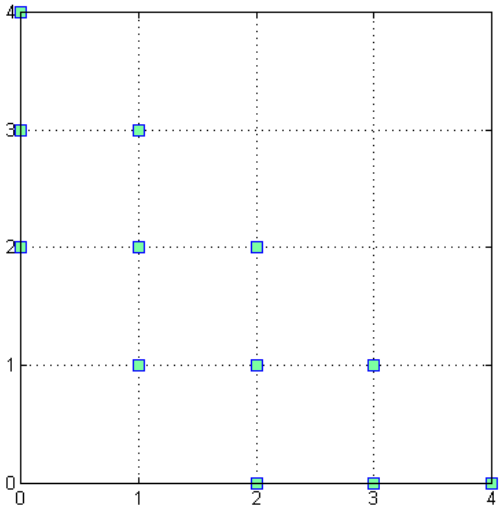
$$\{(r, s) \in \mathbb{N}^2 \, | \, m \leq r + s \leq n\}$$

forms a trapezoid, where $\mathbb{N}$ is the set of nonnegative integers. We denote this set by $\text{T}[m, n]$.

For example, the set

$$\{(2,0), (1,1), (0,2), (3,0), (2,1), (1,2), (0,3), (4,0), (3,1), (2,2), (1,3), (0,4)\}$$

is denoted $\text{T}[2, 4]$.



Observation 3.22. [5] For any graph G on n vertices $\mathcal{I}(G) \subseteq \mathrm{T}[\mathrm{mr}(G), n]$.

Definition 3.23. [5] Let G be a graph on n vertices. If $\mathcal{I}(G) = \mathrm{T}[k, n]$ for some nonnegative integer k, we say that $\mathcal{I}(G)$ is a trapezoid.

Observation 3.24. [5] If $\mathcal{I}(G)$ is a trapezoid then $\mathcal{I}(G) = \mathrm{T}[\mathrm{mr}(G), n]$.

Definition 3.25. [5] Let G be a graph. Then $\mathcal{I}(G)^{\rightarrow}$ is the set that results from adding $(1, 0)$ to each element of $\mathcal{I}(G)$. Similarly, $\mathcal{I}(G)^{\uparrow}$ is the set that results from adding $(0, 1)$ to each element of $\mathcal{I}(G)$.

It will also be necessary to calculate the inertia set of a graph G with parallel edges.

Definition 3.26. [2, Page 4491] For such a graph we say that the simple graph H is a *simple realization* of G, denoted $H \prec G$, if H can be obtained from G by replacing each pair of parallel edges of G by either one edge or no edge: if G has k pairs of parallel edges, there are 2^k simple realizations of G having the same vertex labeling.

Definition 3.27. Given a graph G, possibly with parallel edges,

$$\mathcal{I}(G) = \bigcup_{H \prec G} \mathcal{I}(H).$$

Definition 3.28. An *atom* is a 3-connected graph that is not a join.

3.2 Previous Results

Here is a list of the theorems and propositions that were used. Unless otherwise designated, these can be found in [2]. That paper discusses how to find the inertia set on graphs that have 7 or fewer vertices. Many of those theorems and propositions became the basis for the program that I wrote that will take as input the adjacency matrix and find the inertia set of all 1252 graphs that have 7 or less vertices.

Lemma 3.29. *[3, Lemma 3.6] [Northeast Lemma] Let G be a graph on n vertices and suppose that $A \in S(G)$ with $\text{pin}(A) = (\pi, \nu)$. Then for every pair of integers $r \geq \pi$ and $s \geq \nu$ satisfying $r + s \leq n$, there exists a matrix $B \in S(G)$ with $\text{pin}(B) = (r, s)$.*

Definition 3.30. A *generating set* of an inertia set $\mathcal{I}(G)$, is a set of partial inertias that will give us the inertia set when you apply the Northeast lemma to it.

Minkowski Sum: If R and S are subsets of $\mathbb{N} \times \mathbb{N}$, then

$$R + S = \{(a + c, b + d) : (a, b) \in R \text{ and } (c, d) \in S\}.$$

Proposition 3.31. *Let the graph G be a disjoint union of two graphs G_1 and G_2. Then $\mathcal{I}(G) = \mathcal{I}(G_1) + \mathcal{I}(G_2)$.*

We may extend this further to:

Proposition 3.32. *Let the graph G be a disjoint union of k graphs $G_1, \cdots, G_k$. Then $\mathcal{I}(G) = \sum_{i=1}^{k} \mathcal{I}(G_i)$.*

This proposition is one of the key ones used in my program. If we are given the adjacency matrix of a graph, then to use this proposition, one needs to find the separate components of the graph and find the Inertia set of each of those and then add them together using the Minkowski sum.

Sometimes when we do this though, we end up with points (e, f) such that $e + f > n$, where n is the number of vertices of G. We simply remove these points from the sum. Given a set $R \subset \mathbb{N} \times \mathbb{N}$, then we define

$$[R]_n = \{(a, b) : (a, b) \subseteq R \text{ and } a + b \leq n\}.$$

Theorem 3.33. *Let G be the graph on n vertices that is a vertex sum of G_1 and G_2; i.e., G_1 and G_2 are disjoint graphs on at least two vertices, each with a vertex labeled v, and G*

is the graph defined by identifying the vertices labeled v in G_1 and G_2. Then

$$\mathcal{I}(G) = [\mathcal{I}(G_1) + \mathcal{I}(G_2)]_n \cup [\mathcal{I}(G_1 - v) + \mathcal{I}(G_2 - v) + \{(1,1)\}]_n.$$

Theorem 3.34. *[2, van der Holst] Let (G_1, G_2) be a 2-separation of a graph G with n vertices and let $\{v_1, v_2\} = V(G_1) \cap V(G_2)$. Let H_1 and H_2 be obtained from G_1 and G_2, respectively, by adding an edge(possibly creating a parallel edge) between v_1 and v_2. Then*

$$\begin{aligned}
\mathcal{I}(G) &= [\mathcal{I}(G_1) + \mathcal{I}(G_2)]_n \cup [\mathcal{I}(G_1/v_1v_2) + \mathcal{I}(G_2/v_1v_2) + \{(1,1)\}]_n \\
&\quad \cup [\mathcal{I}(G_1 - v_1) + \mathcal{I}(G_2 - v_1) + \{(1,1)\}]_n \\
&\quad \cup [\mathcal{I}(G_1 - v_2) + \mathcal{I}(G_2 - v_2) + \{(1,1)\}]_n \\
&\quad \cup [\mathcal{I}(G_1 - \{v_1, v_2\}) + \mathcal{I}(G_2 - \{v_1, v_2\}) + \{(2,2)\}]_n \\
&\quad \cup [\mathcal{I}(H_1) + \mathcal{I}(H_2)]_n.
\end{aligned}$$

The following theorem has been reformulated in a more convenient form.

Theorem 3.35. *[4, Theorem 9] Let G be a graph with exactly t isolated vertices. Then*

$$\mathcal{I}(G \vee K_1) = \begin{cases} \mathcal{I}(G) + \{(0,0),(1,0),(0,1)\} & \text{if } t = 0 \\ \mathcal{I}(G) + \{(1,0),(0,1)\} & \text{if } t = 1 \\ \Big[\mathcal{I}(G) + \{(1,1),(t,0),(0,t)\}\Big]_n & \text{if } t \geq 2 \end{cases}$$

Note that Theorem 3.35 generalizes Proposition 1.8 from the minimum rank to inertia sets.

Corollary 3.36. *Given a connected graph G, then*

$$\mathcal{I}(G \vee K_1) = \mathcal{I}(G \cup K_1).$$

Proof. Since G is a connected graph, that means that it doesn't have any isolated vertices.

$$\begin{aligned}\mathcal{I}(G \vee K_1) &= \mathcal{I}(G) + \{(0,0),(0,1),(1,0)\} \\ &= \mathcal{I}(G) + \mathcal{I}(K_1) \\ &= \mathcal{I}(G \cup K_1).\end{aligned}$$

□

Proposition 3.37. *The inertia set of a complete graph on n vertices is $T[1,n]$ for $n > 1$.*

Proposition 3.38. *The inertia set of a complete bipartite graph $K_{m,n}$ is the inertia set generated from the generating set*

$$\{(0, \max(m,n)), (\max(m,n), 0), (1,1)\}.$$

Theorem 3.39. *[5, Theorem 5.1] Let $G \neq K_n$ be a connected graph on $n \geq 3$ vertices for which G^c is a disjoint union of complete bipartite graphs. Then*

$$\mathcal{I}(G) = T[2,n].$$

Proposition 3.40. *If an atom G on n vertices contains K_{n-2} as a minor, then $\mathcal{I}(G) = T[3,n]$.*

Proposition 3.41. *If an atom G on seven vertices is nonplanar and has $K_2 \cup (2K_1)$ as an induced subgraph, where the vertices of the K_2 are not twins (i.e., they have different sets of neighbors), then*

$$\mathcal{I}(G) = T[4,7] \cup \{(2,1),(1,2)\}.$$

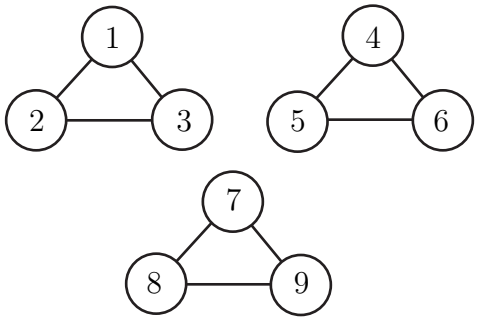
Proposition 3.42. *If G is a planar atom on seven vertices with no $K_{2,2,2}$ or $Q_3Y\triangle$ minor then $\mathcal{I}(G) = T[4,7]$.*

Proposition 3.43. *The remaining 6 atoms on seven vertices, which were not covered by the previous theorems, have* $\mathcal{I}(G) = T[3, 7]$.

3.3 Calculations of the Inertia of a Graph

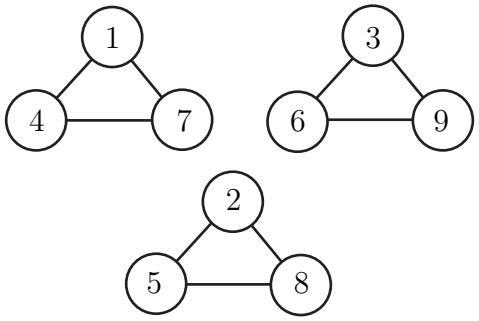
I am now going to discuss the methods that I used to find the inertia sets of graphs in my program given an adjacency matrix.

3.3.1 Inertia Set of a Disjoint Union. There are some cases where it would be easier to find the inertia set of a graph by hand than by a computer. For example, consider the following two graphs and their corresponding adjacency matrices.



$$A_1 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

and

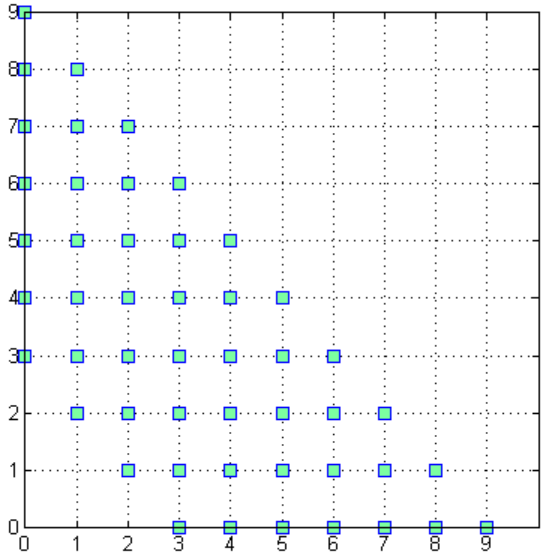


$$A_2 = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

When considering these graphs, we see that they are both the union

$$K_3 \cup K_3 \cup K_3.$$

We note that to find the inertia set of a disjoint union, we need only add the inertia sets of each component together. Hence we need only add the inertia sets for K_3 three times. The result is $T[1,3] + T[1,3] + T[1,3] = T[3,9]$.



However, in my program, I needed to be able to determine the inertia set from both of those adjacency matrices. In the first adjacency matrix, it is fairly easy to see that there are three disjoint parts by taking the 3×3 blocks down the diagonal and seeing that there are no other places that have ones. It is more difficult to see that from the second adjacency matrix. I used a preexisting program from MATLAB called *graphconncomp*. What this program does, is it takes in an adjacency matrix, and it outputs two things. The first output is a positive integer which represents the number of components. The second output is a vector that tells you which vertices correspond to which component.

For example, if we were to run

$$[S1, C1] = graphconncomp(sparse(A1), 'directed', false)$$

we would get

$$S1 = 3$$

$$C1 = \begin{bmatrix} 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 \end{bmatrix}$$

if we were to run

$$[S1, C1] = graphconncomp(sparse(A2), 'directed', false)$$

we would get

$$S1 = 3$$

$$C1 = \begin{bmatrix} 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 \end{bmatrix}$$

Once I was able to determine which vertices corresponded to which component, I could take the principal submatrices corresponding to each component and plug those back into my program so that it could find the inertia set of each of those components separately. That is how I was able to use Proposition 3.32 in my program.

From this point on, we will only need to consider connected graphs. That is because whenever we have a disconnected graph, we can find the inertia sets of the connected components separately.

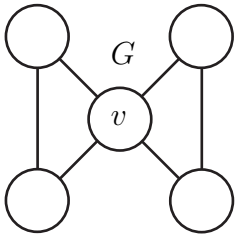
3.3.2 Inertia Set of a Complete Graph. If the adjacency matrix is the 1×1 matrix $[0]$, then we know that we have K_1 as our graph. If this is the case, then the inertia set is $\{(0,0),(0,1),(1,0)\}$.

If we are given an adjacency matrix that is $n \times n$ for $n > 1$, then to check if this adjacency matrix represents a complete graph, we check to see if the degree of every vertex is $n-1$, equivalently, all entries in the row sum vector are $n-1$. If that is the case, then we get the inertia set from Proposition 3.37.

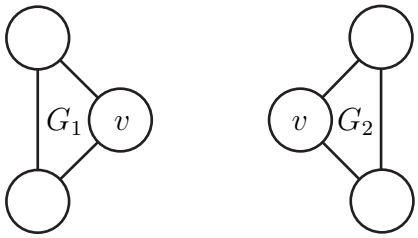
3.3.3 Inertia Set of a Complete Bipartite Graph. To find out if a graph is a complete bipartite graph, we may check the complement to see if the complement is made up of two complete graphs. To do this in my program, I find the complement by taking a matrix of the same size of all ones, subtract the identity matrix of the same size and then subtract the adjacency matrix. Next, check to see if there are two components using the *graphconncomp* program. Then if there are two components, I use the output of *graphconncomp* to extract the principal submatrices and check if those correspond to complete graphs using the row sum vector as in the previous section.

3.3.4 The Inertia Set of a Join. A graph G on n vertices is a join with another graph G_1 on $n-1$ vertices and K_1, if and only if it has a *dominating vertex*, that is a vertex adjacent to the rest of the vertices. So if there is a vertex that has degree $n-1$, we can reduce finding the inertia set to finding the inertia set of G_1 and finding out how many isolated vertices there are in G_1, then apply Theorem 3.35. To find out if there is a dominating vertex, we take the row sum vector and do a simple search to see if any of the degrees are equal to $n-1$. Then to see how many isolated vertices there are in G_1, we just need to find the principal submatrix that corresponds to G_1. We take that matrix and find the row sum vector. The number of 0 entries in the vector is the number of vertices that have degree 0, which is the same as the number of isolated vertices. Now we apply Theorem 3.35

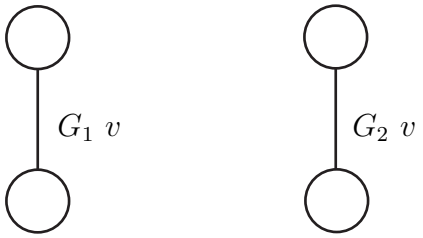
3.3.5 Inertia Set of a Vertex Sum. Now let's consider the following graph.



which we notice is a vertex sum of the following two graphs:

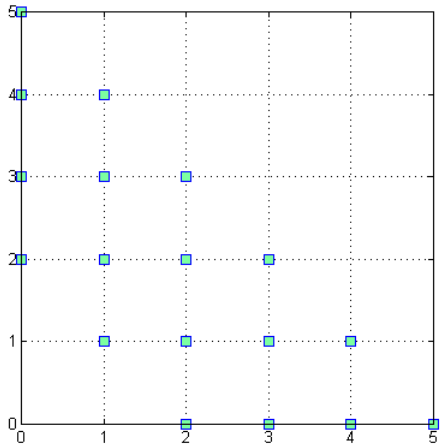


So in this case, to find the inertia set, we are able to use Theorem 3.33. So we find the inertia set of G_1 and G_2 separately and add them together. We see that both of these graphs are K_3, so their inertia sets are $T[1,3]$. Then we find the inertia set of $G_1 - v = K_2$ and $G_2 - v = K_2$



and add those together along with the set $\{(1,1)\}$. So we end up with

$$[T[1,3] + T[1,3]]_5 \cup [T[1,2] + T[1,2] + \{(1,1)\}]_5 = T[2,5].$$



The procedure just outlined can be programmed as follows. To recognize whether or not a graph is a vertex sum in my program, I would run through every vertex and see if its removal gave me a disconnected graph. To do that I would delete column and row i and check to see if the resulting matrix corresponds to a disconnected graph. To do this I use the same program, which found the components of a graph, referenced in the beginning of the section. If I find a vertex that separates the graph, then I can use that vertex as my vertex labeled v in Theorem 3.33. Now that I have found my vertex v, I need to find the vertices that correspond to $G_1 - v$ and $G_2 - v$ because then by adding the vertex v we will have G_1 and G_2. Since I used *graphconncomp* to find out if the removal of the vertex disconnected the graph, I am also able to use that program to determine which vertices correspond to the components.

At this point, I would be able to find the principal submatrices corresponding to all of the subgraphs needed for Theorem 3.33. So I need only find the inertia set of each of those graphs and plug them back into the formula.

3.3.6 Inertia Set of a 2-Separation. This part of the program was a little more difficult. The first step, is to figure out if the graph has a 2-separation. I went about this in much the same way that I went about determining the presence of a cut vertex. I take every possible combination of two vertices at a time, and check to see if their removal gives

a disconnected graph. If the removal of two vertices disconnects the graph, then I proceed to the next step. I save the two vertices, and call them v_1 and v_2 as in Theorem 3.34. First I find the vertices corresponding to the graphs $G_1 - \{v_1, v_2\}$ and $G_2 - \{v_1, v_2\}$ by using the program *graphconncomp*. Now that I have the vertices corresponding to those two graphs, I can find the vertices corresponding to G_1, G_2, $G_1 - \{v_1\}$, $G_1 - \{v_2\}$, $G_2 - \{v_1\}$, and $G_2 - \{v_2\}$. There are only four remaining graphs that I need to find to be able to use the formula from Theorem 3.34. Since I have found the vertices corresponding to G_1 and G_2, I can find H_1 and H_2, by finding the principal submatrices corresponding to G_1 and G_2. Then I add one to the positions corresponding to the edge v_1v_2. If there was already an edge there, then by adding one, I create a parallel edge. If there wasn't an edge then I merely create one.

To find the inertia set of a graph with parallel edges, I needed to find a way of representing parallel edges. I changed the way that my program would take in the adjacency matrices. If there was a two in an off-diagonal position of the matrix, then I said that represented a parallel edge. Whenever that happened, I would take that matrix and create two new matrices. The first with a one in that same position, and the second with a zero in that position. In other words, I would rerun the program once with an edge there, and once with no edge there.

The last two graphs that I needed to find were G_1/v_1v_2 and G_2/v_1v_2. To get G_1/v_1v_2 when starting with the adjacency matrix corresponding to G_1, we need to make v_1 and v_2 become one vertex, but we also need to make sure that all of the edges that were adjacent to v_1 and v_2 will still be adjacent to the new combined vertex v. To do this using the adjacency matrix, I would replace the column corresponding to v_1 by the sum of the vectors corresponding to v_1 and v_2. Then I would replace the row corresponding to v_1 by the sum of the rows corresponding to v_1 and v_2. This would ensure that all of the edges that were adjacent to either v_1 or v_2 are now adjacent to the new vertex represented by the new row and column. If v_1 and v_2 were adjacent to the same vertex, then the new row and column will have a 2 in that place. Just as before, that represents a parallel edge. Sometimes, this

will create a nonzero entry on the diagonal of the matrix. If this occurs then we change it to a zero. Lastly, we change the column and row corresponding to v_2 to zeros. We don't delete this row and column to allow the convenience of working with the same size matrices. We find G_2/v_1v_2 in the same way that we found G_1/v_1v_2. Now that we have found the matrices corresponding to all of the graphs in Theorem 3.34, we need only plug their inertia sets into that formula.

3.3.7 Applying Theorem 3.39. To apply this theorem, I first found the complement as in a previous part. Then I took each of the components of the complement. I then checked each of those components to see if they were bipartite graphs just as in previous section. If each of those parts was a bipartite graph, then this theorem was satisfied, and I could find the inertia set.

3.3.8 The Inertia Set of an Atom on n Vertices that contains K_{n-2} as a Minor. At this point in my program, I have found the inertia set for all graphs that are not atoms on 7 or fewer vertices. So when trying to apply Proposition 3.40, we need only check that K_{n-2} is a minor. This part of the program was fairly difficult conceptually. I had to be able to contract or delete edges and delete vertices and find a complete graph as a minor. There were two cases that I considered in my program.

The first case was if I could find three vertices, so that when I removed them, I was left with a complete graph; if those three vertices induce a component of the graph, and if collectively those vertices are adjacent to the remaining vertices, then we will have K_{n-2} as a minor. That is because we will be able to edge contract twice since there are at least two edges that exist between the three vertices, and then because collectively they were adjacent to the remaining vertices, and because the remaining vertices are all adjacent to each other, we will be left with a complete graph once we delete the parallel edges. To do this in my program, I would go through every set of three vertices and check each of those parts. First I would check to see if when I removed those three vertices I was left with a complete graph.

I did that by checking the row sum vector as in the previous sections. Then if I found that, I would see if those three vertices together formed a connected graph. Since there were only three vertices, if there were at least two edges, then it was connected. So I just checked to make sure that there was at least one vertex with degree two. Then if that was true, I would sum the three columns corresponding to each of the vertices and I would check to make sure that each entry was nonzero.

The next case is similar, except now instead of finding three connected vertices, I needed to find two pairs of adjacent vertices. So I would try to find four vertices such that if I removed them I was left with a complete graph. I would do that the same way that I did in the previous case except I would remove every set of four vertices instead of every set of 3 vertices. Then I would find the row sum vector and check if the remaining vertices formed a complete graph. If that was the case, then I needed to make sure that I could separate the four vertices into two sets, each with two vertices that were adjacent to each other. To do that I would find the principal submatrix of each and make sure that both vertices had degree 1. If that was the case, then I had to make sure that each pair of vertices found were collectively adjacent to all remaining vertices, and at least one vertex in the other pair. I do this in much the same way that I did the previous case.

If either of these cases is true, then I apply Proposition 3.40. Otherwise K_{n-2} is not a minor.

3.3.9 Inertia Set of Graphs Satisfying Proposition 3.41. First, I would take all possible sets of four vertices. Then I would check to see if their induced subgraph had only one edge. If that is true, then we have $K_2 \cup (2K_1)$ as an induced subgraph. To see this, I merely check the row sum vector, and make sure that there are two vertices with degree one, and two vertices with degree 0. Next I check to make sure that the two vertices that form the K_2 i.e. the two vertices with degree one are not twins. To do that, I make sure that their neighbors are not the same. I do this by adding ones to the diagonal of the original adjacency matrix and then comparing the columns corresponding to each vertex. If they are the same,

then they are twins and I would keep checking, or move on to the next part. If they are not twins, then I would move onto the next part of the proposition, that is to determine if the graph is nonplanar. To be nonplanar, a graph must either have K_5 as a minor, or it must have $K_{3,3}$ as a minor. Since we are only dealing with graphs on 7 vertices at this point, and we already know that it doesn't have K_{n-2} (K_5 in this case) as a minor, then the only way for these graphs to be nonplanar is for them to have $K_{3,3}$ as a minor. So now we only need to check to see if $K_{3,3}$ is a minor. Examining all of the graphs in [2] the determination of whose inertia set relies on this proposition, the minor $K_{3,3}$ was found by finding the edge contraction on the K_2 subgraph that we found in the first part. So in my program, I would find the edge contraction on the edge in the K_2 subgraph and then I would check to see if that graph contained $K_{3,3}$ as a minor. To check this I would find the complement of the graph. Then if the complement is a subgraph of $2K_3$, then $K_{3,3}$ is a minor of the original graph. To check to see if the complement is a subgraph of $2K_3$, I would check the number of components of the complement, as well as the size of the components. If there were four or more components then it is true. If there are 3 components, and the biggest component has 3 vertices, then it is true. If there are 2 components, and both components have 3 vertices, then it is true, and if there is only one component then it is not true. To check all of these things I used the *graphconncomp* program built into MATLAB.

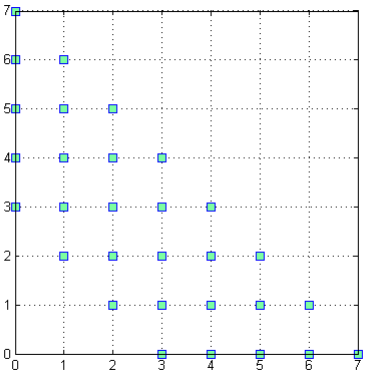
3.3.10 The Inertia Set of the Remaining Graphs on Seven Vertices. There are only two propositions remaining, so if I can find all of the graphs that satisfy one of them, then the rest of the graphs will be satisfied by the other proposition. I first examined the 6 atoms in Proposition 3.43 and found, upon deleting edges, that $Q_3Y\triangle$ is a subgraph (and therefore a minor) of each. Therefore Proposition 3.42 could be applied without considering whether or not $K_{2,2,2}$ is a minor. The difficulty in finding $Q_3Y\triangle$ as a subgraph is that there are many vertex labelings that could occur for this graph. So we cannot just compare the adjacency matrices; we need to compare something else. I looked at all of the graphs on 7 vertices that have the same number of edges as $Q_3Y\triangle$, and the characteristic polynomial of

$Q_3Y\triangle$ was unique.

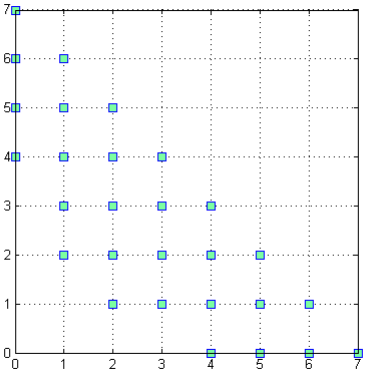
Therefore, if I have a graph that has the same number of edges as $Q_3Y\triangle$, then I compare the characteristic polynomial, and if they are the same, then I know the inertia set. If the graph has more edges than $Q_3Y\triangle$, then I delete edges until it has the right number and compare the characteristic polynomials again. However, I need to check all the possible combinations of deleting the right number of edges. If it turns out that $Q_3Y\triangle$ is a minor of the graph, then I know the inertia set from Proposition 3.43. Otherwise, we get the inertia set from Proposition 3.42.

On the following page, I give the adjacency matrices of 3 atoms that satisfy the hypotheses of Propositions 3.43, 3.41, and 3.42, respectively, and to the right of each I give the output from my program. The program follows on the next 19 pages.

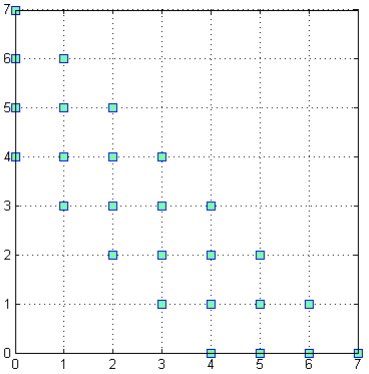
$$G1005 = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$



$$G1004 = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$



$$G999 = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$



The following program executes my main program.

```
function RunInertiaSet(A)
inerset = InertiaSet(A);


if ~isempty(inerset)
    figure
    plot(inerset(:,1),inerset(:,2),'bs','MarkerFaceColor',[.49 1 .63]);
    grid on
    axis square
    set(gca, 'XTick', 0:max(max(inerset)))
    set(gca, 'YTick', 0:max(max(inerset)))
else
    disp('The program is not yet capable of finding the inertia set')
end
```

This is my main program.

```
function inerset = InertiaSet(A)%A is the adjacency matrix of the graph G
inerset=[];
n = size(A,1);
%This part checks to see if any of the entries in the incoming matrix are
%not either 0 or 1. This part is skipped at the beginning because only an
%adjacency matrix is input. However, later on in the program I create
%matrices that have 2 as inputs to represent a parallel edge which can
%either be used or not used and you need to take into account both. If
%this happens, then I rerun the program once with the edge there, and once
%without.
%This will only find the first time that there is an entry larger than 1
%and rerun the program. I chose to do it that way because it was easier
```

```
%than figuring out how many times it happened (k) and making a loop that
%would find the inertia set for 2^k different graphs. Since the program was
%already written recursively, it seemed better to just have this part use
%that property of the program.
[I,J] = find(A>1,1);
if ~isempty(I)
    %This makes that entry that we found a 1.
    A1 = A;
    A1(I,J) = 1;
    A1(J,I) = 1;
    %This makes that entry that we found a 0.
    A2 = A;
    A2(I,J)=0;
    A2(J,I)=0;
    %This finds the union of those two inertia sets.
    inerset = unique([InertiaSet(A1);InertiaSet(A2)],'rows');
    return
end

% Second, check whether or not it is connected. If it is disconnected, do
% each part separately
[S,C] = graphconncomp(sparse(A),'directed',false);
if S>1 %If there are two or more components.
    inerset = [0 0];
    for i=1:S
        %We add up the inertia sets of all of the components.
        inerset =addsets(inerset,InertiaSet(A(C==i,C==i)),n);
```

```
    end
    return
end
%If it is just K1, then return [0 0] as the inertia set. I do this part
%separately because although K1 is a complete graph, it has a different
%inertia set than Kn.
if n==1
    inerset =usegenset([0 0],1);
    return
end

%Check to see if it is a complete graph. If it is, then compute the inertia
%set.
if sum(sum(A))==n*(n-1)
    genset = [0 1;1 0];
    inerset = usegenset(genset,n);
    return
end
%Check to see if it is a complete bipartite graph. If so, then compute the
%inertia set.
if n>3
    complement = ones(n)-eye(n)-A;
    [Sc,Cc] = graphconncomp(sparse(complement),'directed',false);
    %If the complement is disconnected and has two components.
    if Sc==2
        comp1 = complement(Cc==1,Cc==1);
        comp2 = complement(Cc==2,Cc==2);
```

```
        mc1=length(comp1);
        mc2=length(comp2);
        %If each of those components is a complete graph
        if (sum(sum(comp1))== mc1*(mc1-1))&& (sum(sum(comp2))==mc2*(mc2-1))
            mn = max(mc1,mc2);
            inerset = usegenset([1 1;0 mn;mn 0],n);
            return
        end
    end
end

%If the graph is a join of another graph G and K1
I = find(sum(A)==(n-1),1);
if ~isempty(I)
    G=A;
    Acheck = A;
    G(I,:)=0;
    Acheck(I,:)=[];
    G(:,I)=0;
    Acheck(:,I)=[];
    J=sum(sum(Acheck)==0);
    %If it is the join with a connected graph
    if J==0
        inerset= InertiaSet(G);
        return
    %If it is the join with a graph that has one isolated vertex.
    elseif J==1
```

```
        inerset = addsets(InertiaSet(G),[1 0;0 1],n);
        return
    %If it is the join with a graph that has J isolated vertices.
    else
        inerset = addsets(InertiaSet(G),[J 0;0 J;1 1],n);
        return
    end
end

%Find out if there is a cut vertex
for j = 1:n
    temppointsvec = 1:n;
    temppointsvec(j) = [];
    %If any induced subgraph on n-1 vertices is disconnected.
    conn = checkcon(A(temppointsvec,temppointsvec));
    if conn ~=0
        cutvert=j;
        break
    end
end
%Cut vertex formula
if conn==1
    G = A;
    G(cutvert,:)=0;
    G(:,cutvert)=0;
    [S1,C1] = graphconncomp(sparse(G),'directed',false);
    compnum = C1(cutvert);%Since I only zeroed out the row and column, the
```

```
%cut vertex will be its own component. In this line I figure out which
%one it is.
compnotv = 1:S1;
compnotv(compnum)=[];%Then I get rid of it.
inerset1 = [0 0];
for i=1:S1-1 %I got rid of the component corresponding to the cut
    %vertex.
    index = C1==compnotv(i);
    index(cutvert)=true; %Take each component with the cut vertex
    %included and add all of the inertia sets of those graphs together.
    inerset1 = addsets(inerset1,InertiaSet(A(index,index)),n);
end

inerset2 = [0 0];
for i=1:S1-1
    %Then take the inertia set of all of the components with the cut
    %vertex taken out and add all the inertia sets of those graphs
    %together.
    inerset2 =addsets(inerset2,InertiaSet(A(C1==compnotv(i),...
        C1==compnotv(i))),n);
end
inerset2 = addsets(inerset2,[1 1],n);%Then add [1 1] to those
%inertia sets.
inerset=unique([inerset1;inerset2],'rows'); %Then take the union of
%those two things.

return
```

```
end


%Find out if there is a two separation
if conn==0
    for j=1:n
        for k = j+1:n
            temppointsvec = 1:n;
            temppointsvec([j,k]) = [];
            % If any induced subgraph on n-2 vertices is disconnected.
            conn = checkcon(A(temppointsvec,temppointsvec));
            if conn~=0
                conn=2;
                cutvert = [j,k];
                break
            end
        end
        if conn==2
            break
        end
    end
end
%Two separation formula
if conn==2
    G = A;
    %zero out the rows and columns corresponding to the 2-separation
    G(cutvert,:)=0;
    G(:,cutvert)=0;
```

```
[S2,C2] = graphconncomp(sparse(G),'directed',false);
%But now, both of those vertices will be their own components.
compnum = C2(cutvert);
compnotv = 1:S2;
%So I get rid of them as components.
compnotv(compnum) = [];

%finding I(G1)+I(G2)
%G1 is the first component, and G2 the union of all the rest. Still not
%including the vertices corresponding to the 2-separation.
indexG1=C2==compnotv(1);
indexG1(cutvert)=true;
G1 = A;
G1(~indexG1,:)=0;
G1(:,~indexG1) = 0;
indexG2 = C2~=compnotv(1);
G2 = A;
G2(~indexG2,:)=0;
G2(:,~indexG2)=0;
G2(cutvert,cutvert) =0;
inerset1 = addsets(InertiaSet(G1),InertiaSet(G2),n);

%finding I(G1/v1v2)+I(G2/v1v2)+{[1 1]}
G1modv12 = G1;
G1modv12(cutvert(1),:) = G1modv12(cutvert(1),:)+G1modv12(cutvert(2),:);
G1modv12(:,cutvert(1)) = G1modv12(:,cutvert(1))+G1modv12(:,cutvert(2));
G1modv12(cutvert(1),cutvert(1))=0;
```

```
G1modv12(cutvert(2),:)=0;
G1modv12(:,cutvert(2))=0;
G2modv12 = G2;
G2modv12(cutvert(1),:) = G2modv12(cutvert(1),:)+G2modv12(cutvert(2),:);
G2modv12(:,cutvert(1)) = G2modv12(:,cutvert(1))+G2modv12(:,cutvert(2));
G2modv12(cutvert(1),cutvert(1))=0;
G2modv12(cutvert(2),:)=0;
G2modv12(:,cutvert(2))=0;
inerset2 = addsets(InertiaSet(G1modv12),InertiaSet(G2modv12),n);
inerset2 = addsets(inerset2,usegenset([1 1],n),n);

%finding I(G1-v1)+I(G2-v1)+{[1 1]}
indexG1minv1 = indexG1;
indexG1minv1(cutvert(1))=false;
G1minv1=A;
G1minv1(~indexG1minv1,:)=0;
G1minv1(:,~indexG1minv1) = 0;
indexG2minv1 = indexG2;
indexG2minv1(cutvert(1))=false;
G2minv1=A;
G2minv1(~indexG2minv1,:)=0;
G2minv1(:,~indexG2minv1) = 0;
inerset3 = addsets(InertiaSet(G1minv1),InertiaSet(G2minv1),n);
inerset3 = addsets(inerset3,usegenset([1 1],n),n);

%finding I(G1-v2)+I(G2-v2)+{[1 1]}
indexG1minv2 = indexG1;
```

```
indexG1minv2(cutvert(2))=false;
G1minv2=A;
G1minv2(~indexG1minv2,:)=0;
G1minv2(:,~indexG1minv2) = 0;
indexG2minv2 = indexG2;
indexG2minv2(cutvert(2))=false;
G2minv2=A;
G2minv2(~indexG2minv2,:)=0;
G2minv2(:,~indexG2minv2) = 0;
inerset4 = addsets(InertiaSet(G1minv2),InertiaSet(G2minv2),n);
inerset4 = addsets(inerset4,usegenset([1 1],n),n);

%finding I(G1-{v1,v2})+I(G2-{v1,v2})+{[2 2]}
indexG1minv12 = indexG1;
indexG1minv12(cutvert)=false;
G1minv12=A;
G1minv12(~indexG1minv12,:)=0;
G1minv12(:,~indexG1minv12) = 0;
indexG2minv12 = indexG2;
indexG2minv12(cutvert)=false;
G2minv12=A;
G2minv12(~indexG2minv12,:)=0;
G2minv12(:,~indexG2minv12) = 0;
inerset5 = addsets(InertiaSet(G1minv12),InertiaSet(G2minv12),n);
inerset5 = addsets(inerset5,usegenset([2 2],n),n);

%finding I(H1)+I(H2)
```

```
    jkmat = zeros(n);
    jkmat(cutvert,cutvert) = [0 1;1 0];
    H1 = G1+jkmat;
    H2 = G2+jkmat;
    inerset6=addsets(InertiaSet(H1),InertiaSet(H2),n);

    inerset=unique([inerset1;inerset2;inerset3;inerset4;inerset5;...
        inerset6],'rows');
    return
end

if n>=3%Theorem 5.1 in Inertia Sets for Graphs on Six or Fewer Vertices
    Gcomp = ones(n)-eye(n)-A;
    [S C] = graphconncomp(sparse(Gcomp),'directed',false);
    svec = false(1,S);
    for i = 1:S
        Anew = Gcomp(C==i,C==i);
        nnew = size(Anew);
        complement = ones(nnew)-eye(nnew)-Anew;
        [Sc,Cc] = graphconncomp(sparse(complement),'directed',false);
        %If the complement is disconnected and has two components.
        if nnew==1 %K1 is a bipartite graph
            svec(i)=true;
        elseif Sc==2
            comp1 = complement(Cc==1,Cc==1);
            comp2 = complement(Cc==2,Cc==2);
            mc1=length(comp1);
```

```
            mc2=length(comp2);
            %If each of those components is a complete graph
            if (sum(sum(comp1))== mc1*(mc1-1))&&...
                    (sum(sum(comp2))==mc2*(mc2-1))
                svec(i)=true;
            end
        end
    end
    if all(svec)
        inerset = Trap(2,n);
        return
    end
end

%Finding out whether or not K_{n-2} is a minor
if n>5
    verts = nchoosek(1:n,n-3);
    for v=1:size(verts,1)
        if sum(sum(A(verts(v,:),verts(v,:))))==(n-3)*(n-4)
            vertcontract = 1:n;
            vertcontract(verts(v,:))=[];
            G = A;
            G(vertcontract(1),:)=sum(G(vertcontract,:),1);
            if sum(sum(A(vertcontract,vertcontract)))>=4
                if isempty(find(G(vertcontract(1),verts(v,:))==0,1))
                    inerset = usegenset([3 0;2 1;1 2;0 3],n);
                    return
```

```
                end
            end
        end
    end
    verts = nchoosek(1:n,n-4);
    for v=1:size(verts,1)
        if sum(sum(A(verts(v,:),verts(v,:))))==(n-4)*(n-5)
            vertcontract = 1:n;
            vertcontract(verts(v,:))=[];
            for j=2:4
                verts1 = vertcontract([1,j]);
                verts2 = vertcontract;
                verts2([1,j])=[];
                if (sum(sum(A(verts1,verts1)))==2)&& sum(sum(A(verts2,...
                        verts2)))==2
                    G = A;
                    G(verts1(1),:)=sum(G(verts1,:),1);
%                     G(:,verts1(1))=sum(:,G(verts1),2);
                    G(verts2(1),:)=sum(G(verts2,:),1);
%                     G(:,verts2(1))=sum(:,G(verts2),2);
                    if isempty(find(G(verts1(1),verts(v,:))==0,1))&&...
                            isempty(find(G(verts2(1),verts(v,:))==0,1))
                        if length(find(G(verts1(1),verts2)==0,2))~=2&&...
                                length(find(G(verts2(1),verts1)==0,2))~=2
                            inerset = usegenset([3 0;2 1;1 2;0 3],n);
                            return
                        end
```

```
                    end
                end
            end
        end
    end
end

if n==7
    %nonplanar with K_2 U 2K_1 on 7 vertices without twins
    for k=1:size(verts,1)
        vertcontract = 1:n;
        vertcontract(verts(k,:))=[];
        G = A(vertcontract,vertcontract);
        if sum(sum(G))==2
            [I J]=find(G,1);
            NewA = A+eye(size(A));
            if sum(abs(NewA(vertcontract(I),:)-NewA(vertcontract(J),:)))~=0
                K33check = [vertcontract(I) vertcontract(J)];
%                 for p=1:n
%                     findK33 = 1:n;
%                     findK33(p)=[];
%                     if checkK33(A(findK33,findK33))
%                         inerset = usegenset([2,1;1,2;4,0;0,4],n);
%                         return
%                     end
%                 end
%                 [K33checki K33checkj] = find(A);
```

```
%                 K33checkspots =K33checki< K33checkj;
%                 K33check = [K33checki(K33checkspots) ...
%                       K33checkj(K33checkspots)];
%                 for p = 1:size(K33check,1)
                    Aidentify = A;
                    Aidentify(K33check(1),:) = Aidentify(K33check(1),:)...
                        +Aidentify(K33check(2),:);
                    Aidentify(:,K33check(1)) = Aidentify(:,K33check(1))...
                        +Aidentify(:,K33check(2));
                    Aidentify = Aidentify-diag(diag(Aidentify));
                    identvert = 1:n;
                    identvert(K33check(2))=[];
                    Aidentify = real(Aidentify(identvert,identvert)>0);
                    %All of the graphs covered by this proposition have K33
                    %as a minor. To find this minor, you need to find the
                    %edge contraction of the graph G on the edge in the K2.
                    if checkK33(Aidentify)
                        inerset = usegenset([2,1;1,2;4,0;0,4],n);
                        return
                    end
%                 end
            end
        end
    end
    % Find out if Q3Y\delta is a minor
    numedges = sum(sum(A))/2;
    if numedges>=12 && numedges<=15
```

```
%For this one I had to check that the characteristic polynomial of
%Q3Y\delta was unique when compared to graphs of with a similar
%number of vertices and edges. I used the characteristic
%polynomial, because it is provably difficult to determine whether
%two adjacency matrices are permutation similar.
G1005=[0 1 1 1 0 0 0;1 0 1 0 1 1 0;1 1 0 0 0 1 1;1 0 0 0 1 0 1;
    0 1 0 1 0 1 0;0 1 1 0 1 0 1;0 0 1 1 0 1 0];% Q3Y\delta
P = poly(G1005);
%If A does have the right number of vertices and edges then we need
%only compare the characteristic polynomial.
if numedges==12
    if norm(P-poly(A))<1e-10
        inerset = usegenset([0 3;1 2;2 1;3 0],n);
        return
    else
        inerset = usegenset([0 4;1 3;2 2;3 1;4 0],n);
        return
    end
end
%If it doesn't have the correct number of edges, but does have the
%correct number of vertices, then we need to delete edges until it
%has the right number. Then compare the characteristic polynomials
%again. I try to do this in a way that will cut down the number of
%checks that I do. For example, I won't erase edges that will lower
%the degree of a vertex to below what the minimum degree of
%Q3Y\delta.
[Y,I]=sort(sum(A),'descend');
```

```
        A = A(I,I);
        [checki checkj] = find(A);
        checkspots =checki< checkj;
        checki = checki(checkspots);
        checkj = checkj(checkspots);
        bad = find(Y==3,1);
        options = find(checkj==bad,1)-1;
        eraseedges = nchoosek(1:options,numedges-12);
        for i=1:size(eraseedges,1)
            newA = A;
            for j = 1:numedges-12
                points= [checki(eraseedges(i,j)) checkj(eraseedges(i,j))];
                newA(points,points)=0;
            end
            if norm(P-poly(newA))<1e-10
                inerset = usegenset([0 3;1 2;2 1;3 0],n);
                return
            end
        end
        %If it is not a minor, then I return the inertia set.
        inerset = usegenset([0 4;1 3;2 2;3 1;4 0],n);
        return
    end
end

function conn = checkcon(B)
%This function will check to see if the graph is connected. It will do so
```

```
%by checking the second smallest eigenvalue of the LaPlacian matrix. If the
%second smallest eigenvalue is 0, then it is disconnected. That is because
%the multiplicity of zero as an eigenvalue is equal to the number of
%components.
conn = 0;
L = diag(sum(B))-B;
lam = eig(L);
if lam(2)<eps
    conn=1;
end
end

function insets = addsets(G1,G2,n)
    %This is the Minkowski sum
    v1=size(G1,1);
    v2 = size(G2,1);
    h=[];
    repvec = v1*ones(v2,1);
    h(cumsum(repvec))=1;
    G2rep=G2(cumsum(h)-h+1,:);
    G1rep = repmat(G1,v2,1);
    insets = G1rep+G2rep;
    insets = unique(insets,'rows');
    insets = insets(sum(insets,2)<=n,:);
end

function inertiaset = usegenset(gensetmat,n)
```

```
    %Uses the generating set given and uses the northeast lemma to find the
    %inertia set. It takes each point in the generating set and gets all of
    %the points that satisfy the northeast lemma and then takes that union
    %of all of those points.
    inertiaset = [];
    for m = 1:size(gensetmat,1)
        x = gensetmat(m,1);
        y = gensetmat(m,2);
        xvec = (x:n)';
        yvec = (y:n)';
        yvec = repmat(yvec,n-x+1,1);
        xvec = sort(repmat(xvec,n-y+1,1));
        tempinerset = [xvec,yvec];
        %This makes sure that we don't go out too far. We can't have the
        %partial inertia add up to more than n.
        tempinerset = tempinerset(sum(tempinerset,2)<=n,:);
        inertiaset = [inertiaset;tempinerset];
    end
    inertiaset=unique(inertiaset,'rows');
end


function prop10= checkK33(GcheckK33)
    %Checks to see if K33 is a minor. It is helpful to know whether or not
    %K33 is a minor for one of the propositions used to determine the
    %inertia set. To find out whether or not K33 is a minor, we consider
    %the complement of the graph.
    [Scomp,Ccomp] = graphconncomp(sparse(ones(size(GcheckK33))...
```

```
        -eye(size(GcheckK33))-GcheckK33),'directed',false);
    switch Scomp
        case {4,5,6}
            %If the complement has 4 or more components, then that means
            %that K33 is a minor.
            prop10=true;
        case 3
            %If the complement has 3 components and at least one of those
            %components has three vertices, that means that K33 is a minor
            sizecomp = sort([sum(Ccomp==1) sum(Ccomp==2) sum(Ccomp==3)]);
            prop10 = sizecomp(3)==3;
        case 2
            %If the complement has 2 components, and both components have 3
            %vertices, then K33 is a minor.
            prop10=(sum(Ccomp==1)==3);
        otherwise
            prop10=false;
    end
end

function I = Trap(a,b)
n1 = min(a,b);
n2 = max(a,b);
gensettrap = [(0:n1)' (n1:-1:0)'];
I = usegenset(gensettrap,n2);
end
end
```

Bibliography

[1] F. Barioli and S. Fallat. On the minimum rank of the join of graphs and decomposable graphs. *Linear Algebra and Its Applications*, 421:252-263, 2007.

[2] W. Barrett, S. Butler, H. T. Hall, J. Sinkovic, W. So, C. Starr, and A. Yielding. Computing Inertia Sets Using Atoms. *Linear Algebra and Its Applications*, 436:4489-4502, 2012.

[3] W. Barrett, S. Gibelyou, M. Kempton, N. Malloy, C. Nelson, W. Sexton, and J. Sinkovic. The Inverse Eigenvalue and Inertia Problems for Minimum Rank Two Graphs. *Electronic Journal of Linear Algebra*, 22:389-418, 2011.

[4] W. Barrett, H. T. Hall, and H. van der Holst. The Inertia Set of the Join of Graphs. *Linear Algebra and Its Applications*, 434:2197-2203, 2011.

[5] W. Barrett, C. Jepsen, R. Lang, E. McHenry, C. Nelson, and K. Owens. Inertia Sets for Graphs on Six or Fewer Vertices. *Electronic Journal of Linear Algebra*, 20:53-78, 2010.

[6] W. Barrett, M. Kempton, N. Malloy, C. Nelson, W. Sexton, and J. Sinkovic. Decompositions of Minimum Rank Matrices. *Linear Algebra and Its Applications*, 438:3913-3948, 2013.