

Flag Algebra and Applications to Extremal Graph Theory

Mohammad Zubair Konchwalla

March 2020

Contents

1	Acknowledgements	3
2	Introduction	1
2.1	Definitions	1
2.2	Intuition	3
3	A more formal introduction	8
3.1	Types and Flags	8
3.2	Densities	9
3.3	The Semidefinite method	10
4	The Algebra	16
4.1	Multiplication and the Averaging operator	16
4.2	Limits and Positive Homomorphism	19
4.3	Duality and the Semidefinite problem	21
5	Applications to Graphs	27
5.1	Mantel's	27
5.1.1	Structure of the Extremal Graph	27

5.1.2	Minimum density of Triangles	29
5.2	K_4 -Free graphs	31
5.3	The Pentagon Problem	38
6	Appendix	43
6.1	Main Code	43
6.2	Code for K_4	57
6.2.1	Generating c_H terms: Python	57
6.2.2	Code for Semidefinite Optimization: Mathematica	60
6.3	Code for Pentagon Problem	61
6.3.1	Generating c_H terms: Python	61
6.3.2	Code for Semidefinite Optimization: Mathematica	68
6.4	Code for C7	70
6.4.1	Computing $c_H(\sigma_1, 5, P)$	70
6.4.2	c_H Inequalities	72

Chapter 1

Acknowledgements

I would like to thank Dr John Talbot for his supervision during this project.

Abstract

In [11], Razborov developed a systematic approach to the solving problems in extremal combinatorics. This method has made it possible to derive bounds for parameters in a partially automated manner. Flag algebras can be applied to a wide range of structures but this paper focuses on the application to extremal graph theory and solving Turán problems. This paper explains the intuition and theory behind the flag algebra method and present solutions to Turán problems using flag algebras.

Chapter 2

Introduction

Extremal graph theory is the field of mathematics that studies how global properties of graphs affects its local substructure. Some examples of problems include

- What is the maximum number of edges in a graph that does not contain a triangle? (*Mantel's Theorem*)
- What is the maximum number of pentagons in a graph that does not contain a triangle?

These problems are called Turán problems which focus on finding the extremal asymptotic density of a graph, A , within another graph, G given that G does not contain some forbidden subgraph. In this paper, we shall introduce some basic concepts of graph theory, explain the theory of the flag algebra method and show how the method can be utilised to solve a different problems in extremal graph theory. By the end of the paper, we shall show some examples on how this method can lead to new upper bounds for problems in extremal graph theory. We will also show how we can use these bounds to investigate the structure of the extremal graph given some forbidden subgraph.

Let's start with some basic definitions and some concepts to help build up our intuition.

2.1 Definitions

We will use the standard graph theory notation. A *graph* is defined to be a pair $G = (V(G), E(G))$ where $V(G)$ is the set of *vertices* and $E(G)$ is a set of 2-sets of $V(G)$. $E(G)$ is also called the set of edges. We define the *order* of a graph, G , to be $|V(G)|$ and we usually denote this with n when

dealing with unknowns. We define the *edge density* of G to be:

$$d(G) = \frac{|E(G)|}{\binom{n}{2}}.$$

If \mathcal{F} be a family of graphs, we say G is \mathcal{F} -free if G does not contain a subgraph isomorphic to any element of \mathcal{F} . We define the *Turán number* of \mathcal{F} to be:

$$ex(n, \mathcal{F}) = \max\{|E(G)| : G \text{ is } \mathcal{F}\text{-free and } |V(G)| = n\}.$$

Simply, it is the size of the largest possible graph, on n vertices, that is \mathcal{F} -free. But what happens if we want to calculate the density of edges in a graph given that it is \mathcal{F} -free? Furthermore, what happens if we take the limit of this as $n \rightarrow \infty$? We define the *Turán density* of \mathcal{F} to be:

$$\pi(\mathcal{F}) = \lim_{n \rightarrow \infty} \frac{ex(n, \mathcal{F})}{\binom{n}{2}}.$$

Lemma 2.1.1. $\pi(F)$ is well-defined for any graph F .

Proof. Consider a graph G that is F -free and with $ex(n, F)$ edges. Consider $\sum_{v \in V(G)} |E(G - v)|$.

For each vertex v , $G - v$ has at most $ex(n - 1, F)$ edges without containing a copy of F , so we have

$$\sum_{v \in V(G)} |E(G - v)| \leq n \, ex(n - 1, F).$$

Also, every edge in the summation is counted exactly $n - 2$ times. It is not counted when its v is one of the endpoints of the edge. Therefore

$$(n - 2) \, ex(n, F) = \sum_{v \in V(G)} |E(G - v)| \leq n \, ex(n - 1, F).$$

And after some simple manipulation, we can see

$$\frac{ex(n, F)}{\binom{n}{2}} \leq \frac{ex(n - 1, F)}{\binom{n-1}{2}}.$$

So we see that $\{\frac{ex(n, F)}{\binom{n}{2}}\}_{n=1}^{\infty}$ is a decreasing monotonic sequence. Trivially $\pi(F) \in [0, 1]$ and is bounded so a limit exists via monotonic convergence theorem. \square

These definitions are specific to edges in graphs, but we can generalise these definitions to structures other than edges.

Let A be a given graph on k vertices. Let $C_A(G)$ be the set of all k -element subsets of $V(G)$ that induce a subgraph isomorphic to A . We define the *density of A in G* to be:

$$d(A; G) = \frac{|C_A(G)|}{\binom{n}{k}}.$$

This is just the probability that if we select k vertices from $V(G)$ that these k vertices form an induced subgraph of G that is isomorphic to A . Therefore, $d(A; G) \in [0, 1]$. If $A = \text{edge}$, this is the same as our definition for $d(G)$.

We define the *Turán number of \mathcal{F}* to be:

$$ex_A(n, \mathcal{F}) = \max\{|C_A(G)| : G \text{ is } \mathcal{F}\text{-free and } |V(G)| = n\}.$$

We define the *Turán density of \mathcal{F}* to be:

$$\pi_A(\mathcal{F}) = \lim_{n \rightarrow \infty} \frac{ex_A(n, \mathcal{F})}{\binom{n}{k}}.$$

By a similar argument to Lemma 2.1.1, we can show that $\pi_A(\mathcal{F})$ exists and lies in $[0, 1]$.

2.2 Intuition

In this section, we are going to introduce the basic concepts behind flag algebras which provides us a systematic approach to finding counting arguments. In chapters 3 and 4, we are going to define them more rigorously, but for the moment, let us focus on the ideas behind the concept. We are going to focus on the Turán density associated to some fixed graph A , i.e. $\pi_A(\mathcal{F})$. Let \mathcal{F} be a family of forbidden graphs whose Turán density we wish to compute.

Let us fix a graph G with a large number of vertices and fix some small $l \geq 2$. Instead of counting the appearances of A in G , we can count them in each possible graph of H on l vertices and count the appearances of H in G .

$$d(A; G) = \sum_{|V(H)|=l} d(A; H)d(H; G).$$

Let \mathcal{F}_l^\emptyset be a family of all \mathcal{F} -free graphs on l vertices up to isomorphism. If G is an \mathcal{F} -free graph, then for $H \notin \mathcal{F}_l^\emptyset$, we have $d(H; G) = 0$. Therefore our summation becomes

$$d(A; G) = \sum_{H \in \mathcal{F}_l^\emptyset} d(A; H)d(H; G)$$

and then we can take the simple upper bound to get

$$d(A; G) \leq \max_{H \in \mathcal{F}_l^\emptyset} d(A; H).$$

For sufficiently small l , we can explicitly determine $d(A; H)$ for all $H \in \mathcal{F}_l^\emptyset$ by computer search but this bound is usually poor.

The idea behind the flag algebra method is to generate inequalities on $d(H; G)$ to improve this upper bound. If we have a linear inequality of the form

$$\sum_{H \in \mathcal{F}_l^\emptyset} c_H d(H; G) \geq 0$$

then we can combine the two summations to get

$$d(A; G) = \sum_{H \in \mathcal{F}_l^\emptyset} d(A; H) + c_H d(H; G) \leq \max_{H \in \mathcal{F}_l^\emptyset} (d(A; H) + c_H).$$

This will improve the bound if some of the c_H coefficients are negative. We can loosen the forms of the inequalities by taking inequalities of the form.

$$\sum_{H \in \mathcal{F}_l^\emptyset} c_H d(H; G) + o(1) \geq 0$$

with $o(1)$ being taken with respect to $|V(G)|$. Therefore, we get

$$d(A; G) \leq \max_{H \in \mathcal{F}_l^\emptyset} (d(A; H) + c_H + o(1))$$

$$\pi_A(\mathcal{F}) \leq \max_{H \in \mathcal{F}_l^\emptyset} (d(A; H) + c_H).$$

Let's return to one of the questions mentioned at the beginning of the paper. What is the maximum number of edges in a graph that does not contain a triangle? In other words, what is the value of $ex(n, \triangle)$? In 1907, Mantel discovered the solution is $\frac{n^2}{4}$. In this paper, we are going to present a proof of this method using the flag algebra method.

An equivalent form of Mantel's theorem is: If G is a triangle-free graph, then $d(G) \leq \frac{1}{2} + o(1)$.

We are going to consider the case where $l = 3$ and $\mathcal{F} = \{ \triangle \}$. The graphs on 3 vertices up to isomorphism are $\{ \circ \circ \circ, \circ - \circ, \circ \diagup \circ, \triangle \}$. Therefore, $\mathcal{F}_3^\emptyset = \{ \circ \circ \circ, \circ - \circ, \circ \diagup \circ \}$

$$\begin{aligned}
d(G) &= d(\circ \overset{\circ}{\circ} \circ) d(\circ \overset{\circ}{\circ} \circ; G) + d(\circ \overset{\circ}{\circ} \circ) d(\circ \overset{\circ}{\circ} \circ; G) + d(\circ \overset{\circ}{\circ} \circ) d(\circ \overset{\circ}{\circ} \circ; G) \\
d(G) &= \frac{1}{3} d(\circ \overset{\circ}{\circ} \circ; G) + \frac{2}{3} d(\circ \overset{\circ}{\circ} \circ; G).
\end{aligned} \tag{2.1}$$

Now, we shall fix a vertex and denote the fixed vertex with \bullet . We define the density of a graph with a fixed vertex as the probability of finding an induced copy of this graph with one vertex fixed. We denote G^\bullet to be a graph G with some vertex fixed. i.e. $d(\circ \overset{\circ}{\circ} \circ; G^\bullet)$ is equal to the degree of \bullet divided by $n - 1$. Consider

$$\left(d(\circ \overset{\circ}{\circ} \circ; G^\bullet) - d(\bullet \overset{\circ}{\circ} \circ; G^\bullet) \right)^2 \geq 0 \tag{2.2}$$

$$d(\circ \overset{\circ}{\circ} \circ; G^\bullet)^2 - 2d(\circ \overset{\circ}{\circ} \circ; G^\bullet)d(\bullet \overset{\circ}{\circ} \circ; G^\bullet) + d(\bullet \overset{\circ}{\circ} \circ; G^\bullet)^2 \geq 0. \tag{2.3}$$

We can use a combinatorial argument to create a few identities to simplify the expression (2.3). So what does $d(\circ \overset{\circ}{\circ} \circ; G^\bullet)^2 = d(\circ \overset{\circ}{\circ} \circ; G^\bullet)d(\circ \overset{\circ}{\circ} \circ; G^\bullet)$ actually mean? It is the probability that we have randomly chosen two vertices (independently and with replacement) which are both adjacent to vertex \bullet . If we consider choosing random vertices without replacement, we end up with two possible situations. Either there is an edge between the two randomly chosen vertices or there is no edge between them. Therefore,

$$d(\circ \overset{\circ}{\circ} \circ; G^\bullet)d(\circ \overset{\circ}{\circ} \circ; G^\bullet) - P(\text{Same vertex selected twice}) = d(\circ \overset{\circ}{\circ} \circ; G^\bullet) + d(\circ \overset{\circ}{\circ} \circ; G^\bullet).$$

Note: The probability of the same vertex being selected twice tends to 0 as $|V(G)| \rightarrow \infty$ and is equal to $o(1)$. Also G is triangle-free so $d(\circ \overset{\circ}{\circ} \circ; G^\bullet) = 0$. Therefore

$$d(\circ \overset{\circ}{\circ} \circ; G^\bullet)d(\circ \overset{\circ}{\circ} \circ; G^\bullet) = d(\circ \overset{\circ}{\circ} \circ; G^\bullet) + d(\circ \overset{\circ}{\circ} \circ; G^\bullet) + o(1) = d(\circ \overset{\circ}{\circ} \circ; G^\bullet) + o(1).$$

In similar fashion, we can show

$$d(\bullet \overset{\circ}{\circ} \circ; G^\bullet)^2 = d(\bullet \overset{\circ}{\circ} \circ; G^\bullet) + d(\bullet \overset{\circ}{\circ} \circ; G^\bullet) + o(1)$$

and

$$d(\circ \overset{\circ}{\circ} \circ; G^\bullet)d(\bullet \overset{\circ}{\circ} \circ; G^\bullet) = \frac{1}{2} \left(d(\circ \overset{\circ}{\circ} \circ; G^\bullet) + d(\bullet \overset{\circ}{\circ} \circ; G^\bullet) \right) + o(1).$$

The coefficient $\frac{1}{2}$ appears due to the fact that we have an ordered product and an unordered result. On the right hand side, we are selecting two vertices at once. There is one edge between our fixed

vertex and our two randomly chosen vertices. The choice of vertex that our fixed vertex connects to is arbitrary and the result is unordered. Whilst on the left hand side, we are fixing one vertex and then choosing a random vertex two independent times. The first vertex must be adjacent to \bullet and the second vertex must not. Therefore, the order in which we select the vertices matters, and consequently, we divide the unordered result by 2.

So let us substitute our three identities into (2.3) and we get

$$d(\text{↗↘}; G^\bullet) - d(\text{↘↗}; G^\bullet) - d(\text{↗↖}; G^\bullet) + d(\text{↖↗}; G^\bullet) + d(\text{↖↘}; G^\bullet) + o(1) \geq 0. \quad (2.4)$$

Now, we can average over all possible choices of the fixed vertex. For example, averaging $d(\text{↖↗}; G^\bullet)$ over all possible vertices gives us $\frac{1}{3}d(\text{↖↗}; G)$ asymptotically. This is because there are three possibilities of fixing a vertex in ↖↗ and only one of those possibilities will result in graph ↖↗ . Thus, on average, we get the result.

So after averaging (2.4), we get

$$\frac{1}{3}d(\text{↗↘}; G) - \frac{2}{3}d(\text{↗↘}; G) - \frac{2}{3}d(\text{↖↗}; G) + \frac{1}{3}d(\text{↖↗}; G) + d(\text{↖↘}; G) + o(1) \geq 0.$$

After collecting the terms and dividing by 2, we get

$$\frac{1}{2}d(\text{↖↘}; G) - \frac{1}{6}d(\text{↖↗}; G) - \frac{1}{6}d(\text{↗↘}; G) + o(1) \geq 0. \quad (2.5)$$

So lets combine this with equation (2.1)

$$d(G) = \frac{1}{3}d(\text{↖↗}; G) + \frac{2}{3}d(\text{↗↘}; G) \quad (2.1)$$

and we get

$$d(G) \leq \frac{1}{2}d(\text{↖↘}; G) + \frac{1}{6}d(\text{↖↗}; G) + \frac{1}{2}d(\text{↗↘}; G) + o(1) \leq \frac{1}{2} + o(1).$$

Thus, we have arrived at the desired inequality. \square

In our proof, we began with some non-negative inequality of a labelled graph and used identities to manipulate the multiplication of densities into a linear combination of the density of larger graphs.

We then averaged the labelled graphs over the choices of fixed vertices of G to get inequalities of the following form

$$\sum_{H \in \mathcal{F}_l^\emptyset} c_H d(H; G) + o(1) \geq 0 \quad (2.6)$$

But how do we get the initial non-negative inequality (2.2) and how do we extend this to more complicated graphs?

We can manipulate our initial equation in the form of (2.6). Then we can rearrange this as a product of vectors containing the densities of graphs and a Symmetric Positive Semidefinite Matrix. If Q is an *Symmetric Positive Semidefinite Matrix*, then $vQv^T \geq 0$ for any vector $v \in \mathbb{R}^n$. In our previous example, we have

$$\begin{pmatrix} d(\bullet^\circ; G^\bullet) \\ d(\nearrow^\circ; G^\bullet) \end{pmatrix} \begin{pmatrix} q_{11} & q_{12} \\ q_{12} & q_{22} \end{pmatrix} \begin{pmatrix} d(\bullet^\circ; G^\bullet) \\ d(\nearrow^\circ; G^\bullet) \end{pmatrix}^T \geq 0.$$

After expanding through and averaging over the possible choices of the fixed vertex \bullet , we will eventually arrive at an equation of the form of (2.6) which is in terms of q_{11} , q_{12} , q_{22} . We can combine this with our equation for $d(A; G)$ and minimize our upper bound. Since our matrix of variables is symmetric positive semidefinite, we have a semidefinite programming problem which we are going to define later. This allows us to use semidefinite programming to optimize these coefficients.

In the next chapter, we shall formally introduce these ideas and show how this method can be generalised to graphs of higher order and more complicated labelling.

Chapter 3

A more formal introduction

In this chapter, we are going to develop the Razborov's theory of flag algebra. We are going to develop the approach specifically to the Turán problem but this method can be used to solve problems in a wider range of problems such Ramsey Theory or Permutations. Flag algebras also work on additional structures such as directed graphs, hyper-graphs but let us begin with some definitions.

3.1 Types and Flags

For an integer $s \geq 1$, we denote $[s] = \{1, 2, \dots, s\}$. We define a *type* σ to be an \mathcal{F} -free graph on s vertices s.t $V(\sigma) = [s]$. In other words, it is an \mathcal{F} -free graph with vertices labelled $1, 2, \dots, s$. We denote an empty type ($s = 0$) by \emptyset .

Let σ be a type on s vertices and let F be a graph on at least s vertices. An *embedding of σ* is an injective function $\theta : [s] \rightarrow V(F)$ that defines an isomorphism between σ and the subgraph F induced by the Image of θ , $\text{Im } \theta$.

A σ -*flag* is a pair (F, θ) where F is an \mathcal{F} -free graph and θ is an embedding of σ into F . Essentially, σ -flag is a partially labelled graph that is \mathcal{F} -free and whose labelled vertices induce a copy of σ . When the embedding itself is not important, we will ignore it in notation and simply denote it as the σ -*flag* F . Also, we often denote the order of the graph $|F|$ to be $|V(F)|$.

Note: A \emptyset -flag is just an \mathcal{F} -free graph. Additionally, any type σ of size s can be seen as the σ -flag (σ, θ) where θ is the identity mapping on $[s]$.

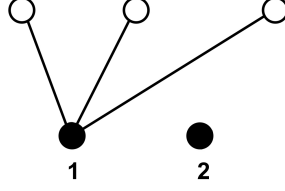


Figure 3.1: An example of a flag with $s = 2$ and $l = 5$

Isomorphism between flags are defined similarly to graphs, but we must add the extra condition that the labels should now be preserved by the bijection. More formally, σ -flags (F, θ) and (G, η) are isomorphic if there exists a graph isomorphism $\rho : V(F) \rightarrow V(G)$ such that $\rho(\theta(i)) = \eta(i)$ for $i = 1, 2, \dots, |\sigma|$.

We denote $(F, \theta) \simeq (G, \eta)$ when (F, θ) and (G, η) are isomorphic or simply, $F \simeq G$ when the embeddings are not important.

For $n \geq |\sigma|$, we define \mathcal{F}_n^σ to be the set of all σ -flags of size n , up to isomorphism. We define \mathcal{F}^σ to be the set of all σ -flags. A type σ is *degenerate* if \mathcal{F}^σ is finite. If σ is *non-degenerate*, then $\mathcal{F}_n^\sigma \neq \emptyset$ for $n \geq |\sigma|$.

We can show an example of a trivial degenerate case. Take \mathcal{F} to be the set of graphs with 1000 vertices containing at least one triangle and if we consider $\sigma = \triangle$, then there are no \triangle -flags of size ≥ 1000 . But we are only going to focus on non-degenerate types as they are more interesting.

3.2 Densities

We can extend the definition of densities given in the introduction to work on σ -flags. Firstly, we say σ -flags F_1, \dots, F_t fit in σ -flag G if

$$|G| - |\sigma| \geq (|F_1| - |\sigma|) + \dots + (|F_t| - |\sigma|).$$

If σ -flags F_1, \dots, F_t fit in σ -flag (G, θ) , we define the *density of σ -flags F_1, \dots, F_t in σ -flag G* , $d(F_1, \dots, F_t; G, \theta)$ to be, given that we have chosen pairwise-disjoint sets $U_1, U_2, \dots, U_t \subseteq V(G) \setminus \text{Im}(\theta)$ of unlabeled vertices with $|U_i| = |F_i| - |\sigma|$ uniformly at random, the probability that the σ -flag $G(U_i \cup \text{Im}(\theta), \theta)$ is going to be isomorphic to F_i . For \emptyset -flags and $t=1$, this coincides with the standard definition of density of graphs.

For example:

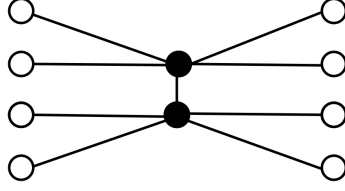


Figure 3.2: A graph of order $l = 10$ contains two graphs on $m = 6$ vertices which overlap exactly on $s = 2$ vertices.

$$d(\bullet \circ ; \square) = \frac{2}{3} \quad d(\bullet \circ, \bullet \circ ; \square) = \frac{1}{3}$$

We can extend these definitions to the define the density of σ -flags F_1, \dots, F_t in a graph G . In this situation, we must specify an embedding of σ , $\theta \in \Theta_G$ where Θ_G is the set of all injections from $[[\sigma]]$ to $V(G)$. The embedding maps G into a σ -flag, G' and we can apply the same definition as above.

Say $|F| \leq n \leq |G|$ and F and G are both σ -flags. Instead of calculating $d(F; G)$ directly, we may alternatively try to embed F into σ -flag F' of order n and then embed F' into G and calculate the density through this.

$$d(F; G, \theta) = \sum_{F' \in \mathcal{F}_n^\sigma} d(F; F', \theta) d(F'; G, \theta).$$

We can generalise this identity, giving us the *chain rule*. [4]

Theorem 3.2.1. *If F_1, \dots, F_t and G are σ -flags such that F_1, \dots, F_t fit in G , then for every $1 \leq k \leq t$ and for every n such that F_1, \dots, F_k fit in a σ -flag of order n and F_{k+1}, \dots, F_t and the same σ -flag of order n fit in G , then the following identity holds.*

$$d(F_1, \dots, F_t, G; \theta) = \sum_{F \in \mathcal{F}_n^\sigma} d(F_1, \dots, F_k, F; \theta) d(F, F_{k+1}, \dots, F_t; G, \theta).$$

3.3 The Semidefinite method

Let G be a large graph and let us fix a type σ on s vertices. Let l and s be integers such that $l > s$ and $m \leq \frac{l+s}{2}$. The bound on m ensures that a graph on l vertices can contain two graphs on m vertices overlapping on exactly s vertices.

Let us consider σ -flags $F_i \in \mathcal{F}_m^\sigma$. Let a_i be real coefficients for these flags and fix a $\theta : [s] \rightarrow V(G)$ and consider the following inequality

$$\left(\sum_{F_i \in \mathcal{F}_m^\sigma} a_i d(F_i; G, \theta) \right)^2 \geq 0.$$

After expanding the square, we get

$$\sum_{F_i, F_j \in \mathcal{F}_m^\sigma} a_i a_j d(F_i; G, \theta) d(F_j; G, \theta) \geq 0$$

Using the following lemma from [6],

Lemma 3.3.1. *Given a graph G , for any $F_1, F_2 \in \mathcal{F}_n$ and $\theta \in \Theta_G$, then the following results holds*

$$d(F_1; G, \theta) d(F_2; G, \theta) = d(F_1, F_2; G, \theta) + o(1)$$

where $o(1) \rightarrow 0$ as $|V(G)| \rightarrow \infty$.

we get

$$\sum_{F_i, F_j \in \mathcal{F}_m^\sigma} a_i a_j d(F_i, F_j; G, \theta) + o(1) \geq 0.$$

We define $q_{ij} = a_i a_j$ and if we assume that matrix $Q = (q_{ij})_{1 \leq i, j \leq |\mathcal{F}_m^\sigma|}$ is symmetric positive semidefinite, then we get

$$\sum_{F_i, F_j \in \mathcal{F}_m^\sigma} q_{ij} d(F_i, F_j; G, \theta) + o(1) \geq 0.$$

We can then take an average of the uniformly random choices of $\theta \in \Theta_G$. Using the linearity of expectation, we obtain

$$\sum_{F_i, F_j \in \mathcal{F}_m^\sigma} q_{ij} \mathbb{E}_{\theta \in \Theta_G} [d(F_i, F_j; G, \theta)] + o(1) \geq 0.$$

Now, we can use the chain rule to calculate the expectation over all subgraphs of G on l vertices. Let H be an arbitrary subgraph of order l and let Θ_H to be the set of all injective mappings $\theta : [s] \rightarrow V(H)$. Therefore, we get

$$\mathbb{E}_{\theta \in \Theta_G}[d(F_i, F_j; G, \theta)] = \sum_{H \in \mathcal{F}_l^\emptyset} \mathbb{E}_{\theta \in \Theta_H}[d(F_i, F_j; H, \theta)]d(H; G).$$

So substituting this into the expression above it, we get

$$\sum_{H \in \mathcal{F}_l^\emptyset} \sum_{F_i, F_j \in \mathcal{F}_m^\sigma} q_{ij} \mathbb{E}_{\theta \in \Theta_H}[d(F_i, F_j; H, \theta)]d(H; G) + o(1) \geq 0.$$

We define $c_H(\sigma, m, Q) = \sum_{F_i, F_j \in \mathcal{F}_m^\sigma} q_{ij} \mathbb{E}_{\theta \in \Theta_H}[d(F_i, F_j; H, \theta)]$ and therefore, we have

$$\sum_{H \in \mathcal{F}_l^\emptyset} c_H(\sigma, m, Q)d(H; G) + o(1) \geq 0$$

which is an inequality of form (2.6) which is exactly what we were looking for! Additionally, we can take k choices of (σ_i, m_i, Q_i) where each σ_i is a type with the bound $m_i \leq \frac{l+|\sigma_i|}{2}$ with $m_i, l, \sigma_i \in \mathbb{Z}$ and Q_i is a symmetric positive semidefinite matrix of dimensions $|\mathcal{F}_{m_i}^{\sigma_i}| \times |\mathcal{F}_{m_i}^{\sigma_i}|$. Therefore, we can redefine our c_H to be

$$c_H = \sum_{i=1}^k c_H(\sigma_i, m_i, Q_i).$$

And so we have arrived at a stronger inequality

$$\sum_{H \in \mathcal{F}_l^\emptyset} c_H d(H; G) + o(1) \geq 0.$$

Now, we can repeat our work in chapter 2 by combining our expression with

$$d(A; G) = \sum_{H \in \mathcal{F}_l^\emptyset} d(A; H)d(H; G)$$

$$d(A; G) \leq \sum_{H \in \mathcal{F}_l^\emptyset} (d(A; H) + c_H)d(H; G) + o(1).$$

Thus,

$$\pi_A(\mathcal{F}) \leq \max_{H \in \mathcal{F}_l^\emptyset} (d(A; H) + c_H).$$

Once again, this works because some of the c_H terms can be negative. This bound has the potential to be significantly better than the averaging bound. Now we just need to minimise $\max_{H \in \mathcal{F}_l^\emptyset} (d(A; H) + c_H)$ with respect to the coefficients of the symmetric positive semidefinite matrix, Q_i which minimizes $c_H(\sigma_i, m_i, Q_i)$ and therefore c_H . This is a semidefinite programming problem [1] which is defined as below.




$$\begin{aligned} \min \quad & c^T x \\ \text{subject to} \quad & F(x) \geq 0 \\ \text{where} \quad & F(x) = F_0 + \sum_{i=1}^m x_i F_i \end{aligned}$$

where $c \in \mathbb{R}^m$ and the symmetric matrices $F_0, F_1, \dots, F_m \in \mathbb{R}^{n \times n}$ and $F(x) \geq 0$ means that $F(x)$ is positive semidefinite. Fundamentally, we need a finite number of linear constraints in terms of our variables and for our matrix of variables to be symmetric positive semidefinite.

So we can rephrase our problem as

$$\begin{aligned} \min \quad & \lambda \\ \text{subject to} \quad & \lambda \geq d(A; H) + c_H \text{ for every } H \in \mathcal{F}_l^\emptyset \text{ where } l > 0 \text{ is fixed} \\ \text{where} \quad & c_H = \sum_{F_i, F_j \in \mathcal{F}_m^\sigma} q_{ij} \mathbb{E}_{\theta \in \Theta_H} [d(F_i, F_j, H, \theta)] \\ \text{and} \quad & Q \text{ is a } |\mathcal{F}_m^\sigma| \times |\mathcal{F}_m^\sigma| \text{ positive semidefinite matrix.} \end{aligned}$$

We are going to return to Mantel's theorem to show how we can compute the values for c_H via the expectation. Let $l = 3$ and $\mathcal{F} = \{\triangle\}$. So we have, $\mathcal{F}_3^\emptyset = \{\circ \circ \circ, \circ \text{---} \circ, \circ \text{---} \circ\}$. We can then compute $d(H)$ for $H \in \mathcal{F}_3^\emptyset$.

H	$d(H)$
	0
	1/3
	2/3

We are going to take the type consisting of a single labelled vertex \bullet and we are going to take flags of order 2 which are flags: \nearrow, \bullet, \circ . We can now calculate the expectation and the values are computed in the table below.

$\mathbb{E}_{\theta \in \Theta_H}[d(F_i, F_j; H, \theta)]$	$\circ \circ$	$\circ \text{---} \circ$	$\nwarrow \nearrow$
\bullet, \bullet	1	1/3	0
\nearrow, \circ	0	2/3	2/3
\nearrow, \nearrow	0	0	1/3

We calculate these values in the table by considering all the possible embeddings of the σ -flag into the graph H and calculating $d(F_i, F_j; H, \theta)$ by taking the average of it. For example, with graph $\nwarrow \nearrow$, we can place the label \bullet in three possible ways. $\nwarrow \bullet \nearrow, \nwarrow \nearrow \bullet, \bullet \nwarrow \nearrow$. We must then randomly select two vertices and compare if they are isomorphic to our flags. Since there are only two remaining vertices, we are not left with much choice. For flags, \bullet, \bullet, \circ , we can see that out of our three options of labelling, none of the embeddings allow us to induce isomorphic flags so we get

$$\mathbb{E}_{\theta \in \Theta_H}[d(\bullet, \bullet, \circ; \nwarrow \nearrow, \theta)] = 0.$$

So now we can take our variables q_{11}, q_{12}, q_{22} and create a symmetric positive semidefinite matrix,

$$Q = \begin{pmatrix} q_{11} & q_{12} \\ q_{12} & q_{22} \end{pmatrix}$$

to find our values for $c_H = \sum_{F_i, F_j \in \mathcal{F}_m^\sigma} q_{ij} \mathbb{E}_{\theta \in \Theta_H}[d(F_i, F_j; H, \theta)]$ and use this to minimise the expression

$$\pi_A(\mathcal{F}) \leq \max_{H \in \mathcal{F}_3^\emptyset} (d(A; H) + c_H).$$

In our example, $\mathcal{F}_3^\emptyset = \{ \circ \circ, \circ \text{---} \circ, \nwarrow \nearrow \}$ so we get

$$\min_{Q \geq 0} \max \left\{ q_{11}, \frac{1}{3} + \frac{1}{3}q_{11} + \frac{2}{3}q_{12}, \frac{2}{3} + \frac{2}{3}q_{12} + \frac{1}{3}q_{22} \right\}$$

which is equivalent to

$$\begin{aligned}
& \min \quad \lambda \\
& \text{subject to} \quad \lambda \geq q_{11} \\
& \quad \text{and} \quad \lambda \geq \frac{1}{3} + \frac{1}{3}q_{11} + \frac{2}{3}q_{12} \\
& \quad \text{and} \quad \lambda \geq \frac{2}{3} + \frac{2}{3}q_{12} + \frac{1}{3}q_{22} \\
& \quad \text{and} \quad Q \text{ is a positive semidefinite matrix.}
\end{aligned} \tag{3.1}$$

Using semidefinite programming, we get

$$Q = \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

and therefore

$$\pi(\triangle) \leq \max\left\{\frac{1}{2}, \frac{1}{6}, \frac{1}{2}\right\} = \frac{1}{2}.$$

□

Chapter 4

The Algebra

In this chapter, we are going to develop the formal definitions of the flag algebra operations in an algebraic setting.

In the previous chapter, we defined and used a type and a flag associated with a family of \mathcal{F} -free graphs. We can extend these definitions for other kinds of structures such as directed graphs or hyper graphs. Razborov presents a more theoretical approach in [11], but in this paper, we are going to present this theory for graphs for simplicity.

4.1 Multiplication and the Averaging operator

The aim for the flag algebra method to extremal problems is to generate inequalities of the form

$$\sum_{F_i \in \mathcal{F}_l^\sigma} b_i d(F_i; G) + o(1)$$

which are valid for every flag $G \in \mathcal{F}^\sigma$. But we are going to consider it slightly differently. We define $\mathbb{R}\mathcal{F}^\sigma$ to be the space of all formal real linear combinations of σ -flags, i.e. a free vector space over the reals generated by σ flag.

We can think of any flag $G \in \mathcal{F}^\sigma$ as acting on $\mathbb{R}\mathcal{F}^\sigma$ through the mapping $\sum b_i F_i \rightarrow \sum b_i d(F_i; G)$. So any \mathcal{F} -free graph is a mapping on $\mathbb{R}\mathcal{F}^\sigma$.

From the chain rule, we know

$$d(\tilde{F}; G) = \sum_{F \in \mathcal{F}_l^\sigma} d(\tilde{F}; F) d(F; G).$$

Therefore, any linear combination of the following form

$$\tilde{F} - \sum_{F \in \mathcal{F}_l^\sigma} d(\tilde{F}; F) F$$

is going to be mapped to zero by G , so it makes intuitive sense that we should try to factor it out. This will allow us to define a quotient space where it is possible to create an algebra with a multiplication operation.

We define \mathcal{K}^σ to be the linear subspace of $\mathbb{R}\mathcal{F}^\sigma$ generated by all elements of the form

$$\tilde{F} - \sum_{F \in \mathcal{F}_l^\sigma} d(\tilde{F}; F) F$$

with $\tilde{F} \in \mathcal{F}_{\tilde{l}}^\sigma$ and $s \leq \tilde{l} \leq l$. We define $\mathcal{A}^\sigma = \mathbb{R}\mathcal{F}^\sigma / \mathcal{K}^\sigma$ to be the desired quotient space. In [11], Razborov proves that the multiplication operation is well-defined in \mathcal{A}^σ with identity element σ and that multiplication is not defined in $\mathbb{R}\mathcal{F}^\sigma$.

For any $F_1 \in \mathcal{F}_{l_1}^\sigma$ and $F_2 \in \mathcal{F}_{l_2}^\sigma$ with any arbitrary $l \geq l_1 + l_2 - s$, we define

$$F_1 \cdot F_2 = \sum_{F \in \mathcal{F}_l^\sigma} d(F_1, F_2; F) F$$

and we expand it through linearity. This doesn't depend on our choice of l and this gives the structure of a commutative algebra on \mathcal{A}^σ .

Informally, we know that

$$d(F_1, F_2; G) = d(F_1; G) d(F_2; G) + o(1)$$

so for large $|V(G)|$ we can view the mapping G as an approximate homomorphism from \mathcal{A}^σ to \mathbb{R} .

We can return to our earlier work of Mantel's theorem to understand the intuition behind these definitions. Let us look at the following examples. $\mathcal{F}_3^\bullet = \{ \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix}, \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix} \circ, \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix} \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix}, \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix} \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix} \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix}, \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix} \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix} \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix} \}$

$$\begin{smallmatrix} \bullet \\ \circ \end{smallmatrix} \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix} = \frac{1}{3} \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix} \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix} \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix} + \frac{2}{3} \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix} \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix} \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix} + \frac{1}{3} \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix} \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix} \begin{smallmatrix} \bullet \\ \circ \end{smallmatrix}$$

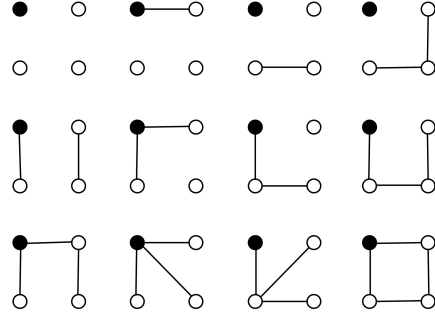


Figure 4.1: \mathcal{F}_4^\bullet

which implies that

$$\text{flag} - \frac{1}{3} \text{flag} - \frac{2}{3} \text{flag} - \frac{1}{3} \text{flag} \in \mathcal{K}^\emptyset.$$

$$\text{flag} \cdot \text{flag} = \sum_{F \in \mathcal{F}_3^\bullet} d(\text{flag}, \text{flag}, F) F = \text{flag}$$

Furthermore, consider that the two \bullet -flags, flag, flag also fit in \mathcal{F}_4^\bullet which consists of the flags in Figure 4.1. So

$$\text{flag} \cdot \text{flag} = \sum_{F \in \mathcal{F}_4^\bullet} d(\text{flag}, \text{flag}, F) F = \frac{1}{3} \text{flag} + \frac{1}{3} \text{flag} + \text{flag} + \frac{1}{3} \text{flag}.$$

Often in the flag algebra method, we are showing that a specific inequality holds for a specific labeling of vertices and then we average over all possible choices of vertices to get an inequality valid for densities that do not depend on specific vertices.

Formally, we define this operation as a linear operator from \mathcal{A}^σ to \mathcal{A}^\emptyset . (\mathcal{A}^\emptyset is the quotient space of unlabelled graphs). We define the *averaging operator* of $F \in \mathcal{F}^\sigma$ to be

$$\llbracket F \rrbracket = q_\sigma(F) \cdot F'$$

where F' is the unlabeled version of F and $q_\sigma(F)$ is the probability that an injective mapping $\theta : \llbracket \sigma \rrbracket \rightarrow V(F')$ (chosen uniformly at random) defines an induced embedding of θ in F' with the resulting σ -flag being isomorphic to F . We then extend the operator $\llbracket \cdot \rrbracket$ from \mathcal{F}^σ to \mathcal{A}^σ via linearity.

Examples:

$$\llbracket \bullet \circ \rrbracket = \circ \circ, \quad \llbracket \bullet \circ \circ \rrbracket = \frac{1}{3} \circ \circ \circ$$

Say D is a path on 3 edges, P_3 , with the first vertex labeled by 1 and the adjacent vertex labeled 2. Then $\llbracket D \rrbracket = \frac{1}{6} P_3$ as the labels can only be placed in 2 ways to maintain the label preserving requirement of isomorphic flags.

4.2 Limits and Positive Homomorphism

Given a σ -flag G , we can identify a mapping $d : \mathbb{R}^{\mathcal{F}^\sigma} \rightarrow \mathbb{R}$ as defined before. However, we can identify this mapping with infinite vector of densities $(d(F; G))_{F \in \mathcal{F}} \in [0, 1]^{\mathcal{F}}$. This is a compact space via Tychonoff's theorem.

Theorem 4.2.1. *Tychonoff's Theorem [3]: If (X_α, τ_α) are compact topological spaces for each $\alpha \in A$, then so is $X = \prod_{\alpha \in A} X_\alpha$ (endowed with the product topology).*

Since the space is compact, any sequence contains a convergent subsequence. In this context, we define *convergence* by saying sequence of graphs in $(G_k)_{k \geq 0}$ in \mathcal{F} is convergent if for every flag $F \in \mathcal{F}^\sigma$, $\lim_{k \rightarrow \infty} d(F; G_k)$ exists. We then can define a *limit functional*, $\phi : \mathbb{R}^{\mathcal{F}^\sigma} \rightarrow \mathbb{R}$, if ϕ is linear and there exists a convergent sequence $(G_k)_{k \geq 0}$ of flags in \mathcal{F} such that $\phi(F) = \lim_{k \rightarrow \infty} d(A; G_k)$, for all $F \in \mathcal{F}^\sigma$.

We define $\text{Hom}^+(\mathcal{A}^\sigma, \mathbb{R})$ to be the set of all homomorphisms, ϕ , from \mathcal{A}^σ to \mathbb{R} such that $\phi(F) \geq 0$ for every $F \in \mathcal{F}^\sigma$. So any σ -flag can be identified with a positive homomorphism. In [11], Razborov has shown that if $\phi : \mathcal{A}^\sigma \rightarrow \mathbb{R}$ is a limit functional associated with a sequence of graphs, $(G_k)_{k \geq 0}$, then $\phi \in \text{Hom}^+(\mathcal{A}^\sigma, \mathbb{R})$, hence the notation of using ϕ twice. Furthermore, Razborov has shown

Theorem 4.2.2. *For any convergent sequence of σ -flags in \mathcal{F}^σ , the limit is in $\text{Hom}^+(\mathcal{A}^\sigma, \mathbb{R})$. Conversely, any element of $\text{Hom}^+(\mathcal{A}^\sigma, \mathbb{R})$ is the limit of a sequence of σ -flags.*

This implies that we have a correspondence between the inequalities $\sum_{F_i \in \mathcal{F}_l^\sigma} b_i d(F_i; G) + o(1) \geq 0$ and $\phi(\sum_{F_i \in \mathcal{F}_l^\sigma} b_i F_i) \geq 0$ for $\phi \in \text{Hom}^+(\mathcal{A}^\sigma, \mathbb{R})$. This allows us to rephrase the Turán density in flag algebra terms. We can redefine the Turán density as

$$\pi_A(\mathcal{F}) = \sup_{(G_k)_{k \geq 0}} \lim_{k \rightarrow \infty} \sup d(A; G_k).$$

where the supremum is taken over all sequences $(G_k)_{k \geq 0}$ of \mathcal{F} -free graphs that are increasing in size ($(|G_k|)_{k \geq 0}$ is increasing). Using theorem 4.2.2, this is equivalent to

$$\pi_A(\mathcal{F}) = \max_{\phi \in \text{Hom}^+(\mathcal{A}^\sigma, \mathbb{R})} \phi(A). \quad (\text{P1})$$

We have rephrased the definition for $\pi_A(\mathcal{F})$ as an optimization problem for $\phi \in \text{Hom}^+(\mathcal{A}^\sigma, \mathbb{R})$ and the maximum is achieved for some extremal homomorphism. Essentially, positive homomorphisms are precisely those corresponding to the limits of convergent graph sequences.

Note: we denote $F \geq 0$ for a σ -flag $F \in \mathcal{A}^\sigma$ if $\phi(F) \geq 0$ for any $\phi \in \text{Hom}^+(\mathcal{A}^\sigma, \mathbb{R})$. The averaging operator is essentially the probability that the embedding creates an isomorphic copy of the flag. Therefore, it is trivial that if $\phi(F) \geq 0$, then $\phi(\llbracket F \rrbracket) \geq 0$.

We are going to quickly revisit Mantel's Theorem but we are going to approach this formally in the language of flag algebra. Let G be a large \mathcal{F} -free graph where $\mathcal{F} = \{ \triangle \}$. We need to show $\phi(\circ \circ) \leq \frac{1}{2}$ for any $\phi \in \text{Hom}^+(\mathcal{A}^\sigma, \mathbb{R})$. i.e $\circ \circ \leq \frac{1}{2}$.

Consider

$$(\circ \circ - \bullet \bullet)^2 \geq 0.$$

After expanding the square,

$$\circ \circ \bullet \bullet - 2 \bullet \circ \bullet \bullet + \bullet \bullet \bullet \bullet \geq 0.$$

Using our definition for multiplication, we get

$$\bullet \bullet \bullet \bullet + \bullet \bullet \bullet \bullet - \bullet \bullet \bullet \bullet - \bullet \bullet \bullet \bullet + \bullet \bullet \bullet \bullet \geq 0.$$

We can then apply the averaging operator and since it linear, we obtain

$$\llbracket \bullet \bullet \bullet \bullet \rrbracket + \llbracket \bullet \bullet \bullet \bullet \rrbracket - \llbracket \bullet \bullet \bullet \bullet \rrbracket - \llbracket \bullet \bullet \bullet \bullet \rrbracket + \llbracket \bullet \bullet \bullet \bullet \rrbracket \geq 0$$

$$\bullet \bullet \bullet \bullet + \frac{1}{3} \bullet \bullet \bullet \bullet - \frac{2}{3} \bullet \bullet \bullet \bullet - \frac{2}{3} \bullet \bullet \bullet \bullet + \frac{1}{3} \bullet \bullet \bullet \bullet \geq 0$$

$$\bullet \bullet \bullet \bullet - \frac{1}{3} \bullet \bullet \bullet \bullet - \frac{1}{3} \bullet \bullet \bullet \bullet \geq 0.$$

We then divide this by 2 and add this to the following equation

$$\text{hook} = \frac{1}{3} \text{triple} + \frac{2}{3} \text{fork}$$

and we arrive at

$$\text{hook} \leq \frac{1}{2} \text{triple} + \frac{1}{6} \text{triple} + \frac{1}{2} \text{fork} \leq \frac{1}{2}.$$

Therefore, $\pi(K_3) = \max_{\phi \in \text{Hom}^+(\mathcal{A}^\sigma, \mathbb{R})} \phi(\text{hook}) \leq \frac{1}{2}.$ \square

In this example, we started with our previous inequality from chapter 2, but it is useful to have a generalized theorem for generating inequalities. In [11], Razborov proved a flag algebraic version of the Cauchy-Schwartz inequality which are commonly used to generate the inequalities.

Theorem 4.2.3. *For any $f, g \in \mathcal{A}^\sigma$,*

$$\phi(\llbracket f^2 \rrbracket) \phi(\llbracket g^2 \rrbracket) \geq \phi(\llbracket fg \rrbracket)^2$$

More specifically when $g = \sigma$,

$$\phi(\llbracket f^2 \rrbracket) \geq \phi(\llbracket f^2 \rrbracket) \phi(\sigma') \geq \phi(\llbracket f \rrbracket)^2 \geq 0.$$

In the next section, we will showcase how we can use the Cauchy-Schwartz inequalities to formulate the semidefinite problem.

4.3 Duality and the Semidefinite problem

One of the key concepts in optimization is the concept of duality. Duality means that optimization problems can be viewed from two equivalent perspectives. These are denoted as the primal problem and the dual problem. We have denoted the problem in primal form as the RHS of P1 and we are going to rephrase it in terms of the dual problem

$$\begin{aligned} \min \quad & \lambda \\ \text{subject to} \quad & \phi \leq \lambda \text{ for all } \phi \in \text{Hom}^+(\mathcal{A}^\sigma, \mathbb{R}). \end{aligned} \tag{D1}$$

In general, dual problems provide a bound on the optimal solutions of primal problems. By construction, an optimal solution to D1 provides an optimal solution to P1, however in general it is not true for all optimization problems.

We define a *semantic cone of type σ* to be the set of all elements of the corresponding flag algebra \mathcal{A}^σ which are non-negative under the image of every positive homomorphism. More specifically.

$$\mathcal{S}^\sigma = \{F \in \mathcal{A}^\sigma : \phi(F) \geq 0, \forall \phi \in \text{Hom}^+(\mathcal{A}^\sigma, \mathbb{R})\}.$$

We define $(\mathcal{A}^\sigma)^*$ to be the *dual vector space to \mathcal{A}^σ* . This is the space of all linear maps from \mathcal{A}^σ to \mathbb{R} . We then define *the dual cone of type σ* to be the elements of $(\mathcal{A}^\sigma)^*$ which map every element of \mathcal{S}^σ to a non-negative real number. Formally

$$(\mathcal{S}^\sigma)^* = \{\phi \in (\mathcal{A}^\sigma)^* : \phi(F) \geq 0, \forall F \in \mathcal{S}^\sigma\}.$$

Since the averaging operator maps \mathcal{A}^σ to \mathcal{A}^\emptyset , we can see that the image of \mathcal{S}^σ under the averaging operator is a subset of \mathcal{S}^\emptyset . Also note that $(\mathcal{S}^\sigma)^*$ contains all positive homomorphisms which are the elements of $\text{Hom}^+(\mathcal{A}^\sigma, \mathbb{R})$. If \emptyset is an empty flag and $\phi \in \text{Hom}^+(\mathcal{A}^\sigma, \mathbb{R})$, it is trivial to see $\phi(\emptyset) = 1$ as $d(\emptyset; G) = 1$ by the definition of density. So it follows that

$$\max_{\phi \in \text{Hom}^+(\mathcal{A}^\sigma, \mathbb{R})} \phi(A) \leq \max_{\phi \in (\mathcal{S}^\emptyset)^*, \phi(\emptyset)=1} \phi(A)$$

This is true because every ϕ that satisfies the constraints the right hand side satisfies the constraints on the left. So we can write the right hand side as an optimization problem.

$$\begin{aligned} & \max \quad \phi(A) \\ & \text{subject to} \quad \phi \in (\mathcal{S}^\sigma)^*, \phi(\emptyset) = 1. \end{aligned} \tag{P2}$$

So, the optimal solution for P1 is less than or equal to the optimal solution for P2.

P2 is a conic programming problem [2] as we are optimizing over the intersection between a cone, $(\mathcal{S}^\sigma)^*$ and a subspace generated by $\phi(\emptyset) = 1$. Let us consider the dual problem to P2 which is

$$\begin{aligned} & \min \quad \lambda \\ & \text{subject to} \quad \lambda \emptyset - A \in \mathcal{S}^\emptyset. \end{aligned} \tag{D2}$$

Every solution to D2 provides an upper bound to the optimal value of P2. Since if $\lambda \emptyset - A \in \mathcal{S}^\emptyset$, then by definition, $\phi(\lambda \emptyset - A) \geq 0$. We can further show that the values of P2 and D2 are equal [9], however a more important result is that the the optimal values for D1 and D2 are equal because $\phi(A) \leq \lambda$ for $\phi \in \text{Hom}^+(\mathcal{A}^\sigma, \mathbb{R})$ if and only if $\phi(\lambda \emptyset - A) \geq 0$ for all ϕ if and only if $\lambda \emptyset - A \in (\mathcal{S}^\sigma)^*$.

All four of our optimization problems have the same optimal solution and we can use the formulation of D2 to solve this problem. Though solving D2 seems like we need to understand the global structure of \mathcal{S}^\emptyset however, we can simplify this problem by replacing \mathcal{S}^\emptyset with a cone $\mathcal{C} \subseteq \mathcal{S}^\emptyset$. In general, by the definition of a positive homomorphism, any conic-combination of σ -flags (a linear combination with non-negative coefficients) is in \mathcal{S}^σ . Also, if $f \in \mathcal{A}^\sigma$ and $f = g_1^2 + \dots + g_t^2$, then via the Cauchy-Schwarz inequality, we get

$$\phi(f) = \phi(g_1^2) + \dots + \phi(g_t^2) \geq \phi(g_1)^2 + \dots + \phi(g_t)^2 \geq 0.$$

This implies $f \in \mathcal{S}^\sigma$ and the elements will generate a subcone of \mathcal{S}^σ for which we can optimize over and use the averaging operator to map it to $\mathcal{C} \subseteq \mathcal{S}^\emptyset$ which provides a valid bound for D2. So, let us define some concepts to formulate these ideas.

We define the degree of a vector $f \in \mathbb{R}\mathcal{F}^\sigma$ to the largest order of a flag appearing in the linear combination of F and we define the degree of the zero vector to be -1 . This is to distinguish between the degrees of the zero vector and \emptyset . We can extend this definition and define the degree of a vector $f \in \mathcal{A}^\sigma$, by setting the degree of $f + \mathcal{K}^\sigma \in \mathcal{A}^\sigma$ to be the smallest degree of any $g \in f + \mathcal{K}^\sigma$. In other words, it is the largest order of a flag appearing in the linear combination of f in \mathcal{A}^σ i.e after the elements of \mathcal{K}^σ have been factored out.

For a fixed type σ and $n \geq |V(\sigma)|$, we define $v_{\sigma,n}$ to be elements of the $|\mathcal{F}_n^\sigma| \times 1$ vector of elements in \mathcal{A}^σ which enumerate the elements of \mathcal{A}^σ in some fixed order. As an example, if $\mathcal{F} = \triangle$, then

$$v_{\emptyset,2} = \left(\begin{array}{c} \circ \\ \diagup \end{array}, \begin{array}{c} \circ \\ \diagdown \end{array} \right)^T \quad v_{\bullet,3} = \left(\begin{array}{c} \bullet \\ \circ \end{array}, \begin{array}{c} \bullet \\ \circ \end{array}, \begin{array}{c} \bullet \\ \circ \end{array}, \begin{array}{c} \bullet \\ \circ \end{array}, \begin{array}{c} \bullet \\ \circ \end{array} \right)^T$$

This allows us to introduce the following theorem from [9].

Theorem 4.3.1. *Let σ be a fixed type, and let $f \in \mathcal{A}^\sigma$ and $n \geq |V(\sigma)|$. Then there are vectors $g_1, \dots, g_t \in \mathcal{A}^\sigma$ for some $t \geq 1$ and each of degree of at most n , such that $f = g_1^2 + \dots + g_t^2$ if and only if there exists a positive semidefinite matrix Q with dimensions $|\mathcal{F}_n^\sigma| \times |\mathcal{F}_n^\sigma|$, such that $f = v_{\sigma,n}^T Q v_{\sigma,n}$.*

If we fix a $v_{\sigma,n}$ and Q , then $v_{\sigma,n}^T Q v_{\sigma,n}$ is the sum of squares and $v_{\sigma,n}^T Q v_{\sigma,n} \in \mathcal{S}^\sigma$ and therefore, $\llbracket v_{\sigma,n}^T Q v_{\sigma,n} \rrbracket \in \mathcal{S}^\emptyset$. Since any conic combination, γ , of \emptyset -flags lies in \mathcal{S}^\emptyset , any feasible solution to the following problem is an upper bound for the optimal solution for our problem P1.

$$\begin{aligned} & \min \quad \lambda \\ & \text{subject to} \quad \lambda \emptyset - A = \gamma + \llbracket v_{\sigma,n}^T Q v_{\sigma,n} \rrbracket \\ & \text{where} \quad \gamma \text{ is an conic combination of } \emptyset\text{-flags} \\ & \text{and} \quad Q \text{ is a } |\mathcal{F}_n^\sigma| \times |\mathcal{F}_n^\sigma| \text{ positive semidefinite matrix.} \end{aligned} \tag{D3}$$

This is almost a semidefinite problem. The identity $\lambda\emptyset - A = \gamma + \llbracket v_{\sigma,n}^T Q v_{\sigma,n} \rrbracket$ is an identity on vectors in \mathcal{A}^\emptyset and is not a finite number of linear constraints on λ and the entries of Q . However, we can translate this into several linear constraints as follows.

Let B and C be $n \times n$ matrices, then we denote

$$\langle B, C \rangle = \text{tr}(B^T C) = \sum_{i,j=1}^n B_{ij} C_{ij}.$$

Then,

$$\llbracket v_{\sigma,n}^T Q v_{\sigma,n} \rrbracket = \llbracket \langle v_{\sigma,n}^T v_{\sigma,n}, Q \rangle \rrbracket = \langle \llbracket v_{\sigma,n}^T v_{\sigma,n} \rrbracket, Q \rangle.$$

Note: When we are applying the averaging operator to a matrix, we are applying the averaging operator pointwise to every entry in the matrix.

We can rewrite $\lambda\emptyset - A = \gamma + \llbracket v_{\sigma,n}^T Q v_{\sigma,n} \rrbracket$ as $\lambda\emptyset - A = \gamma + \langle \llbracket v_{\sigma,n} v_{\sigma,n}^T \rrbracket, Q \rangle$. However, this is a condition in \mathcal{A}^\emptyset . We can lift this condition to $\mathbb{R}\mathcal{F}^\emptyset$ by choosing a representative element for A from $A + \mathcal{K}^\emptyset$ and a representative for every element of \mathcal{A}^\emptyset in the matrix $\llbracket v_{\sigma,n}^T v_{\sigma,n} \rrbracket$ from their respective cosets.

For example, if $v = v_{\bullet,2} = \left(\begin{smallmatrix} \circ \\ \nearrow \end{smallmatrix}, \begin{smallmatrix} \circ \\ \bullet \end{smallmatrix} \right)^T$. Then by applying the definition of multiplication in \mathcal{A}^\bullet and choosing the representatives in $\mathbb{R}\mathcal{F}^\bullet$ of smallest size ($l = 3$ and not $l = 4$), we get

$$vv^T = \begin{pmatrix} \begin{smallmatrix} \bullet \\ \circ \circ + \nearrow \circ \end{smallmatrix} & \frac{1}{2} \begin{smallmatrix} \bullet \\ \circ \circ + \nearrow \circ \end{smallmatrix} \\ \frac{1}{2} \begin{smallmatrix} \bullet \\ \circ \circ + \nearrow \circ \end{smallmatrix} & \begin{smallmatrix} \bullet \\ \triangle \end{smallmatrix} \end{pmatrix}$$

and after applying the averaging operator, we get

$$\llbracket vv^T \rrbracket = \begin{pmatrix} \begin{smallmatrix} \circ \\ \circ \circ + \frac{1}{3} \circ \circ \end{smallmatrix} & \frac{1}{3} \begin{smallmatrix} \circ \\ \circ \circ + \nearrow \circ \end{smallmatrix} \\ \frac{1}{3} \begin{smallmatrix} \circ \\ \circ \circ + \nearrow \circ \end{smallmatrix} & \frac{1}{3} \begin{smallmatrix} \circ \\ \triangle \end{smallmatrix} \end{pmatrix}.$$

Now that we are working with representatives in $\mathbb{R}\mathcal{F}^\emptyset$, we need $\lambda\emptyset - A = \gamma + \langle \llbracket v_{\sigma,n} v_{\sigma,n}^T \rrbracket, Q \rangle$ to hold in \mathcal{A}^\emptyset .

Earlier, we identified a mapping $d : \mathbb{R}\mathcal{F}^\sigma \rightarrow \mathbb{R}$ and associated with a compact vector space. This also means that we have extended it linearly so $d(F_1 + F_2; H) = d(F_1; H) + d(F_2; H)$ and $d(cF_1; H) = cd(F_1; H)$ for $F_1, F_2 \in \mathbb{R}\mathcal{F}^\sigma$ and $c \in \mathbb{R}$. We can then use the following lemma from [9] to show that our identity on vectors in \mathcal{A}^σ is going to hold if a set of comparable relations hold in $\mathbb{R}\mathcal{F}^\emptyset$ for densities for every graph H in a sufficiently large \mathcal{F}_l^\emptyset .

Lemma 4.3.1. *Suppose that $d(\lambda\emptyset - A; H) = d(\gamma; H) + d(\langle \llbracket v_{\sigma,n} v_{\sigma,n}^T \rrbracket, Q \rangle; H)$ for every $H \in \mathcal{F}_l^\emptyset$ for some fixed $l > 0$. Then $\lambda\emptyset - A = \gamma + \langle \llbracket v_{\sigma,n} v_{\sigma,n}^T \rrbracket, Q \rangle$ holds in \mathcal{A}^\emptyset .*

If we can eliminate γ from our identity, then we no longer have to optimize over the space of conic combination of \emptyset -flags. In $\mathbb{R}\mathcal{F}^\emptyset$, $d(\gamma; H) \geq 0$ for any conic combination of \emptyset -flags as the density is always non-negative. We can then use the following lemma from [9] to do this.

Lemma 4.3.2. *Let $l > 0$ be fixed. If $d(\lambda\emptyset - A; H) \geq d(\langle \llbracket v_{\sigma,n} v_{\sigma,n}^T \rrbracket, Q \rangle; H)$ for every $H \in \mathcal{F}_l^\emptyset$, then there exists a conic combination γ of \emptyset -flags such that $d(\lambda\emptyset - A; H) = d(\gamma; H) + d(\langle \llbracket v_{\sigma,n} v_{\sigma,n}^T \rrbracket, Q \rangle; H)$ for every $H \in \mathcal{F}_l^\emptyset$.*

Finally, for a matrix M with entries in $\mathbb{R}\mathcal{F}^\sigma$, let $d(M; H)$ denote the matrix generated by applying the linearly extended density function $d(F; H)$ to every entry in the matrix. So by linearity and since $d(\emptyset; H) = 1$, we have

$$d(\lambda\emptyset - A; G) = d(\gamma; H) + d(\langle \llbracket v_{\sigma,n} v_{\sigma,n}^T \rrbracket, Q \rangle; H) \text{ for every } H \in \mathcal{F}_l^\emptyset$$

if and only if

$$\lambda \geq d(A; H) + d(\langle \llbracket v_{\sigma,n} v_{\sigma,n}^T \rrbracket; H), Q \rangle \text{ for every } H \in \mathcal{F}_l^\emptyset.$$

This is a finite number of linear constraints on λ and entries of Q . Therefore, we can write this as an optimization problem

$$\begin{aligned} \min \quad & \lambda \\ \text{subject to} \quad & \lambda \geq d(A; H) + d(\langle \llbracket v_{\sigma,n} v_{\sigma,n}^T \rrbracket; H), Q \rangle \text{ for every } H \in \mathcal{F}_l^\emptyset \text{ where } l > 0 \text{ is fixed.} \quad (\text{D4}) \\ \text{and} \quad & Q \text{ is a } |\mathcal{F}_n^\sigma| \times |\mathcal{F}_n^\sigma| \text{ positive semidefinite matrix.} \end{aligned}$$

This is our desired optimization problem and is D3 in semidefinite form. Therefore the solution to D4 provides an upper bound to P1 which is the Turán density.

To show how this works in practice, we can demonstrate how we can to generate the inequalities for Mantel's theorem .

Recall that , if $v = v_{\bullet,2} = \left(\begin{smallmatrix} \circ \\ \bullet \end{smallmatrix}, \begin{smallmatrix} \circ \\ \bullet \end{smallmatrix} \right)^T$, then

$$\llbracket vv^T \rrbracket = \begin{pmatrix} \begin{smallmatrix} \circ & \circ \\ \circ & \circ \end{smallmatrix} + \frac{1}{3} \begin{smallmatrix} \circ & \circ \\ \circ & \circ \end{smallmatrix} & \frac{1}{3} \left(\begin{smallmatrix} \circ & \circ \\ \circ & \circ \end{smallmatrix} + \begin{smallmatrix} \circ & \circ \\ \circ & \circ \end{smallmatrix} \right) \\ \frac{1}{3} \left(\begin{smallmatrix} \circ & \circ \\ \circ & \circ \end{smallmatrix} + \begin{smallmatrix} \circ & \circ \\ \circ & \circ \end{smallmatrix} \right) & \frac{1}{3} \begin{smallmatrix} \circ & \circ \\ \circ & \circ \end{smallmatrix} \end{pmatrix}.$$

Therefore, D4 becomes

$$\begin{aligned} \min \quad & \lambda \\ \text{subject to} \quad & \lambda \geq d(\mathcal{J}; H) + d(\llbracket vv^T \rrbracket; H), Q \rangle \text{ for every } H \in \mathcal{F}_3^\emptyset \\ \text{and} \quad & Q \text{ is a } |\mathcal{F}_2^\bullet| \times |\mathcal{F}_2^\bullet| \text{ positive semidefinite matrix.} \end{aligned}$$

Let $Q = \begin{pmatrix} q_{11} & q_{12} \\ q_{12} & q_{22} \end{pmatrix}$ be a symmetric positive semidefinite matrix. Once again, $\mathcal{F}_3^\emptyset = \{ \begin{smallmatrix} \circ & & \circ \\ \circ & \circ & \circ \end{smallmatrix}, \begin{smallmatrix} \circ & & \circ \\ \circ & \circ & \circ \end{smallmatrix}, \begin{smallmatrix} \circ & & \circ \\ \circ & \circ & \circ \end{smallmatrix} \}$.

For $H = \begin{smallmatrix} \circ & & \circ \\ \circ & \circ & \circ \end{smallmatrix}$, we have $d(\begin{smallmatrix} \circ & & \circ \\ \circ & \circ & \circ \end{smallmatrix}) = 0$ and $d(\llbracket vv^T \rrbracket; H) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$. Therefore, the linear constraint generated is

$$\lambda \geq \left\langle \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, Q \right\rangle = q_{11}.$$

For $H = \begin{smallmatrix} \circ & & \circ \\ \circ & \circ & \circ \end{smallmatrix}$, we have $d(\begin{smallmatrix} \circ & & \circ \\ \circ & \circ & \circ \end{smallmatrix}) = \frac{1}{3}$ and $d(\llbracket vv^T \rrbracket; H) = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & 0 \end{pmatrix}$. Therefore, the linear constraint generated is

$$\lambda \geq \left\langle \begin{pmatrix} \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & 0 \end{pmatrix}, Q \right\rangle = \frac{1}{3} + \frac{1}{3}q_{11} + \frac{2}{3}q_{12}.$$

And finally, for $H = \begin{smallmatrix} \circ & & \circ \\ \circ & \circ & \circ \end{smallmatrix}$, we have $d(\begin{smallmatrix} \circ & & \circ \\ \circ & \circ & \circ \end{smallmatrix}) = \frac{1}{3}$ and $d(\llbracket vv^T \rrbracket; H) = \begin{pmatrix} 0 & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} \end{pmatrix}$. Therefore, the final linear constraint generated is

$$\lambda \geq \left\langle \begin{pmatrix} 0 & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} \end{pmatrix}, Q \right\rangle = \frac{2}{3} + \frac{2}{3}q_{12} + \frac{1}{3}q_{22}.$$

These are the same inequalities generated in the proof for Mantels theorem from chapter 3 (3.1) and this concludes the algebra of the flag algebra method. In the next chapter, we are going to show some of applications of the flag algebra method to some problems in extremal graph theory.

Applications to Graphs

5.1.1 Structure of the Extremal Graph

Let G be the extremal case for a K_3 -free graph ($\circ = \frac{1}{2}$). In our proofs, we derived

$$1 \leq \begin{array}{c} \circ \\ \circ \quad \circ \end{array} + \frac{1}{3} \begin{array}{c} \circ \\ \circ - \circ \end{array} + \begin{array}{c} \circ \\ \diagup \quad \diagdown \\ \circ \quad \circ \end{array}$$

$$1 = \phi(\emptyset) = \sum_{H \in \mathcal{F}_3^\sigma} d(\emptyset; H)H$$

Combining the two equations together, we get

$$0 \leq -\frac{2}{3} \begin{array}{c} \circ \\ \circ - \circ \end{array}$$

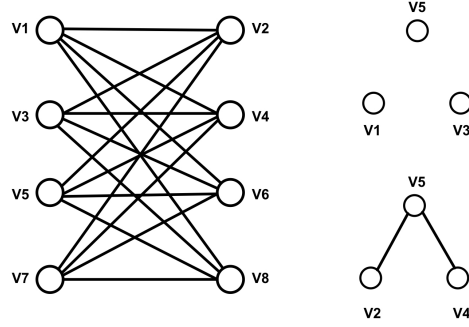
and since $F \geq 0$ for all $F \in \mathcal{A}^\sigma$,

$$\begin{array}{c} \circ \\ \circ - \circ \end{array} = 0$$

and therefore, it is not present in the extremal case.

We can show that the extremal graph is a complete bipartite subgraph with classes as even as possible. $K_{\lfloor n/2 \rfloor, \lceil n/2 \rceil}$ has precisely $\lfloor \frac{n^2}{4} \rfloor$ edges and is triangle-free. We can also observe that any

three vertex subgraph of the extremal graph is either $\begin{array}{c} \circ \\ \circ - \circ \end{array}$ or $\begin{array}{c} \circ \\ \diagup \quad \diagdown \\ \circ \quad \circ \end{array}$.



The extremal graph $K_{4,4}$ for $n=8$ and all 3-vertex subgraphs.

In fact, for all $H \in \mathcal{F}_l^\emptyset$, if H is an induced subgraph of G , then $d(A; H) + c_H$ must be equal to the upper-bound generated by this method, $\pi_A(\mathcal{F})$. This means that if $d(A; H) + c_H \neq \pi_A(\mathcal{F})$ for some $H \in \mathcal{F}_l^\emptyset$, then H is not present in the extremal case which allows us to build an idea about the structure of this graph. However, it is worth stressing that $d(A; H) + c_H = \pi_A(\mathcal{F})$ does not guarantee that H is a subgraph of the extremal case. It is the lack of equivalence which gives us information.

This is a very useful advantage of the flag algebra method. It gives us valuable insight into the structure of the extremal graph by allowing us to see what subgraphs of order l are present in the extremal case.

5.1.2 Minimum density of Triangles

Let us consider a slightly different problem to Mantel's Theorem. Given a large G , instead of trying to maximise \nearrow subject to $\triangle = 0$, consider

$$\begin{aligned} & \min \quad \triangle \\ & \text{subject to} \quad \nearrow \geq p. \end{aligned}$$

Since we are no longer working with triangle-free graphs, $\mathcal{F}_3^\emptyset = \{ \circ \overset{\circ}{\circ} \circ, \circ \text{---} \overset{\circ}{\circ}, \nearrow, \triangle \}$. Using the chain rule, we get

$$\begin{aligned} \triangle &= \sum_{H \in \mathcal{F}_3^\emptyset} d(\triangle; H)H \\ \triangle &= 0 \circ \overset{\circ}{\circ} \circ + 0 \circ \text{---} \overset{\circ}{\circ} + 0 \nearrow + \triangle \geq \min\{0, 0, 0, 1\}. \end{aligned} \tag{5.1}$$

Furthermore, we have

$$\begin{aligned} p \leq \nearrow &= \sum_{H \in \mathcal{F}_3^\emptyset} d(\nearrow; H)H \\ p \leq \nearrow &= 0 \circ \overset{\circ}{\circ} \circ + \frac{1}{3} \circ \text{---} \overset{\circ}{\circ} + \frac{2}{3} \nearrow + \frac{3}{3} \triangle. \end{aligned}$$

Using the chain rule again, we have

$$\begin{aligned} 1 = \phi(\emptyset) &= \sum_{H \in \mathcal{F}_3^\sigma} d(\emptyset; H)H \\ 1 &= \circ \overset{\circ}{\circ} \circ + \circ \text{---} \overset{\circ}{\circ} + \nearrow + \triangle, \end{aligned}$$

we have

$$p = p \circ \overset{\circ}{\circ} \circ + p \circ \text{---} \overset{\circ}{\circ} + p \nearrow + p \triangle.$$

.

Therefore,

$$p \circ \circ + (p - \frac{1}{3}) \circ \circ + (p - \frac{2}{3}) \circ \circ + (p - 1) \circ \circ \leq 0.$$

We can multiply a multiplicative constraint $k \geq 0$ which provides us another variable to optimize for.

$$k(p \circ \circ + (p - \frac{1}{3}) \circ \circ + (p - \frac{2}{3}) \circ \circ + (p - 1) \circ \circ) \leq 0. \quad (5.2)$$

Recall, $\mathcal{F}_2^\bullet = \{\bullet \circ, \bullet \circ\}$ and

$\mathbb{E}_{\theta \in \Theta_H}[d(F_i, F_j, H, \theta)]$	$\circ \circ$	$\circ \circ$	$\circ \circ$
$\bullet \circ, \bullet \circ$	1	1/3	0
$\bullet \circ, \bullet \circ$	0	2/3	2/3
$\bullet \circ, \bullet \circ$	0	0	1/3

We can combine this with a variation of our averaging inequalities. If

$$\sum_{H \in \mathcal{F}_3^\emptyset} c_H(\sigma, m, Q) H \geq 0.$$

where $c_H(\sigma, m, Q) = \sum_{F_i, F_j \in \mathcal{F}_2^\bullet} q_{ij} \mathbb{E}_{\theta \in \Theta_H}[d(F_i, F_j, H, \theta)]$, and q_{ij} is the entry of a symmetric positive definite matrix $Q \in \mathbb{R}^{2 \times 2}$, then

$$-q_{11} \circ \circ - \frac{q_{11} + 2q_{12}}{3} \circ \circ - \frac{2q_{12} + q_{22}}{3} \circ \circ - q_{22} \circ \circ \leq 0. \quad (5.3)$$

Combining (5.1) with (5.2) and (5.3), we get

$$\circ \circ \geq (pk - q_{11}) \circ \circ + ((p - \frac{1}{3})k - \frac{q_{11} + 2q_{12}}{3}) \circ \circ + ((p - \frac{2}{3})k - \frac{2q_{12} + q_{22}}{3}) \circ \circ + (1 + (p - 1)k - q_{22}) \circ \circ$$

$$\circ \circ \geq \min\{pk - q_{11}, (p - \frac{1}{3})k - \frac{q_{11} + 2q_{12}}{3}, (p - \frac{2}{3})k - \frac{2q_{12} + q_{22}}{3}, 1 - (p - 1)k - q_{22}\}$$

for which we can optimize for. We can replicate the example from [5]. If $p = 0.6$, our equation becomes

$$\triangle \geq \min\{0.6k - q_{11}, (0.6 - \frac{1}{3})k - \frac{q_{11} + 2q_{12}}{3}, (0.6 - \frac{2}{3})k - \frac{2q_{12} + q_{22}}{3}, 1 + (0.6 - 1)k - q_{22}\}.$$

For

$$Q = \begin{pmatrix} 0.72 & -0.48 \\ -0.48 & 0.32 \end{pmatrix}$$

and $d = 1.4$, we get

$$\triangle \geq \min\{0.12, 0.453333, 0.12, 0.12\} = 0.12$$

We could further improve this bound by considering $H \in \mathcal{F}_l^\emptyset$ by using $l = 4$ or $l = 5$. Furthermore, Razborov [12] has shown, that

$$\triangle \geq \frac{(t-1)(t-2\sqrt{t(t-p(t+1))})^2(t+\sqrt{t(t-p(t+1))})^2}{t^2(t+1)^2}$$

where $t = \lfloor \frac{1}{1-p} \rfloor$.

For $p = 0.6$, this formula gives us $\triangle \geq 0.1415009$ which is a better bound than the bound we have found in the example above. However, the example above showcases one way flag algebras can be used to solve extremal problems besides the Turán problem.

5.2 K_4 -Free graphs

Let us finally move away from Mantel's Theorem and consider a K_4 -free graph. We are going to investigate the structure of K_4 -free graphs and let's start with the edge Turán density of a K_4 -free graph.

Theorem 5.2.1. $\pi(K_4) \leq \frac{2}{3}$

Proof. Firstly, let us consider a family, \mathcal{F}_l^\emptyset , of K_4 -free graphs on $l = 4$ vertices. I.e

$$\mathcal{F}_4^\emptyset = \left\{ \begin{array}{c} \text{Diagram 1: } K_4 \\ \text{Diagram 2: } K_4 \\ \text{Diagram 3: } K_4 \\ \text{Diagram 4: } K_4 \\ \text{Diagram 5: } K_4 \\ \text{Diagram 6: } K_4 \\ \text{Diagram 7: } K_4 \\ \text{Diagram 8: } K_4 \\ \text{Diagram 9: } K_4 \\ \text{Diagram 10: } K_4 \\ \text{Diagram 11: } K_4 \\ \text{Diagram 12: } K_4 \\ \text{Diagram 13: } K_4 \\ \text{Diagram 14: } K_4 \\ \text{Diagram 15: } K_4 \\ \text{Diagram 16: } K_4 \\ \text{Diagram 17: } K_4 \\ \text{Diagram 18: } K_4 \\ \text{Diagram 19: } K_4 \\ \text{Diagram 20: } K_4 \end{array} \right\}$$

We can calculate the edge density of each of the graphs in \mathcal{F}_l^\emptyset

H										
$d(H)$	5/6	2/3	2/3	1/2	1/2	1/2	1/3	1/3	1/6	0

and we can define our two sets of σ -flags of order $m = 4$ with $s = 2$.

$$\mathcal{F}_4^{\sigma_1} = \left\{ \begin{array}{c} 1 \quad 2 \\ \bullet \quad \bullet \\ \circ \end{array}, \begin{array}{c} 1 \quad 2 \\ \bullet \quad \bullet \\ \diagup \circ \end{array}, \begin{array}{c} 1 \quad 2 \\ \bullet \quad \bullet \\ \diagdown \circ \end{array}, \begin{array}{c} 1 \quad 2 \\ \bullet \quad \bullet \\ \nabla \circ \end{array} \right\}$$

$$\mathcal{F}_4^{\sigma_2} = \left\{ \begin{array}{c} 1 \quad 2 \\ \bullet \quad \bullet \\ \circ \end{array}, \begin{array}{c} 1 \quad 2 \\ \bullet \quad \bullet \\ \diagup \circ \end{array}, \begin{array}{c} 1 \quad 2 \\ \bullet \quad \bullet \\ \diagdown \circ \end{array}, \begin{array}{c} 1 \quad 2 \\ \bullet \quad \bullet \\ \nabla \circ \end{array} \right\}$$

Now, we are going to find for each $H \in \mathcal{F}_4^\emptyset$, the value for $c_H(\sigma_1, 3, Q)$ and $c_H(\sigma_2, 3, P)$. We can compute our values via python and the table below contains the results.

$\mathbb{E}_{\theta \in \Theta_H}[d(F_i, F_j, H, \theta)]$										
							4/12		2/12	
				2/12				2/12		
				2/12				2/12		
			2/12			6/12				
			1/12		3/12					
		8/12		2/12						
	4/12		2/12							
			1/12		3/12					
	4/12		2/12							
	2/12									

$\mathbb{E}_{\theta \in \Theta_H}[d(F_i, F_j, H, \theta)]$											
										2/12	1
									2/12	4/12	
									2/12	4/12	
					6/12			2/12			
						3/12		1/12			
				2/12			8/12				
			2/12	2/12							
					3/12			1/12			
			2/12	2/12							
	2/12	4/12									

Therefore, we have the following inequalities

$$\pi(K_4) \leq \max_{H \in \mathcal{F}_4^0} (d(H) + c_H(\sigma_1, 3, Q) + c_H(\sigma_2, 3, P))$$

$$\begin{aligned} \pi(K_4) \leq & \frac{1}{12} \max \{ 10 + 4q_{24} + 4q_{34} + 2q_{44} + 2p_{44}, \\ & 8 + 8q_{23} + 4p_{44}, \\ & 8 + 3 + 2q_{14} + q_{22} + 2q_{24} + q_{33} + 2q_{34} + 2p_{24} + 2p_{34}, \\ & 6 + 2q_{12} + 2q_{13} + 2q_{23} + 2p_{23} + 2p_{24} + 2p_{34}, \\ & 6 + 3q_{22} + 3q_{33} + 6p_{14}, \\ & 6 + 6q_{14} + 3p_{22} + 3p_{33}, \\ & 4 + 4q_{11} + 8p_{23}, \\ & 4 + 2q_{12} + 2q_{13} + 2p_{12} + 2p_{13} + 2p_{14} + p_{22} + p_{33}, \\ & 2 + 2q_{11} + 2p_{11} + 4p_{12} + 4p_{13}, \\ & 12p_{11} \} \end{aligned}$$

where P and Q are symmetric positive semidefinite matrices. Now, we can use semidefinite programming to solve for matrices P and Q and we get the solution. We have

$$Q = \begin{pmatrix} 1.16353 & 0.392333 & 0.392332 & -0.78466 \\ 0.392333 & 0.666665 & -0.0833325 & -0.58333 \\ 0.392332 & -0.0833325 & 0.666665 & -0.58333 \\ -0.78466 & -0.58333 & -0.58333 & 1.16666 \end{pmatrix}$$

and

$$P = \begin{pmatrix} 0.666667 & -0.0800196 & -0.0800196 & -0.333332 \\ -0.0800196 & 0.835333 & -0.56355 & 0.0400102 \\ -0.0800196 & -0.56355 & 0.835333 & 0.0400102 \\ -0.333332 & 0.0400102 & 0.0400102 & 0.166665 \end{pmatrix}.$$

We can confirm these are symmetric positive semidefinite matrices by evaluating the eigenvalues for P and Q and confirming that they are all greater than or equal to 0.

$$\lambda_Q = 6.93665, 1.45049 \times 10^{-8}, 1.32055 \times 10^{-9}, 3.94712 \times 10^{-10} \geq 0$$

$$\lambda_P = 3.49387, 0.067044, 1.78732 \times 10^{-9}, 2.85903 \times 10^{-10} \geq 0.$$

Therefore, we get

$$\pi(K_4) \leq \max\{0.666667, 0.666667, 0.465894, 0.5363, 0.666667, 0.525337, 0.345477, 0.521105, 0.418353, 0.666667\}$$

and thus,












$$\pi(K_4) \leq \frac{2}{3}.$$

□

Another advantage of the flag algebra method is that we can use the same flags and computations of c_H to solve similar problems. We can use the same flags and choice of l , in the previous example, to find the Turán density of triangles in a K_4 -free graph.

Theorem 5.2.2. $\pi_{K_3}(K_4) \leq \frac{2}{9}$

Proof. Firstly, we must compute $d(K_3, H)$ for $H \in \mathcal{F}_4^\emptyset$.

H											
$d(K_3; H)$	1/2	0	1/4	0	0	1/4	0	0	0	0	0

and using the same σ -flags σ_1 and σ_2 , we have

$$\mathcal{F}_4^{\sigma_1} = \left\{ \begin{array}{c} \overset{1}{\bullet} \text{---} \overset{2}{\bullet} \\ \circ \end{array}, \begin{array}{c} \overset{1}{\bullet} \text{---} \overset{2}{\bullet} \\ \diagup \circ \end{array}, \begin{array}{c} \overset{1}{\bullet} \text{---} \overset{2}{\bullet} \\ \diagdown \circ \end{array}, \begin{array}{c} \overset{1}{\bullet} \text{---} \overset{2}{\bullet} \\ \diagup \diagdown \circ \end{array} \right\}$$

$$\mathcal{F}_4^{\sigma_2} = \left\{ \begin{array}{c} \overset{1}{\bullet} \\ \circ \end{array}, \begin{array}{c} \overset{2}{\bullet} \\ \circ \end{array}, \begin{array}{c} \overset{1}{\bullet} \text{---} \overset{2}{\bullet} \\ \diagup \circ \end{array}, \begin{array}{c} \overset{2}{\bullet} \text{---} \overset{1}{\bullet} \\ \diagdown \circ \end{array}, \begin{array}{c} \overset{2}{\bullet} \text{---} \overset{1}{\bullet} \\ \diagup \diagdown \circ \end{array}, \begin{array}{c} \overset{1}{\bullet} \text{---} \overset{2}{\bullet} \\ \diagdown \diagup \circ \end{array} \right\}$$

$$\pi_{K_3}(K_4) \leq \max_{H \in \mathcal{F}_4^0} (d(K_3; H) + c_H(\sigma_1, 3, Q) + c_H(\sigma_2, 3, P))$$

$$\begin{aligned} \pi_{K_3}(K_4) \leq \frac{1}{12} \max \{ & 6 + 4q_{24} + 4q_{34} + 2q_{44} + 2p_{44}, \\ & 8q_{23} + 4p_{44}, \\ & 3 + 2q_{14} + q_{22} + 2q_{24} + q_{33} + 2q_{34} + 2p_{24} + 2p_{34}, \\ & 2q_{12} + 2q_{13} + 2q_{23} + 2p_{23} + 2p_{24} + 2p_{34}, \\ & 3q_{22} + 3q_{33} + 6p_{14}, \\ & 3 + 6q_{14} + 3p_{22} + 3p_{33}, \\ & 4q_{11} + 8p_{23}, \\ & 2q_{12} + 2q_{13} + 2p_{12} + 2p_{13} + 2p_{14} + p_{22} + p_{33}, \\ & 2q_{11} + 2p_{11} + 4p_{12} + 4p_{13}, \\ & 12p_{11} \} \end{aligned}$$

Through semidefinite programming, we get the following matrices.

$$Q = \frac{1}{12} \begin{pmatrix} 7.32498 & 4.1006 & 4.1006 & -8.20112 \\ 4.1006 & 6.66665 & 3.66668 & -10.3333 \\ 4.1006 & 3.66668 & 6.66665 & -10.3333 \\ -8.20112 & -10.3333 & -10.3333 & 20.6664 \end{pmatrix}$$

and

$$P = \frac{1}{12} \begin{pmatrix} 2.66667 & -0.6944 & -0.6944 & -1.33332 \\ -0.6944 & 5.38242 & -3.17974 & 0.347201 \\ -0.6944 & -3.17974 & 5.38242 & 0.347201 \\ -1.33332 & 0.347201 & 0.347201 & 0.666649 \end{pmatrix}.$$

Therefore, our inequalities become

$$\pi_{K_3}(K_4) \leq \max\{0.222222, 0.222222, -0.0487029, 0.130313, 0.222222, \\ 0.132554, 0.0268195, 0.150854, 0.100195, 0.222222\}$$

Thus,

$$\pi_{K_3}(K_4) \leq \frac{2}{9}.$$

□

We can use this to prove the following theorem.

Theorem 5.2.3. *The number of triangles in a K_4 -free graph is at most $(\frac{n}{3})^3$.*

Proof. Consider a K_4 -free graph, G_n , that contains at least $(\frac{n}{3})^3 + \epsilon$ triangles, where $\epsilon > 0$. Consider the K_4 -free graph, G_{nN} which is a *blow-up* of G_n . This means G_{nN} consists of n independent sets of N vertices. The independent sets form a complete bipartite subgraph if there is an edge between the corresponding vertices in G_n . Therefore, a single triangle in G_n corresponds to N^3 different triangles in G_{nN} . Thus, G_{nN} has nN vertices and at least $((\frac{n}{3})^3 + \epsilon)N^3$ triangles which implies

$$\pi_{K_3}(K_4) \geq \lim_{N \rightarrow \infty} \frac{((\frac{n}{3})^3 + \epsilon)N^3}{\binom{nN}{3}} = \frac{(\frac{nN}{3})^3 + \epsilon N^3}{\binom{nN}{3}} = \frac{2}{9} + \frac{6\epsilon}{n^3} > \frac{2}{9}$$

which is a contradiction. □

If we consider the extremal graph, G , we can show that if the coefficients of any subgraph, $H \in \mathcal{F}_4^\emptyset$, in our inequality, is not equal to $\frac{2}{9}$, then they are not present in the extremal case.

Consider

$$\begin{aligned} \frac{2}{9} = \triangle &\leq \sum_{H \in \mathcal{F}_4^\emptyset} (d(K_3; H) + c_H)H = \sum_{H \in \mathcal{F}_4^\emptyset} k_H H. \\ \frac{2}{9} &\leq \sum_{\substack{H \in \mathcal{F}_4^\emptyset \\ k_H = \frac{2}{9}}} \frac{2}{9}H + \sum_{\substack{H \in \mathcal{F}_4^\emptyset \\ k_H \neq \frac{2}{9}}} k_H H. \end{aligned}$$

Let $\epsilon = \max_{H \in \mathcal{F}_4^\emptyset, k_H \neq \frac{2}{9}} k_H < \frac{2}{9}$. Therefore,

$$\frac{2}{9} \leq \sum_{\substack{H \in \mathcal{F}_4^\emptyset \\ k_H = \frac{2}{9}}} \frac{2}{9}H + \sum_{\substack{H \in \mathcal{F}_4^\emptyset \\ k_H \neq \frac{2}{9}}} \epsilon H.$$

After multiplying by $\frac{2}{9}$, we get

$$1 \leq \sum_{\substack{H \in \mathcal{F}_4^\emptyset \\ k_H = \frac{2}{9}}} H + \sum_{\substack{H \in \mathcal{F}_4^\emptyset \\ k_H \neq \frac{2}{9}}} \frac{9}{2} \epsilon H.$$

Through the chain rule, we have

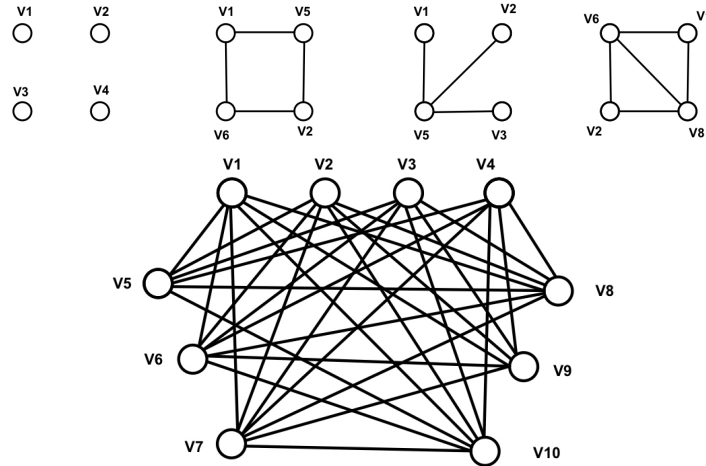
$$1 = \phi(\emptyset) = \sum_{H \in \mathcal{F}_4^\emptyset} d(\phi, \emptyset) H.$$

$$1 = \sum_{H \in \mathcal{F}_4^\emptyset} H.$$

Therefore, we get

$$0 \leq \sum_{H \in \mathcal{F}_4^\emptyset, k_H \neq \frac{2}{9}} \left(\frac{9}{2} \epsilon - 1\right) H.$$

Since $\frac{9}{2} \epsilon - 1 < 0$, and $H \geq 0$ for $H \in \mathcal{A}^\emptyset$, $H = 0$ for $k_H \neq \frac{2}{9}$. Therefore, it is not present in the extremal graph G . The diagram below shows the structure of the extremal graph for $n = 10$ and the valid 4 vertex subgraphs.



The complete tripartite graph $K_{4,3,3}$ and all 4-vertex subgraphs.

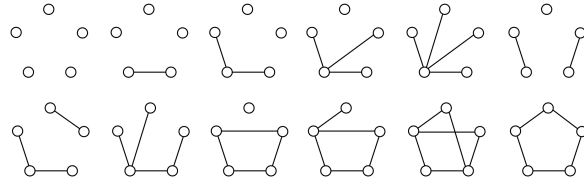
5.3 The Pentagon Problem

In [8], Erdos conjectured that the number of pentagons in a triangle free graph is at most $(\frac{n}{5})^5$ and Grzesik proved this in [6] using the flag algebra method. We are going to replicate this proof in this section.

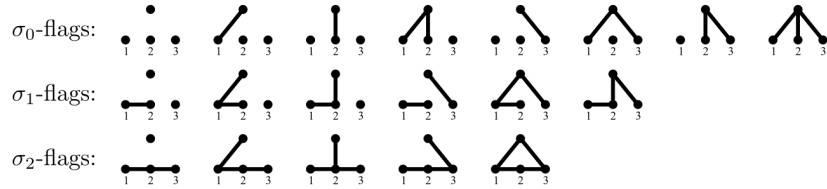
Theorem 5.3.1.

$$\pi_{C_5}(K_3) \leq \frac{24}{625}$$

Proof. We are going to approach this proof using the flag algebra method. We are going to consider triangle free graphs of order $l = 5$. The set of all K_3 -free graphs on 5 vertices are:



We are going to use a combination of 3 different types of order $s = 3$ on $m = 4$ vertices. We are going to denote them $\sigma_0, \sigma_1, \sigma_2$ and associate them with Matrices P, Q, R respectively.



Now we need to find $c_H(\sigma_0, 4, P), c_H(\sigma_1, 4, Q), c_H(\sigma_2, 4, R)$ for each $H \in \mathcal{F}_5^\emptyset$ by computing the expectation of $d(F_i, F_j; H, \theta)$ for each pair of flags over $\theta \in \Theta_H$ and then use the inequality

$$\pi_{C_5}(K_3) \leq \max_{H \in \mathcal{F}_5^\emptyset} \{d(C_5; H) + c_H(\sigma_0, 4, P) + c_H(\sigma_1, 4, Q) + c_H(\sigma_2, 4, R)\}.$$

And after the computations done via the code in the appendix,

$$\begin{aligned}
\pi_{C_5}(K_3) \leq \frac{1}{120} \max\{ & 120p_{11}, \\
& 12p_{11} + 24p_{12} + 24p_{13} + 24p_{15} + 12q_{11}, \\
& 8p_{12} + 8p_{13} + 8p_{14} + 8p_{15} + 8p_{16} + 8p_{17} + 4p_{22} + 4p_{33} + 4p_{55} + 8q_{12} + 8q_{13} + 4r_{11}, \\
& 12p_{14} + 12p_{16} + 12p_{17} + 12p_{18} + 6q_{22} + 6q_{33} + 12r_{13}, \\
& 48p_{18} + 24r_{33}, \\
& 16p_{23} + 16p_{25} + 16p_{35} + 8q_{11} + 16q_{14}, \\
& 8p_{27} + 8p_{36} + 8p_{45} + 8q_{14} + 8q_{24} + 8q_{34} + 4q_{44} + 4r_{11}, \\
& 4p_{23} + 4p_{24} + 4p_{25} + 4p_{26} + 4p_{34} + 4p_{35} + 4p_{37} + 4p_{56} + 4p_{57} + 4q_{12} + 4q_{13} \\
& + 4q_{15} + 4q_{16} + 4q_{23} + 4r_{12} + 4r_{14}, \\
& 4p_{27} + 4p_{28} + 4p_{36} + 4p_{38} + 4p_{45} + 4p_{58} + 4q_{15} + 4q_{16} + 4q_{25} \\
& + 4q_{36} + 4r_{13} + 2r_{22} + 4r_{23} + 4r_{34} + 2r_{44}, \\
& 8p_{44} + 8p_{66} + 8p_{77} + 16q_{23} + 16r_{15}, \\
& 4p_{48} + 4p_{68} + 4p_{78} + 4q_{26} + 4q_{35} + 2q_{55} + 2q_{66} + 4r_{15} + 4r_{23} + 4r_{25} + 4r_{34} + 4r_{35} + 4r_{45}, \\
& 12p_{88} + 24r_{35} + 12r_{55}, \\
& 120 + 4p_{46} + 4p_{47} + 4p_{67} + 4q_{24} + 4q_{26} + 4q_{34} + 4q_{35} + 4q_{45} + 4q_{46} + 4r_{12} + 4r_{14} + 4r_{24}, \\
& 20q_{56} + 20r_{24} \}
\end{aligned}$$

And via semidefinite programming we get a solution for

$$\begin{aligned}
P = & \begin{pmatrix} 0.0384 & -0.0898 & -0.0898 & 0.0706 & -0.0898 & 0.0706 & 0.0706 & -0.0576 \\ -0.0898 & 0.4501 & 0.1648 & -0.12 & 0.1648 & -0.12 & -0.4052 & 0.1346 \\ -0.0898 & 0.1648 & 0.4501 & -0.12 & 0.1648 & -0.4052 & -0.12 & 0.1346 \\ 0.0706 & -0.12 & -0.12 & 0.37 & -0.4052 & 0.0847 & 0.0847 & -0.1058 \\ -0.0898 & 0.1648 & 0.1648 & -0.4052 & 0.4501 & -0.12 & -0.12 & 0.1346 \\ 0.0706 & -0.12 & -0.4052 & 0.0847 & -0.12 & 0.37 & 0.0847 & -0.1058 \\ 0.0706 & -0.4052 & -0.12 & 0.0847 & -0.12 & 0.0847 & 0.37 & -0.1058 \\ -0.0576 & 0.1346 & 0.1346 & -0.1058 & 0.1346 & -0.1058 & -0.1058 & 0.0864 \end{pmatrix} \\
Q = & \begin{pmatrix} 0.884125 & -0.625595 & -0.625595 & -0.742438 & 0.183532 & 0.183532 \\ -0.625595 & 0.864373 & 0.308202 & 0.240488 & -0.290763 & -0.256216 \\ -0.625595 & 0.308202 & 0.864373 & 0.240488 & -0.256216 & -0.290763 \\ -0.742438 & 0.240488 & 0.240488 & 1.43807 & 0.130731 & 0.130731 \\ 0.183532 & -0.290763 & -0.256216 & 0.130731 & 3.46909 & -3.10565 \\ 0.183532 & -0.256216 & -0.290763 & 0.130731 & -3.10565 & 3.46909 \end{pmatrix}
\end{aligned}$$

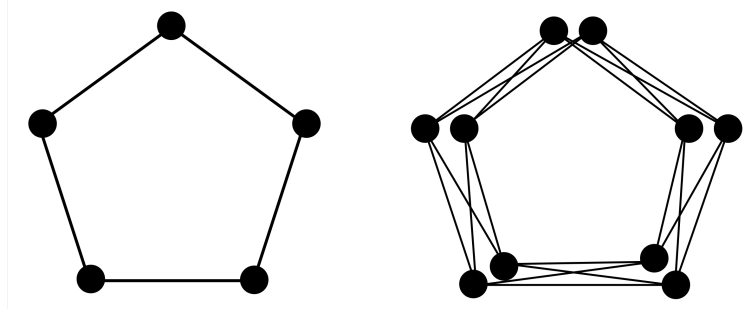


Figure 5.1: The Extremal graph for $n = 5$ and $n = 10$

$$R = \begin{pmatrix} 2.4192 & 0.922 & -0.6344 & 0.922 & -0.5752 \\ 0.922 & 3.4372 & -0.3122 & -2.664 & -0.1488 \\ -0.6344 & -0.3122 & 0.3072 & -0.3122 & 0.01 \\ 0.922 & -2.664 & -0.3122 & 3.4372 & -0.1488 \\ -0.5752 & -0.1488 & 0.01 & -0.1488 & 0.2776 \end{pmatrix}$$

The eigenvalues of these matrices are:

$$\lambda_P = 1.4143, 0.5706, 0.5706, 0.0296, 3.7524 \times 10^{-8}, 1.1616 \times 10^{-8}, 1.1616 \times 10^{-8}, 6.6916 \times 10^{-10}$$

$$\lambda_Q = 6.5749, 2.5372, 1.2719, 0.556, 0.0491, 4.4188 \times 10^{-9}$$

$$\lambda_R = 6.1011, 3.4432, 0.3339, 1.2147 \times 10^{-8}, 1.3317 \times 10^{-9}.$$

All eigenvalues $\lambda_P, \lambda_Q, \lambda_R \geq 0$ which implies that P, Q and R are symmetric positive semidefinite. Therefore, we obtain the following upper bound.

$$\begin{aligned} \pi_{C_5}(K_3) &\leq \max\{0.0384, 0.0384, 0.0384, 0.0384, 0.0384, 0.025882, 0.030095, \\ &\quad 0.034763, 0.0384, 0.0384, 0.0384, 0.0384, -0.011196, 0.0384\} \\ &= 0.0384 = \frac{24}{625}. \end{aligned}$$

□

Note: If we compare our matrices to the matrices in [6], we get different values. The solution for the matrices is not unique in order to get a bound. Different matrices can be generated depending on the algorithm used by the semidefinite solver.

Theorem 5.3.2. *The number of pentagons in a triangle-free graph of order n is at most $(\frac{n}{5})^5$.*

Proof. Say there is a K_3 -free graph G on n vertices that has at least $(\frac{n}{5})^5 + \epsilon$ pentagons, C_5 , where $\epsilon > 0$. Now consider the the blow-up of G_n , G_{nN} which consists of n sets of N independent

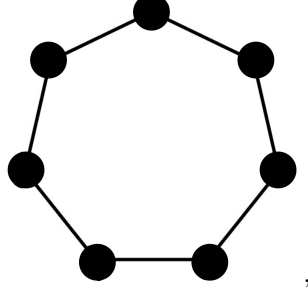


Figure 5.2: C_7 : We can not add another edge without creating a copy of either C_3 or C_5

vertices. The independent sets form a complete bipartite subgraph if there is an edge between the corresponding vertices in G . This means a single pentagon in G corresponds to N^5 different cycles in G_{nN} . Therefore G_{nN} has nN vertices and at least $((\frac{n}{5})^5 + \epsilon)N^5$ cycles. Therefore

$$\pi_{C_5}(K_3) \geq \lim_{N \rightarrow \infty} \frac{((\frac{n}{5})^5 + \epsilon)N^5}{\binom{nN}{5}} = \frac{(\frac{nN}{5})^5 + \epsilon N^5}{\binom{nN}{5}} = \frac{24}{625} + \frac{120\epsilon}{n^5} > \frac{24}{625}.$$

which contradict Theorem 5.3.1. \square

The intuition behind this conjecture is that we can construct a triangle free graph, G , with $n = 5t$ vertices that can be partitioned into 5 independent sets, P_1, \dots, P_5 . Each vertex in a set P_i , is connected to every vertex in set P_{i-1} or P_{i+1} . WLOG, we can say $P_0 = P_5$ and $P_1 = P_6$. This graph contains $t^5 = (\frac{n}{5})^5$ pentagons, and as $n \rightarrow \infty$,

$$\lim_{n \rightarrow \infty} d(C_5; G) = \lim_{n \rightarrow \infty} \frac{(\frac{n}{5})^5}{\binom{n}{5}} = \frac{24}{625}$$

Thus, we have $\frac{24}{625}$ as a lower bound for $\pi_{C_5}(K_3)$. If we can show that $\frac{24}{625}$ is the maximum value by finding the upper bound, we can deduce this is the extremal graph and consequently contains the maximum number of pentagons in a triangle free graph.

We can expand on this by considering the Turán density of C_7 in a graph does not contain a copy of K_3 and C_5 . Let $\mathcal{F} = \{C_5, K_3\}$. Through an example, we can show that $\pi_{C_7}(\mathcal{F}) \geq \frac{720}{117649}$. Let G be the blowup of C_7 on n vertices. Therefore,

$$\pi_{C_7}(\mathcal{F}) \geq \lim_{n \rightarrow \infty} d(C_7; G) = \lim_{n \rightarrow \infty} \frac{(\frac{n}{7})^7}{\binom{n}{7}} = \frac{(\frac{1}{7})^7}{\frac{1}{7!}} = \frac{6!}{7^6} = \frac{720}{117649}.$$

If we can show $\pi_{C_7}(\mathcal{F}) \leq \frac{720}{117649}$, then we have shown that the blow-up of C_7 is the extremal graph, and we can prove that the number of copies of C_7 is at most $(\frac{n}{7})^7$ in a similar way to Theorem 5.3.2.

We can begin to solve this problem by using $l = 7$ and determining \mathcal{F}_7^\emptyset by computer search. There are 89 \mathcal{F} -free graphs on 7 vertices up to isomorphism, so $|\mathcal{F}_7^\emptyset| = 89$, and $d(C_7; H) = 0$ for $H \in \mathcal{F}_7^\emptyset$ except for $H = C_7$. We can generate our c_H inequalities by using three types and flags with $s = 3$ and $m = 5$.

$$\begin{aligned}\sigma_0 &= \begin{array}{ccc} \bullet & \bullet & \bullet \\ 1 & 2 & 3 \end{array} \\ \sigma_1 &= \begin{array}{ccc} \bullet & \bullet & \bullet \\ 1 & 2 & 3 \end{array} \\ \sigma_2 &= \begin{array}{ccc} \bullet & \bullet & \bullet \\ 1 & 2 & 3 \end{array} \end{aligned}$$

Therefore, we have $|\mathcal{F}_5^{\sigma_0}| = 50, |\mathcal{F}_5^{\sigma_1}| = 32, |\mathcal{F}_5^{\sigma_2}| = 24$. and thus

$$\pi_{C_7}(\mathcal{F}) \leq \max_{H \in \mathcal{F}_4^\emptyset} \{d(C_7; H) + c_H(\sigma_0, 5, P) + c_H(\sigma_1, 5, Q) + c_H(\sigma_2, 5, R)\}.$$

Appendix 6.4 contains the inequalities for every $H \in \mathcal{F}_7^\emptyset$. Unfortunately, the semidefinite solver used in this paper was unable to compute a solution for this.

Nonetheless, the theory of flag algebras provides a powerful approach to solving problems in extremal graph theory. It's application to a wide range of different structure has been instrumental to solving problems in extremal combinatorics. In [13], Baber has applied the method to significantly improve the bound of the hypercube Turán density of C_4 . In [14], Pfender and Lidicky have used flag algebras to find upper bounds for many different Ramsey number and [15] demonstrates how flag algebras can be used to find the minimum number of monotone subsequences of length 4 in permutations. In this paper, we have shown the theory of flag algebras with respect to the Turán problem, and we have used flag algebras to obtain bounds for the Turán density and used it analyze the extremal structure of graphs.

Chapter 6

Appendix

The section only contains a sample of the code written for this project. All code, graphs, flags can be found at <https://github.com/mkonchwalla/graph>.

6.1 Main Code

```
import sys
print("Python version: {}".format(sys.version))
import itertools
import numpy as np
import random
from collections import Counter

class Graph:
    def __init__(self, V=[], E=[]):
        self.Vertices= list(V)
        self.Edges = list(E)

    def add_vertex(self, v):
        if v in self.Vertices:
            raise Exception("Vertex is already in the set")
        elif isinstance(v, Vertex):
            self.Vertices.append(v)
```



```

def create_vertex(self,nbd=set() , name = "vertex" , label = None):
    v=Vertex(nbd,name,label)
    self+=v

def delete_vertex(self,v):
    if not v in self.Verticies:
        raise Exception("Vertex is not in the set")
    elif isinstance(v,Vertex):
        self.Verticies.pop(self.Verticies.index(v))

def add_edge(self,v1,v2):
    if v1 in self.Verticies and v2 in self.Verticies:
        v1.neighbourhood.add(v2)
        v2.neighbourhood.add(v1)
        if (v1,v2) not in self.Edges:
            self.Edges.append((v1,v2))
    else:
        raise Exception("Both edges are not valid verticies in this vertex
↪ class")

def delete_edge(self,v1,v2):
    if v1 in self.Verticies and v2 in self.Verticies and (v1,v2) in
↪ self.Edges:
        v1.neighbourhood.remove(v2)
        v2.neighbourhood.remove(v1)
        self.Edges.remove((v1,v2))
    else:
        raise Exception("Both edges are not valid verticies in this vertex
↪ class")

def __add__(self,element):
    if isinstance(element, Edge):
        self.add_edge(element.v1,element.v2)
    elif isinstance(element,Vertex):
        self.add_vertex(element)
    else:
        raise Exception("Invalid data structure")

def __len__(self):
    return len(self.Verticies)

```

```

def contains_copy(self,A):
    """
    This checks if G contains a copy of A.
    This works by taking all the k tuples of V and checking if the induced
    → subgraph is isomorphic to A
    """
    k=len(A)
    G_v = self.Verticies[:]
    for k_tup in itertools.combinations(G_v,k):
        H=induced_subgraph(k_tup,self)
        if isIsomorphic(A,H):
            return True
    return False

def __repr__(self):
    return str(self.Edges)

def degrees(self):
    "Returns the degrees of each vertex in a Graph"
    d=dict(zip(self.Verticies,[0]*len(self)))
    for e in self.Edges:
        d[e[0]]+=1
        d[e[1]]+=1
    return d

def neighbourhood(self):
    """
    Returns the neighbourhood for each vertex in a Graph
    """
    d=dict(zip(self.Verticies,[[] for i in range(len(self))]))
    for e in self.Edges:
        d[e[0]].append(e[1])
        d[e[1]].append(e[0])
    return d

class Vertex:
    def __init__(self, nbd=set(),name="vertex",label=None):
        self.neighbourhood = set(nbd)
        self.name=name
        self.label=label

```

```

def __repr__(self):
    if self.label==None:
        return self.name
    else:
        return self.label

def clear_neighbourhood(self):
    self.neighbourhood = set()

def unlabel_vertex(self):
    self.label= None

class Edge:

    def __init__(self,v1,v2):
        self.head=v1
        self.tail = v2
        self.edge= (v1,v2)

class DiGraph(Graph):
    def __init__(self,V=[], E=[]):
        super().__init__(V, E)

class Flag(Graph):

    def __init__(self,V=[], E=[],sigma= None):
        super().__init__(V, E)
        self.sigma= sigma
        self.labelled_vertices= {}
        self.unlabelled_vertices = self.Vertices[:]
        if sigma !=None and type(sigma)==dict:
            self.assign_label(dict(sigma))

    def add_vertex(self,v):
        if v in self.Vertices:
            raise Exception("Vertex is already in the set")
        elif isinstance(v,Vertex):
            self.Vertices.append(v)
            self.unlabelled_vertices.append(v)

```

```

def mul(self,element):
    """
    Multiplies different graphs together to form partially labelled graphs.
    """
    if isinstance(element,Flag): #Checks Flag
        if isIsomorphic( self.create_type(), element.create_type()): #Checks
            ↪ types are the same
            if set([v.label for v in self.labelled_verticies.values()]) ==
            ↪ set([v.label for v in element.labelled_verticies.values()]): #
            ↪ Checks the same labels
                F_B_V = element.labelled_verticies.values()
                New_F=induced_subgraph(self.Verticies,self)
                corresponding_verticies=dict([(u,v) for u in F_B_V for v in
                ↪ New_F.labelled_verticies.values() if u.label==v.label] )
                for v in F_B_V:
                    corresponding_v = corresponding_verticies[v]
                    for u in element.Verticies: # Goes through the element
                        ↪ verticies and adds verticies
                            if u not in New_F.Verticies and u not in F_B_V: #If
                                ↪ its not in the set already and not a labelled
                                ↪ vertex
                                    New_F.add_vertex(u)
                                    if (u,v) in element.Edges or (v,u) in
                                    ↪ element.Edges:
                                        New_F.add_edge(corresponding_v,u)
                return New_F
            else:
                raise Exception("The labels are not the same")
        else:
            raise Exception(" The Types are not Isomorphic") # The idea
            ↪ here is to generate the new Flag by recreating it and
            ↪ generating the linear span of combinations with labels
    else: raise Exception("Invalid Structure { Not a flag}")

def assign_label(self,sigma):
    """
    Assigns label to the flag given the embedding.
    Sigma should be a dictionary of the required Verticies and the label
    Example: sigma = {a: v4 , b :v6}
    """
    self.labelled_verticies={}

```

```

        self.sigma = sigma
        for l,v in sigma.items():
            v.label = l
            self.labelled_vertices[self.Vertices.index(v)]=v
        self.unlabelled_vertices=self.Vertices[:]
        [self.unlabelled_vertices.remove(elem) for elem in
        ↪ self.labelled_vertices.values() ]

    def unlabel_all(self):
        for v in self.labelled_vertices.values():
            v.unlabel_vertex()

    def create_type(self):
        return induced_subgraph(self.labelled_vertices.values(), self)

class DiFlag(Flag):
    def __init__(self,V=[], E=[],sigma=None):
        super().__init__(V, E,sigma)
    """
    Miscellaneous functions
    """
    def nCk(n,k):
        return np.math.factorial(n) / (np.math.factorial(k)*np.math.factorial(n-k))

    def nPk(n,k):
        return np.math.factorial(n) / np.math.factorial(n-k)

    def find_key(value,dictionary):
        for k,v in dictionary.items():
            if v==value: return k

    def powerset(iterable):
        s = list(iterable)
        return itertools.chain.from_iterable(itertools.combinations(s, r) for r in
        ↪ range(len(s)+1))
    """
    The following section of code generates some common graphs:
    """

    def generate_complete_graph(n):
        V = [Vertex(name="v"+str(i)) for i in range(1,n+1)]

```

```

G=Graph(V)
for V1,V2 in itertools.combinations(G.Verticies,2):
    G.add_edge(V1,V2)
return G

def generate_path_graph(n):
    V = [Vertex(name="v"+str(i)) for i in range(1,n+1)]
    G=Graph(V)
    edges=[(V[i],V[i+1]) for i in range(n-1)]
    for e in edges:
        G.add_edge(e[0],e[1])
    return G

def generate_cycle_graph(n):
    V = [Vertex(name="v"+str(i)) for i in range(1,n+1)]
    G=Graph(V)
    edges=[(V[i],V[i+1]) for i in range(n-1)]
    for e in edges:
        G.add_edge(e[0],e[1])
    G.add_edge(V[0],V[-1])
    return G

def generate_random_graph(n,p):
    """
    n - no of edges
    p - probability
    This follows from the Erdos Renyi Model"""
    V = [Vertex(name="v"+str(i)) for i in range(1,n+1)]
    G=Graph(V)
    for V1,V2 in itertools.combinations(G.Verticies,2):
        if random.random() <= p:
            G.add_edge(V1,V2)
    return G

def generate_hyper_cube(n):
    V=[Vertex(name=''.join([str(y) for y in x])) for x in
    ↪ itertools.product([0,1],repeat=n)]
    G=Graph(V)
    def differAtOneBitPos( a , b ):
        def isPowerOfTwo( x ):
            return x and (not(x & (x - 1)))

```

```

        return isPowerOfTwo(a ^ b)
    for V1,V2 in itertools.combinations(V,2):
        if differAtOneBitPos(int(V1.name,2),int(V2.name,2)):
            G.add_edge(V1,V2)
    return G

def generate_complete_digraph(n):
    V = [Vertex(name="v"+str(i)) for i in range(1,n+1)]
    G=DiGraph(V)
    for V1,V2 in itertools.permutations(G.Verticies,2):
        G.add_edge(V1,V2)
    return G

def generate_dipath(n):
    V = [Vertex(name="v"+str(i)) for i in range(1,n+1)]
    G=DiGraph(V)
    edges=[(V[i],V[i+1]) for i in range(n-1)]
    for e in edges:
        G.add_edge(e[0],e[1])
    return G

def generate_directed_cycle(n):
    V = [Vertex(name="v"+str(i)) for i in range(1,n+1)]
    G=DiGraph(V)
    edges=[(V[i],V[i+1]) for i in range(n-1)]
    for e in edges:
        G.add_edge(e[0],e[1])
    G.add_edge(V[-1],V[0])
    return G

"""
Core functions
The following functions are the computations for the values to generate the
→ Cauchy Schwartz Inequalities.
"""

def complement(H,G):
    l=H.Edges[:]
    l_c=[x for x in G.Edges if x not in l]
    return Graph(H.Verticies, l_c)

```

```

def generate_adjacency_matrix(G):
    if isinstance(G,Flag) or isinstance(G,DiFlag):
        verts = list(G.labelled_vertices.values()) +
        → list(G.unlabelled_vertices)[:]#This basically fixes the labelled
        → vertices at the start for the adjacency matrix
    else:
        verts = list(G.Verticies) #Bugfix - use list
    n=len(verts)
    M=np.zeros((n,n),dtype=int)
    if isinstance(G,DiGraph):
        for v,u in list(G.Edges):
            M[verts.index(v),verts.index(u)] = 1
    else:
        for v,u in list(G.Edges):
            M[verts.index(u),verts.index(v)] = 1
            M[verts.index(v),verts.index(u)] = 1
    return M

def induced_subgraph(A,G):
    """
    A - The list of verticies composing of the subgraph.
    G - The graph
    """
    if not set(A).issubset(set(G.Verticies)):
        raise Exception("A contains an invalid vertex")
    else:
        A=list(A)
        [v.clear_neighbourhood() for v in A]
        if isinstance(G,Flag):
            H=Flag(V=A,sigma=G.sigma)
        else:
            H=Graph(V=A)
        for e in G.Edges:
            if e[0] in A and e[1] in A:
                H.add_edge(e[0],e[1])
        return H

def isIsomorphic(A,B):
    """
    Checks if Graphs A and B are isomorphic by comparing permutations of adjacency
    → matrix.

```



```

"""
A_M = generate_adjacency_matrix(A)
B_M = generate_adjacency_matrix(B)
n=len(A_M)
m=len(B_M)
if n!=m or len(A.Edges)!=len(B.Edges):
    return False
if Counter(A.degrees().values()) != Counter(B.degrees().values()):
    return False
for it in itertools.permutations(range(n)):
    P=np.array([[1 if j ==i else 0 for j in range(n)] for i in it]) #
    ↪ Permtuation matrix generated
    if np.array_equal(A_M ,np.dot(np.dot(P, B_M), P.T)):
        return True
return False

def isFlagIsomorphic(F1,F2):
    """
    This compares if 2 Flags are isomorphic.
    This fixes the types in the adjacency matrix and checks the permutations of
    ↪ the remaining combinations.
    """
    F1_M = generate_adjacency_matrix(F1)
    F2_M = generate_adjacency_matrix(F2)
    if isIsomorphic(F1.create_type(),F2.create_type()):
        n=len(F1_M)
        m=len(F2_M)
        s=len(F1.create_type())
        if n!=m or len(F1.Edges)!=len(F2.Edges):
            return False
        if Counter(F1.degrees().values()) != Counter(F2.degrees().values()):
            return False
        for it in itertools.permutations(range(s,n)): #Requires labelling in the
            ↪ same order
            P=np.array([[1 if j ==i else 0 for j in range(n)] for i in
                ↪ list(range(s))+list(it)]] # Permtuation matrix generated

            if np.array_equal(F1_M ,np.dot(np.dot(P, F2_M), P.T)):
                return True
    return False

```

```

def edge_density(G):
    """
    This calculates the edge density of a graph, G
    """
    n=len(G)
    return (2.*len(G.Edges))/ (n*(n-1))

def induced_homomorphism_density(A,G):
    """
    This function calculates the induced homomorphism density of a graph A in graph
    ↪ G.
    This works by taking all k tuples of G and taking the subgraph and checking
    ↪ isomorphism.
    """
    k=len(A)
    n=len(G)
    G_v = G.Verticies[:]
    c=0 # the counter
    for k_tup in itertools.combinations(G_v,k):
        H=induced_subgraph(k_tup,G)
        if isIsomorphic(A,H):
            c+=1
    return c/nCk(n,k)

def flag_density(F,G,theta):
    """
    F - Flag
    G - Graph
    Theta { which maps the labelled vertices of F to G
    """
    k=len(F)
    s=len(F.sigma)
    n=len(G)
    G_flag = Flag( V=G.Verticies , E = G.Edges, sigma = theta)
    G_v = G_flag.unlabelled_verticies[:]
    c=0 # the counter
    for k_tup in itertools.combinations(G_v,k-s):
        ↪ H=induced_subgraph(list(G_flag.labelled_verticies.values())+list(k_tup),G_flag)
        if isFlagIsomorphic(F,H):
            c+=1

```

```

    return c/nCk(n-s,k-s)

def two_flag_density(F_i,F_j,G,theta):
    """
    F_i , F_j - Flags of the same size and with the same type and they fit in G.
    G - Graph
    Theta { which maps the labelled vertices of F to G
    """
    k_1=len(F_i)
    k_2 = len(F_j)
    s=len(F_i.sigma)
    G_flag = Flag( V=G.Vertices , E = G.Edges, sigma = theta)
    G_v = G_flag.unlabelled_vertices[:]
    c=0 # the counter
    total=0
    for k_tup in itertools.permutations(G_v,k_1 + k_2- 2*s):
        total+=1

        → H1=induced_subgraph(list(G_flag.labelled_vertices.values())+list(k_tup)[:k_1
        → - s],G_flag)

        → H2=induced_subgraph(list(G_flag.labelled_vertices.values())+list(k_tup)[k_1
        → - s:],G_flag)
        if isFlagIsomorphic(F_i,H1) and isFlagIsomorphic(F_j, H2):
            c+=1
    return c/total

def E_theta(F_i,F_j,G):
    labels=list(F_i.sigma.keys())
    s= len(labels)
    n=len(G)
    E_value=0
    for vertices in itertools.permutations(G.Vertices,s):
        theta=dict(zip(labels,list(vertices)))
        E_value+= two_flag_density(F_i,F_j,G,theta)
    return E_value / nPk(n,s)

"""
More generators
"""

```

```

def generate_isomorphic_graphs_n_e(n,e):
    """
    Generates all graphs up to isomorphism on n vertices with e edges
    """
    V = [Vertex(name="v"+str(i)) for i in range(1,n+1)]
    unique_graphs=[]
    for edges in itertools.combinations(list(itertools.combinations(V,2)),e):
        G=Graph(V,E=[])
        for edge in edges:
            G.add_edge(edge[0],edge[1])
        if True in list(map(lambda x: isIsomorphic(x,G),unique_graphs)):
            pass
        else:
            unique_graphs.append(G)
    return unique_graphs

def generate_all_isomorphisms(n):
    graphs=[]
    E=nCk(n,2)
    for e in range(int(E+1)):
        V = [Vertex(name="v"+str(i)) for i in range(1,n+1)]
        unique_graphs=[]
        for edges in itertools.combinations(list(itertools.combinations(V,2)),e):
            G=Graph(V,E=[])
            for edge in edges:
                G.add_edge(edge[0],edge[1])
            if True in list(map(lambda x: isIsomorphic(x,G),unique_graphs)):
                pass
            else:
                unique_graphs.append(G)
        graphs+=unique_graphs
    return graphs

def generate_subcubes(n):
    cube=generate_hyper_cube(n)
    Gs=[]
    i=0
    for edges in itertools.combinations(cube.Edges,5):
        i+=1
        if i%10==0: print(i)

```

```

    G=Graph(cube.Verticies)
    for edge in edges:
        G.add_edge(edge[0],edge[1])
    if True in list(map(lambda x: isIsomorphic(x,G),Gs)):
        pass
    else:
        print(G.Edges)
        Gs.append(G)
return Gs

def generate_all_flags_isomorphisms(n,txt):
    "Not the best function but it works { Do not call and is not generalized copy
    ↪ and use when needed"
    graphs=[]
    f=open(txt,'w')
    k3=generate_complete_graph(3)
    i=0
    for e in range(int(E+1)):
        V = [Vertex(name="v"+str(i)) for i in range(1,n+1)]
        sigma = {'a':V[0], 'b':V[1] , 'c': V[2]}
        unique_graphs=[]
        for edges in itertools.combinations(list(itertools.combinations(V,2)),e):
            if (V[0],V[2]) in edges or (V[1],V[2]) in edges or (V[0],V[1]) in
            ↪ edges:
                continue
            i+=1
            G=Flag(V,E=[],sigma=sigma)
            for edge in edges:
                G.add_edge(edge[0],edge[1])
            H=Graph(G.Verticies,G.Edges)
            if True in list(map(lambda x: isFlagIsomorphic(x,G),unique_graphs)):
                pass
            elif not H.contains_copy(k3):
                unique_graphs.append(G)
                f.write(str(G.Verticies) +str(G.Edges) +"\n")
                print(G.Verticies,G.Edges)
        graphs+=unique_graphs
    return graphs

```

6.2 Code for K4

6.2.1 Generating c_H terms: Python

```
from main import *
"""
Creating Graphs
"""
K3 = generate_complete_graph(3)

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
G1 = Graph(V)

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
G2 = Graph(V)
G2.add_edge(G2.Verticies[0], G2.Verticies[1])

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
G3 = Graph(V)
G3.add_edge(G3.Verticies[0], G3.Verticies[1])
G3.add_edge(G3.Verticies[1], G3.Verticies[2])

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
G4 = Graph(V)
G4.add_edge(G4.Verticies[0], G4.Verticies[1])
G4.add_edge(G4.Verticies[2], G4.Verticies[3])

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
G5 = Graph(V)
G5.add_edge(G5.Verticies[0], G5.Verticies[1])
G5.add_edge(G5.Verticies[1], G5.Verticies[2])
G5.add_edge(G5.Verticies[0], G5.Verticies[2])

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
G6 = Graph(V)
G6.add_edge(G6.Verticies[0], G6.Verticies[1])
G6.add_edge(G6.Verticies[0], G6.Verticies[2])
G6.add_edge(G6.Verticies[0], G6.Verticies[3])

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
```

```

G7 = Graph(V)
G7.add_edge(G7.Verticies[0], G7.Verticies[1])
G7.add_edge(G7.Verticies[0], G7.Verticies[2])
G7.add_edge(G7.Verticies[2],G7.Verticies[3])

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
G8 = Graph(V)
G8.add_edge(G8.Verticies[0], G8.Verticies[1])
G8.add_edge(G8.Verticies[0], G8.Verticies[2])
G8.add_edge(G8.Verticies[0],G8.Verticies[3])
G8.add_edge(G8.Verticies[1],G8.Verticies[3])

G9 = generate_cycle_graph(4)

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
G10 = Graph(V)
G10.add_edge(G10.Verticies[0], G10.Verticies[1])
G10.add_edge(G10.Verticies[0], G10.Verticies[2])
G10.add_edge(G10.Verticies[0],G10.Verticies[3])
G10.add_edge(G10.Verticies[1],G10.Verticies[3])
G10.add_edge(G10.Verticies[1],G10.Verticies[2])

"""
Creating Flags
"""

V = [Vertex(name="v"+str(i)) for i in range(1,4)]
f1 = Graph(V)
f1.add_edge(f1.Verticies[0] , f1.Verticies[1])
sigma=dict([ ("12"[i],f1.Verticies[i]) for i in range(2)])
f1=Flag(f1.Verticies,f1.Edges,sigma)
print(f1.Verticies,f1.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,4)]
f2 = Graph(V)
f2.add_edge(f2.Verticies[0] , f2.Verticies[1])
f2.add_edge(f2.Verticies[1] , f2.Verticies[2])
sigma=dict([ ("12"[i],f2.Verticies[i]) for i in range(2)])
f2=Flag(f2.Verticies,f2.Edges,sigma)
print(f2.Verticies,f2.Edges)

```

```

V = [Vertex(name="v"+str(i)) for i in range(1,4)]
f3 = Graph(V)
f3.add_edge(f3.Verticies[0] , f3.Verticies[1])
f3.add_edge(f3.Verticies[0] , f3.Verticies[2])
sigma=dict([ ("12"[i],f3.Verticies[i]) for i in range(2)])
f3=Flag(f3.Verticies,f3.Edges,sigma)
print(f3.Verticies,f3.Edges)

f4 = generate_complete_graph(3)
sigma=dict([ ("12"[i],f4.Verticies[i]) for i in range(2)])
f4=Flag(f4.Verticies,f4.Edges,sigma)
print(f4.Verticies,f4.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,4)]
y1 = Graph(V)
sigma=dict([ ("12"[i],y1.Verticies[i]) for i in range(2)])
y1=Flag(y1.Verticies,y1.Edges,sigma)
print(y1.Verticies,y1.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,4)]
y2 = Graph(V)
y2.add_edge(y2.Verticies[1] , y2.Verticies[2])
sigma=dict([ ("12"[i],y2.Verticies[i]) for i in range(2)])
y2=Flag(y2.Verticies,y2.Edges,sigma)
print(y2.Verticies,y2.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,4)]
y3 = Graph(V)
y3.add_edge(y3.Verticies[0] , y3.Verticies[2])
sigma=dict([ ("12"[i],y3.Verticies[i]) for i in range(2)])
y3=Flag(y3.Verticies,y3.Edges,sigma)
print(y3.Verticies,y3.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,4)]
y4 = Graph(V)
y4.add_edge(y4.Verticies[0] , y4.Verticies[2])
y4.add_edge(y4.Verticies[1] , y4.Verticies[2])
sigma=dict([ ("12"[i],y4.Verticies[i]) for i in range(2)])

```



```

y4=Flag(y4.Verticies,y4.Edges,sigma)
print(y4.Verticies,y4.Edges)

G_list = [G1,G2,G3,G4,G5,G6,G7,G8,G9,G10]

print("Starting Computation")
for i in range(len(G_list)):
    G=G_list[i]
    print('G{}: {}'.format(i+1,G))
    print("Density of K3 in G{}
    ↪ is:".format(i+1),induced_homomorphism_density(K3,G))
    for F1_index,F2_index in itertools.combinations_with_replacement(range(4),2):
        # print("F"+str(F1_index+1), "F"+str(F2_index+1))
        F1= [f1,f2,f3,f4] [F1_index]
        F2= [f1,f2,f3,f4] [F2_index]
        if E_theta(F1,F2,G)!=0.0:
            if F1 != F2:
                print(12*2*E_theta(F1,F2,G),"q{0}{1} in graph
                ↪ G{2}".format(F1_index+1,F2_index+1,i+1) )
            else:
                print(12*E_theta(F1,F2,G),"q{0}{1} in graph
                ↪ G{2}".format(F1_index+1,F2_index+1,i+1) )
    for F1_index,F2_index in itertools.combinations_with_replacement(range(4),2):
        # print("F"+str(F1_index+1), "F"+str(F2_index+1))
        F1= [y1,y2,y3,y4] [F1_index]
        F2= [y1,y2,y3,y4] [F2_index]
        if E_theta(F1,F2,G)!=0.0:
            if F1 != F2:
                print(12*2*E_theta(F1,F2,G),"p{0}{1} in graph
                ↪ G{2}".format(F1_index+1,F2_index+1,i+1) )
            else:
                print(12*E_theta(F1,F2,G),"p{0}{1} in graph
                ↪ G{2}".format(F1_index+1,F2_index+1,i+1) )
    print("\n")

```

6.2.2 Code for Semidefinite Optimization: Mathematica

```

ClearAll[q11, q12, q13, q14, q22, q23, q24, q33, q34, q44, b, d]
ClearAll[p11, p12, p13, p14, p22, p23, p24, p33, p34, p44]
ClearAll[e1, e2, e3, e4, e5, e6, e7, e8, e9, b, e10]
Q = {{q11, q12, q13, q14}, {q12, q22, q23, q24}, {q13, q23, q33,

```

```

    q34} , {q14, q24, q34, q44}};
P = {{p11, p12, p13, p14}, {p12, p22, p23, p24}, {p13, p23, p33,
    p34} , {p14, p24, p34, p44}};
e10 = 1/2 + (4 q24 + 4 q34 + 2 q44 + 2 p44)/12 ;
e9 = (8 q23 + 4 p44)/12;
e8 = 1/4 + 1/12 (2 p24 + 2 p34 + 2 q14 + q22 + 2 q24 + q33 + 2 q34) ;
e7 = 1/12 (2 p23 + 2 p24 + 2 p34 + 2 q12 + 2 q13 + 2 q23) ;
e6 = p14/2 + 1/12 (3 q22 + 3 q33);
e5 = 1/4 + 1/12 (3 p22 + 3 p33) + q14/2 ;
e4 = (2 p23)/3 + q11/3 ;
e3 = 1/12 (2 p12 + 2 p13 + 2 p14 + p22 + p33) +
    1/12 (2 q12 + 2 q13) ;
e2 = 1/12 (2 p11 + 4 p12 + 4 p13) + q11/6 ;
e1 = p11 ;
res = SemidefiniteOptimization[
    d , {e1 <= d , e2 <= d, e3 <= d, e4 <= d, e5 <= d, e6 <= d, e7 <= d,
        e8 <= d, e9 <= d, e10 <= d, Q >= 0, P >= 0}, {q11, q12, q13, q14, q22, q23, q24,
        q33, q34, q44, p11, p12, p13, p14, p22, p23, p24, p33, p34, p44,
        d}]
ReplaceAll[{e1, e2, e3, e4, e5, e6, e7, e8, e9, e10} , res];

```

6.3 Code for Pentagon Problem

6.3.1 Generating c_H terms: Python

```

from main import *
import time
start_time = time.time()

V = [Vertex(name="v"+str(i)) for i in range(1,6)]
G1 = Graph(V)
print("G1",G1.Verticies,G1.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,6)]
G2 = Graph(V)
G2.add_edge(V[0],V[1])
print("G2",G2.Verticies,G2.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,6)]
G3 = Graph(V)
G3.add_edge(V[0],V[1])
G3.add_edge(V[0],V[4])

```

```

print("G3",G3.Verticies,G3.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,6)]
G4 = Graph(V)
G4.add_edge(V[0],V[1])
G4.add_edge(V[0],V[4])
G4.add_edge(V[0],V[2])
print("G4",G4.Verticies,G4.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,6)]
G5 = Graph(V)
G5.add_edge(V[0],V[1])
G5.add_edge(V[0],V[4])
G5.add_edge(V[0],V[2])
G5.add_edge(V[0],V[3])
print("G5",G5.Verticies,G5.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,6)]
G6 = Graph(V)
G6.add_edge(V[1],V[2])
G6.add_edge(V[0],V[4])
print("G6",G6.Verticies,G6.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,6)]
G7 = Graph(V)
G7.add_edge(V[0],V[1])
G7.add_edge(V[0],V[4])
G7.add_edge(V[2],V[3])
print("G7",G7.Verticies,G7.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,6)]
G8 = Graph(V)
G8.add_edge(V[0],V[1])
G8.add_edge(V[0],V[4])
G8.add_edge(V[1],V[2])
print("G8",G8.Verticies,G8.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,6)]
G9 = Graph(V)
G9.add_edge(V[0],V[1])
G9.add_edge(V[0],V[4])
G9.add_edge(V[1],V[2])
G9.add_edge(V[0],V[3])

```

```

print("G9",G9.Verticies,G9.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,6)]
G10 = Graph(V)
G10.add_edge(V[0],V[1])
G10.add_edge(V[0],V[4])
G10.add_edge(V[1],V[2])
G10.add_edge(V[2],V[4])
print("G10",G10.Verticies,G10.Edges)


V = [Vertex(name="v"+str(i)) for i in range(1,6)]
G11 = Graph(V)
G11.add_edge(V[0],V[1])
G11.add_edge(V[0],V[4])
G11.add_edge(V[1],V[2])
G11.add_edge(V[2],V[4])
G11.add_edge(V[3],V[4])
print("G11",G11.Verticies,G11.Edges)


V = [Vertex(name="v"+str(i)) for i in range(1,6)]
G12 = Graph(V)
G12.add_edge(V[0],V[1])
G12.add_edge(V[0],V[4])
G12.add_edge(V[1],V[2])
G12.add_edge(V[2],V[4])
G12.add_edge(V[3],V[4])
G12.add_edge(V[1],V[3])
print("G12",G12.Verticies,G12.Edges)


V = [Vertex(name="v"+str(i)) for i in range(1,6)]
G13 = Graph(V)
G13.add_edge(V[0],V[1])
G13.add_edge(V[0],V[4])
G13.add_edge(V[1],V[2])
G13.add_edge(V[3],V[4])
print("G13",G13.Verticies,G13.Edges)


V = [Vertex(name="v"+str(i)) for i in range(1,6)]
G14 = Graph(V)
G14.add_edge(V[0],V[1])
G14.add_edge(V[0],V[4])
G14.add_edge(V[1],V[2])
G14.add_edge(V[3],V[4])

```

```

G14.add_edge(V[2],V[3])
print("G14",G14.Verticies,G14.Edges)
print("\n")

G_list = [G1,G2,G3,G4,G5,G6,G7,G8,G9,G10,G11,G12,G13,G14]

C5 = generate_cycle_graph(5)

print("Flags - sigma 0")

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
f1 = Graph(V)
sigma=dict([ ("123"[i],f1.Verticies[i]) for i in range(3)])
f1=Flag(f1.Verticies,f1.Edges,sigma)
print("f1",f1.Verticies,f1.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
f2 = Graph(V)
sigma=dict([ ("123"[i],f2.Verticies[i]) for i in range(3)])
f2.add_edge(V[0],V[3])
f2=Flag(f2.Verticies,f2.Edges,sigma)
print("f2",f2.Verticies,f2.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
f3 = Graph(V)
sigma=dict([ ("123"[i],f3.Verticies[i]) for i in range(3)])
f3.add_edge(V[1],V[3])
f3=Flag(f3.Verticies,f3.Edges,sigma)
print("f3",f3.Verticies,f3.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
f4 = Graph(V)
sigma=dict([ ("123"[i],f4.Verticies[i]) for i in range(3)])
f4.add_edge(V[0],V[3])
f4.add_edge(V[1],V[3])
f4=Flag(f4.Verticies,f4.Edges,sigma)
print("f4",f4.Verticies,f4.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
f5 = Graph(V)
sigma=dict([ ("123"[i],f5.Verticies[i]) for i in range(3)])
f5.add_edge(V[2],V[3])
f5=Flag(f5.Verticies,f5.Edges,sigma)
print("f5",f5.Verticies,f5.Edges)

```

```

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
f6 = Graph(V)
sigma=dict([ ("123"[i],f6.Verticies[i]) for i in range(3)])
f6.add_edge(V[0],V[3])
f6.add_edge(V[2],V[3])
f6=Flag(f6.Verticies,f6.Edges,sigma)
print("f6",f6.Verticies,f6.Edges)

```

```

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
f7 = Graph(V)
sigma=dict([ ("123"[i],f7.Verticies[i]) for i in range(3)])
f7.add_edge(V[1],V[3])
f7.add_edge(V[2],V[3])
f7=Flag(f7.Verticies,f7.Edges,sigma)
print("f7",f7.Verticies,f7.Edges)

```

```

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
f8 = Graph(V)
sigma=dict([ ("123"[i],f8.Verticies[i]) for i in range(3)])
f8.add_edge(V[0],V[3])
f8.add_edge(V[2],V[3])
f8.add_edge(V[1],V[3])
f8=Flag(f8.Verticies,f8.Edges,sigma)
print("f8",f8.Verticies,f8.Edges)

```

```

print("Flags - sigma 1")

```

```

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
y1 = Graph(V)
sigma=dict([ ("123"[i],y1.Verticies[i]) for i in range(3)])
y1.add_edge(V[0],V[1])
y1=Flag(y1.Verticies,y1.Edges,sigma)
print("Y1",y1.Verticies,y1.Edges)

```

```

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
y2 = Graph(V)
sigma=dict([ ("123"[i],y2.Verticies[i]) for i in range(3)])
y2.add_edge(V[0],V[3])
y2.add_edge(V[0],V[1])
y2=Flag(y2.Verticies,y2.Edges,sigma)
print("Y2",y2.Verticies,y2.Edges)

```

```

V = [Vertex(name="v"+str(i)) for i in range(1,5)]

```

```

y3 = Graph(V)
sigma=dict([ ("123"[i],y3.Verticies[i]) for i in range(3)])
y3.add_edge(V[1],V[3])
y3.add_edge(V[0],V[1])
y3=Flag(y3.Verticies,y3.Edges,sigma)
print("Y3",y3.Verticies,y3.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
y4 = Graph(V)
sigma=dict([ ("123"[i],y4.Verticies[i]) for i in range(3)])
y4.add_edge(V[2],V[3])
y4.add_edge(V[0],V[1])
y4=Flag(y4.Verticies,y4.Edges,sigma)
print("Y4",y4.Verticies,y4.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
y5 = Graph(V)
sigma=dict([ ("123"[i],y5.Verticies[i]) for i in range(3)])
y5.add_edge(V[0],V[3])
y5.add_edge(V[2],V[3])
y5.add_edge(V[0],V[1])
y5=Flag(y5.Verticies,y5.Edges,sigma)
print("Y5",y5.Verticies,y5.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
y6 = Graph(V)
sigma=dict([ ("123"[i],y6.Verticies[i]) for i in range(3)])
y6.add_edge(V[1],V[3])
y6.add_edge(V[2],V[3])
y6.add_edge(V[0],V[1])
y6=Flag(y6.Verticies,y6.Edges,sigma)
print("Y6",y6.Verticies,y6.Edges)

print(
    """
    Flags -- Sigma 2
    """)

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
z1 = Graph(V)
sigma=dict([ ("123"[i],z1.Verticies[i]) for i in range(3)])
z1.add_edge(V[0],V[1])
z1.add_edge(V[1],V[2])

```

```

z1=Flag(z1.Verticies,z1.Edges,sigma)
print("Z1",z1.Verticies,z1.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
z2 = Graph(V)
sigma=dict([ ("123"[i],z2.Verticies[i]) for i in range(3)])
z2.add_edge(V[0],V[3])
z2.add_edge(V[0],V[1])
z2.add_edge(V[1],V[2])
z2=Flag(z2.Verticies,z2.Edges,sigma)
print("Z2",z2.Verticies,z2.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
z3 = Graph(V)
sigma=dict([ ("123"[i],z3.Verticies[i]) for i in range(3)])
z3.add_edge(V[1],V[3])
z3.add_edge(V[0],V[1])
z3.add_edge(V[1],V[2])
z3=Flag(z3.Verticies,z3.Edges,sigma)
print("Z3",z3.Verticies,z3.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
z4 = Graph(V)
sigma=dict([ ("123"[i],z4.Verticies[i]) for i in range(3)])
z4.add_edge(V[2],V[3])
z4.add_edge(V[0],V[1])
z4.add_edge(V[1],V[2])
z4=Flag(z4.Verticies,z4.Edges,sigma)
print("Z4",z4.Verticies,z4.Edges)

V = [Vertex(name="v"+str(i)) for i in range(1,5)]
z5 = Graph(V)
sigma=dict([ ("123"[i],z5.Verticies[i]) for i in range(3)])
z5.add_edge(V[0],V[3])
z5.add_edge(V[2],V[3])
z5.add_edge(V[0],V[1])
z5.add_edge(V[1],V[2])
z5=Flag(z5.Verticies,z5.Edges,sigma)
print("Z5",z5.Verticies,z5.Edges)
print("\n")

for i in range(14):
    G=G_list[i]
    print("Density of C5 in G{} is:".format(i+1),induced_homomorphism_density(C5,G)*120)
    for F1_index,F2_index in itertools.combinations_with_replacement(range(8),2):

```



```

F1= [f1,f2,f3,f4,f5,f6,f7,f8][F1_index]
F2= [f1,f2,f3,f4,f5,f6,f7,f8][F2_index]

if E_theta(F1,F2,G)!=0.0 or False:
    if F1 != F2:
        print(120*2*E_theta(F1,F2,G),"p{0}-{1} in graph
        ↪ G{2}".format(F1_index+1,F2_index+1,i+1) )
    else:
        print(120*E_theta(F1,F2,G),"p{0}-{1} in graph
        ↪ G{2}".format(F1_index+1,F2_index+1,i+1) )

for F1_index,F2_index in itertools.combinations_with_replacement(range(6),2):
    F1= [y1,y2,y3,y4,y5,y6][F1_index]
    F2= [y1,y2,y3,y4,y5,y6][F2_index]

    if E_theta(F1,F2,G)!=0.0 or False:
        if F1 != F2:
            print(120*2*E_theta(F1,F2,G),"q{0}-{1} in graph
            ↪ G{2}".format(F1_index+1,F2_index+1,i+1) )

        else:
            print(120*E_theta(F1,F2,G),"q{0}-{1} in graph
            ↪ G{2}".format(F1_index+1,F2_index+1,i+1) )

for F1_index,F2_index in itertools.combinations_with_replacement(range(5),2):
    F1= [z1,z2,z3,z4,z5][F1_index]
    F2= [z1,z2,z3,z4,z5][F2_index]

    if E_theta(F1,F2,G)!=0.0 or False:
        if F1 != F2:
            print(120*2*E_theta(F1,F2,G),"r{0}-{1} in graph
            ↪ G{2}".format(F1_index+1,F2_index+1,i+1) )
        else:
            print(120*E_theta(F1,F2,G),"r{0}-{1} in graph
            ↪ G{2}".format(F1_index+1,F2_index+1,i+1) )

print("\n")
print("Executed in --- %s seconds ---" % (time.time() - start_time))

```

6.3.2 Code for Semidefinite Optimization: Mathematica

```

P = {{p11, p12, p13, p14, p15, p16, p17, p18}, {p12, p22, p23, p24,
p25, p26, p27, p28}, {p13, p23, p33, p34, p35, p36, p37,
p38}, {p14, p24, p34, p44, p45, p46, p47, p48} , {p15, p25, p35,
p45, p55, p56, p57, p58} , {p16, p26, p36, p46, p56, p66, p67,

```

```

    p68} , {p17, p27, p37, p47, p57, p67, p77, p78}, {p18, p28, p38,
    p48, p58, p68, p78, p88}} ;
Q = {{q11, q12, q13, q14, q15, q16}, {q12, q22, q23, q24, q25,
    q26}, {q13, q23, q33, q34, q35, q36}, {q14, q24, q34, q44, q45,
    q46}, {q15, q25, q35, q45, q55, q56}, {q16, q26, q36, q46, q56,
    q66}} ;
R = {{r11, r12, r13, r14, r15} , {r12, r22, r23, r24, r25}, {r13, r23,
    r33, r34, r35}, {r14, r24, r34, r44, r45}, {r15, r25, r35, r45,
    r55}} ;
Flatt = Union[DeleteDuplicates[Flatten[P]] ,
    DeleteDuplicates[Flatten[Q]] , DeleteDuplicates[Flatten[R]]] ; ;
e1 = 120 p11 /120;
e2 = (12 p11 + 24 p12 + 24 p13 + 24 p15 + 12 q11)/120;
e3 = (8 p12 + 8 p13 + 8 p14 + 8 p15 + 8 p16 + 8 p17 + 4 p22 + 4 p33 +
    4 p55 + 8 q12 + 8 q13 + 4 r11)/120;
e4 = (12 p14 + 12 p16 + 12 p17 + 12 p18 + 6 q22 + 6 q33 + 12 r13) /120;
e5 = (48 p18 + 24 r33)/120;
e6 = (16 p23 + 16 p25 + 16 p35 + 8 q11 + 16 q14 )/120 ;
e7 = (8 p27 + 8 p36 + 8 p45 + 8 q14 + 8 q24 + 8 q34 + 4 q44 + 4 r11)/
    120 ;
e8 = ( 4 p23 + 4 p24 + 4 p25 + 4 p26 + 4 p34 + 4 p35 + 4 p37 +
    4 p56 + 4 p57 + 4 q12 + 4 q13 + 4 q15 + 4 q16 + 4 q23 + 4 r12 +
    4 r14)/120;
e9 = (4 p27 + 4 p28 + 4 p36 + 4 p38 + 4 p45 + 4 p58 + 4 q15 +
    4 q16 + 4 q25 + 4 q36 + 4 r13 + 2 r22 + 4 r23 + 4 r34 + 2 r44)/
    120;
e10 = (8 p44 + 8 p66 + 8 p77 + 16 q23 + 16 r15 )/120;
e11 = (4 p48 + 4 p68 + 4 p78 + 4 q26 + 4 q35 + 2 q55 + 2 q66 +
    4 r15 + 4 r23 + 4 r25 + 4 r34 + 4 r35 + 4 r45 )/120;
e12 = (12 p88 + 24 r35 + 12 r55)/120 ;
e13 = (4 p46 + 4 p47 + 4 p67 + 4 q24 + 4 q26 + 4 q34 + 4 q35 +
    4 q45 + 4 q46 + 4 r12 + 4 r14 + 4 r24)/120;
e14 = ( 120 + 20 q56 + 20 r24 )/120;

res = SemidefiniteOptimization[
    d , {e1 <= d , e2 <= d, e3 <= d, e4 <= d , e5 <= d , e6 <= d,
    e7 <= d, e8 <= d, e9 <= d, e10 <= d, e11 <= d, e12 <= d ,
    e13 <= d, e14 <= d, P>= 0 , Q>= 0 ,R >=0
    , {p11, p12, p13, p14, p15, p16, p17,
    p18, p22, p23, p24, p25, p26, p27, p28, p33, p34, p35, p36, p37,
    p38, p44, p45, p46, p47, p48, p55, p56, p57, p58, p66, p67, p68,
    p77, p78, p88, q11, q12, q13, q14, q15, q16, q22, q23, q24, q25,
    q26, q33, q34, q35, q36, q44, q45, q46, q55, q56, q66, r11, r12,
    r13, r14, r15, r22, r23, r24, r25, r33, r34, r35, r44, r45, r55,
    d}] ;

```

```
ReplaceAll[{e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11, e12, e13,
e14} , res]
```

6.4 Code for C7

6.4.1 Computing $c_H(\sigma_1, 5, P)$

There are similar files for σ_0, σ_2 which can be found in the github repository. Similarly, all flags, graphs and text files can be found in the repository.

```
from main import *

import time
start_time = time.time()

Graphs=[]
C5=generate_cycle_graph(5)
C3=generate_cycle_graph(3)

print('Importing Graphs')
with open('g7_c3c5free.txt','r') as f:
    for line in f:
        x=line.strip('\n').split(';')
        x_v=x[1]
        l=x_v.replace('v','')
        l=eval(l)
        n=7
        V = [Vertex(name="v"+str(i)) for i in range(1,n+1)]
        G=Graph(V)
        if l!=[]:
            for e in l:
                G.add_edge(V[e[0]-1],V[e[1]-1])
            Graphs.append(G)
print(len(Graphs))

C7 = generate_cycle_graph(7)

print("""
Flags - sigma 1
""")

F_1=[]
n=5
```

```

with open('flag_5_3_1.txt','r') as f:
    for line in f:
        V = [Vertex(name="v"+str(i)) for i in range(1,n+1)]
        x=line.strip('\n').replace(' ','').split('xxx')
        x_v=x[1]
        l=x_v.replace('v','')
        l=l.replace('a','1').replace('b','2').replace('c','3')
        l=eval(l)
        sigma={'a':V[0],'b':V[1],'c':V[2]}
        F=Flag(V,sigma=sigma)
        if l!=[]:
            for e in l:
                F.add_edge(V[e[0]-1],V[e[1]-1])
            F_1.append(F)

output = open('c7_1.txt','w')

for i in range(1,len(Graphs)):

    start_time = time.time()
    G=Graphs[i]
    output.write(str(G.Edges)+'\n')
    print("Density of C7 in G{} is:".format(i+1),induced_homomorphism_density(C7,G)*210)
    for F1_index,F2_index in itertools.combinations_with_replacement(range(len(F_1)),2):
        # print("F"+str(F1_index+1),"F"+str(F2_index+1))
        F1= F_1[F1_index]
        F2= F_1[F2_index]
        if E_theta(F1,F2,G)!=0.0 or True:
            if F1 != F2:
                print(210*2*E_theta(F1,F2,G),"q_{0},{1} in graph
                ↪ G{2}".format(F1_index+1,F2_index+1,i+1) )
                output.write( str(210*2*E_theta(F1,F2,G))
                ↪ +'_q_{0},{1}_G{2}\n'.format(F1_index+1,F2_index+1,i+1))
            else:
                print(210*E_theta(F1,F2,G),"q_{0},{1} in graph
                ↪ G{2}".format(F1_index+1,F2_index+1,i+1) )
                output.write( str(210*E_theta(F1,F2,G))
                ↪ +'_q_{0},{1}_G{2}\n'.format(F1_index+1,F2_index+1,i+1))
    print(start_time -time.time())
    print('\n')
output.close()

```

6.4.2 c_H Inequalities

$e1 = 1/630 (0.0 + 630p1mm1)$
 $e2 = 1/630 (0.0+ 30 q1mm1);$
 $e3 = 1/630 (0.0+ 24 q1mm2 + 24 q1mm3 + 6 r1mm1);$
 $e4 = 1/630 (0.0+ 9 q1mm6 + 9 q1mm11 + 18 q2mm2 + 18 q3mm3 + 18 r1mm3);$
 $e5 = 1/630 (0.0+ 24 q2mm6 + 24 q3mm11 + 12 r1mm11 + 24 r3mm3);$
 $e6 = 1/630 (0.0+ 15 q6mm6 + 15 q11mm11 + 60 r3mm11);$
 $e7 = 1/630 (0.0+ 90 r11mm11);$
 $e8 = 1/630 (0.0+ 24 q1mm1 + 24 q1mm4 + 12 q1mm5);$
 $e9 = 1/630 (0.0+ 4 q1mm1 + 8 q1mm2 + 8 q1mm3 + 8 q1mm4 + 8 q1mm5 + 4 q1mm9 + 4 q1mm13 + 2 q1mm15 + 4 q1mm16 + 8 q2mm4 + 4 q2mm5 + 8 q3mm4 + 4 q3mm5 + 4 q4mm4 + 4 r1mm1 + 2 r1mm5);$
 $e10 = 1/630 (0.0+ 5 q1mm4 + 6 q1mm5 + 11 q1mm16 + 6 q2mm2 + 11 q2mm9 + 6 q3mm3 + 11 q3mm13 + 5 q4mm6 + 5 q4mm11 + 6 q4mm15 + 3 q5mm6 + 3 q5mm11 + 11 r1mm3 + 5 r3mm5);$
 $e11 = 1/630 (0.0+ 24 q1mm16 + 24 q6mm9 + 24 q11mm13 + 6 q15mm15 + 24 r3mm3 + 12 r5mm11);$
 $e12 = 1/630 (0.0+ 11 q1mm2 + 11 q1mm3 + 5 q1mm7 + 6 q1mm8 + 5 q1mm10 + 6 q1mm12 + 5 q1mm14 + 11 q2mm3 + 6 r1mm2 + 6 r1mm4);$
 $e13 = 1/630 (0.0+ 2 q1mm2 + 2 q1mm3 + 4 q1mm6 + 4 q1mm8 + 4 q1mm10 + 4 q1mm11 + 4 q1mm12 + 4 q1mm14 + 2 q1mm17 + 2 q1mm22 + 2 q1mm24 + 2 q1mm26 + 4 q2mm2 + 4 q2mm7 + 4 q2mm8 + 4 q2mm10 + 2 q2mm11 + 4 q3mm3 + 2 q3mm6 + 4 q3mm7 + 4 q3mm12 + 4 q3mm14 + 4 r1mm3 + 1 r1mm6 + 2 r1mm7 + 2 r1mm12 + 2 r1mm13 + 1 r1mm14 + 2 r2mm2 + 4 r2mm3 + 4 r3mm4 + 2 r4mm4);$
 $e14 = 1/630 (0.0+ 3 q1mm8 + 3 q1mm10 + 3 q1mm12 + 3 q1mm14 + 6 q1mm22 + 6 q1mm26 + 6 q2mm6 + 6 q2mm17 + 6 q3mm11 + 6 q3mm24 + 3 q6mm7 + 3 q6mm8 + 3 q6mm10 + 3 q7mm11 + 3 q11mm12 + 3 q11mm14 + 5 r1mm1 + 3 r2mm6 + 3 r2mm11 + 5 r3mm3 + 6 r3mm7 + 6 r3mm12 + 5 r3mm13 + 3 r4mm11 + 3 r4mm14);$
 $e15 = 1/630 (0.0+ 12 q1mm22 + 12 q1mm26 + 12 q6mm17 + 12 q11mm24 + 24 r3mm11 + 3 r6mm6 + 12 r7mm11 + 12 r11mm12 + 12 r11mm13 + 3 r14mm14);$
 $e16 = 1/630 (0.0+ 4 q1mm2 + 4 q1mm3 + 8 q1mm9 + 8 q1mm13 + 8 q2mm4 + 8 q2mm5 + 4 q2mm15 + 8 q2mm16 + 8 q3mm4 + 8 q3mm5 + 4 q3mm15 + 8 q3mm16 + 8 q4mm9 + 8 q4mm13 + 4 r1mm1 + 8 r1mm5);$
 $e17 = 1/630 (0.0+ 4 q1mm6 + 4 q1mm11 + 8 q1mm17 + 8 q1mm24 + 8 q2mm8 + 8 q2mm10 + 8 q2mm22 + 8 q3mm12 + 8 q3mm14 + 8 q3mm26 + 2 q6mm11 + 4 q7mm7 + 4 r1mm3 + 8 r1mm13 + 8 r2mm7 + 4 r3mm6 + 4 r3mm14 + 8 r4mm12);$
 $e18 = 1/630 (0.0+ 5 q1mm9 + 5 q1mm13 + 11 q2mm9 + 11 q2mm16 + 11 q3mm13 + 11 q3mm16 + 3 q6mm15 + 5 q6mm16 + 6 q9mm9 + 6 q9mm15 + 3 q11mm15 + 5 q11mm16 + 6 q13mm13 + 6 q13mm15 + 5 r1mm3 + 6 r1mm5 + 11 r3mm5);$
 $e19 = 1/630 (0.0+ 5 q1mm17 + 5 q1mm24 + 11 q2mm17 + 11 q2mm22 + 11 q3mm24 + 11 q3mm26 + 5 q6mm22 + 5 q11mm26 + 5 r1mm11 + 6 r1mm13 + 11 r3mm13 + 6 r6mm7 + 3 r6mm11 + 6 r7mm7 + 3 r11mm14 + 6 r12mm12 + 6 r12mm14);$
 $e20 = 1/630 (0.0+ 2 q1mm7 + 1 q1mm10 + 1 q1mm14 + 2 q1mm18 + 2 q1mm19 + 2 q1mm23 + 2 q1mm27 + 2 q2mm2 + 2 q2mm4 + 2 q2mm5 + 2 q2mm7 + 2 q2mm9 + 2 q2mm10 + 2 q2mm12 + 2 q2mm14 + 2 q2mm16 + 2 q2mm19 + 2 q2mm21 + 2 q2mm23 + 2 q2mm26 + 2 q3mm3 + 2 q3mm4 + 2 q3mm5 + 2 q3mm7 + 2 q3mm8 + 2 q3mm10 + 2 q3mm13 + 2 q3mm14 + 2 q3mm16 + 2 q3mm18 + 2 q3mm22 + 2 q3mm25 + 2 q3mm27 + 2 q4mm6 + 2 q4mm11 + 2 q4mm17 + 2 q4mm21 + 2 q4mm24 + 2 q4mm25 + 2 q5mm6 + 2 q5mm11 + 1 q6mm12 + 1 q6mm14 + 2 q8mm9 + 1 q8mm16 + 1 q10mm11 + 2 q12mm13 + 1 q12mm15 + 1 r1mm2 + 2 r1mm3 + 1 r1mm4 + 2 r1mm7 + 2 r1mm10 + 2 r1mm12 + 2 r1mm13 + 2 r1mm15 + 2 r2mm3 + 2 r2mm9 + 1 r2mm14 + 2 r3mm4 + 2 r3mm5 + 2 r3mm10 + 2 r3mm15 + 1 r4mm6 + 2 r4mm9);$
 $e21 = 1/630 (0.0+ 3 q1mm18 + 3 q1mm19 + 3 q1mm23 + 3 q1mm27 + 6 q2mm16 + 6 q2mm26 + 6 q3mm16 + 6 q3mm22 + 6 q6mm9 + 3 q6mm19 + 3 q6mm21 + 3 q6mm23 + 6 q9mm17 + 6 q11mm13 + 3 q11mm18 + 3 q11mm25 + 3 q11mm27 + 6 q13mm24 + 3 q15mm21 + 3 q15mm25 + 3 r1mm10 + 3 r1mm15 + 5 r3mm3 + 6 r3mm7 + 6 r3mm12 + 5 r3mm13 + 5 r5mm11 + 3 r6mm9 + 3 r9mm14 + 3 r10mm11 + 3 r11mm15);$
 $e22 = 1/630 (0.0+ 23 q1mm20 + 47 q2mm3 + 24 r1mm8);$
 $e23 = 1/630 (0.0+ 2 q1mm6 + 2 q1mm11 + 8 q1mm20 + 1 q1mm28 + 2 q1mm30 + 2 q1mm31 + 1 q1mm32 + 8 q2mm3 + 4 q2mm7 + 4 q2mm10 + 4 q2mm11 + 4 q2mm12 + 4 q2mm14 + 4 q2mm20 + 4 q3mm6 + 4 q3mm7 + 4 q3mm8 + 4 q3mm10 + 4 q3mm14 + 4 q3mm20 + 2 q8mm8 + 2 q12mm12 + 4 r1mm8 + 2 r1mm16 + 2 r1mm17 + 2 r1mm18 + 2 r1mm19 + 2 r1mm20 + 2 r1mm22 + 4 r2mm3 + 4 r2mm8 + 4$

$$r3mm4 + 4 r3mm8 + 4 r4mm8);$$

$$e24 = 1/630 (0.0 + 4 q1mm20 + 4 q1mm30 + 4 q1mm31 + 2 q2mm6 + 4 q2mm12 + 4 q2mm14 + 4 q2mm26 + 2 q2mm28 + 4 q3mm8 + 4 q3mm10 + 2 q3mm11 + 4 q3mm22 + 2 q3mm32 + 4 q6mm7 + 4 q6mm10 + 2 q6mm20 + 4 q7mm11 + 4 q8mm17 + 4 q11mm14 + 2 q11mm20 + 4 q12mm24 + 2 r1mm8 + 2 r1mm11 + 4 r1mm20 + 4 r2mm11 + 4 r2mm16 + 4 r3mm7 + 4 r3mm12 + 4 r3mm13 + 4 r3mm17 + 4 r3mm18 + 4 r3mm22 + 4 r4mm11 + 4 r4mm19 + 2 r6mm8 + 2 r8mm11 + 2 r8mm14);$$

$$e25 = 1/630 (0.0 + 5 q1mm30 + 5 q1mm31 + 11 q2mm26 + 11 q3mm22 + 3 q6mm28 + 3 q11mm32 + 6 q17mm17 + 6 q24mm24 + 6 r1mm20 + 5 r3mm11 + 6 r6mm16 + 11 r7mm11 + 11 r11mm12 + 11 r11mm13 + 6 r11mm17 + 5 r11mm18 + 5 r11mm22 + 6 r14mm19);$$

$$e26 = 1/630 (0.0 + 8 q2mm9 + 8 q2mm19 + 8 q2mm23 + 8 q3mm13 + 8 q3mm18 + 8 q3mm27 + 8 q6mm16 + 4 q6mm26 + 8 q9mm21 + 8 q11mm16 + 4 q11mm22 + 8 q13mm25 + 4 q15mm17 + 4 q15mm24 + 4 r1mm7 + 4 r1mm12 + 4 r1mm13 + 8 r3mm5 + 8 r3mm10 + 8 r3mm15 + 2 r6mm14 + 4 r9mm9);$$

$$e27 = 1/630 (0.0 + 4 q2mm7 + 4 q2mm10 + 4 q2mm19 + 4 q2mm20 + 4 q2mm23 + 4 q2mm31 + 4 q3mm7 + 4 q3mm14 + 4 q3mm18 + 4 q3mm20 + 4 q3mm27 + 4 q3mm30 + 2 q4mm6 + 2 q4mm11 + 2 q4mm28 + 2 q4mm32 + 2 q5mm6 + 2 q5mm11 + 4 q6mm12 + 4 q6mm14 + 4 q8mm11 + 4 q8mm21 + 4 q10mm11 + 4 q12mm25 + 4 r1mm17 + 4 r1mm18 + 4 r1mm22 + 4 r2mm3 + 4 r2mm19 + 4 r3mm4 + 4 r3mm8 + 4 r3mm10 + 4 r3mm15 + 4 r3mm20 + 4 r4mm16 + 4 r8mm9);$$

$$e28 = 1/630 (0.0 + 4 q2mm19 + 4 q2mm23 + 4 q2mm31 + 4 q3mm18 + 4 q3mm27 + 4 q3mm30 + 2 q6mm9 + 2 q6mm16 + 4 q6mm19 + 4 q6mm23 + 4 q6mm26 + 2 q6mm31 + 2 q9mm28 + 2 q11mm13 + 2 q11mm16 + 4 q11mm18 + 4 q11mm22 + 4 q11mm27 + 2 q11mm30 + 2 q13mm32 + 1 q15mm28 + 1 q15mm32 + 4 q17mm21 + 2 q21mm21 + 4 q24mm25 + 2 q25mm25 + 2 r1mm17 + 2 r1mm18 + 2 r1mm22 + 4 r3mm7 + 4 r3mm10 + 4 r3mm12 + 4 r3mm13 + 4 r3mm15 + 4 r3mm17 + 4 r3mm18 + 4 r3mm20 + 4 r3mm22 + 2 r5mm11 + 2 r6mm19 + 4 r9mm16 + 4 r9mm19 + 4 r10mm11 + 4 r11mm15 + 2 r11mm20 + 2 r14mm16);$$

$$e29 = 1/630 (0.0 + 12 q2mm11 + 24 q2mm20 + 12 q3mm6 + 24 q3mm20 + 5 r1mm23 + 12 r1mm24 + 24 r3mm8 + 11 r8mm8);$$

$$e30 = 1/630 (0.0 + 6 q2mm20 + 6 q2mm31 + 6 q3mm20 + 6 q3mm30 + 3 q6mm7 + 3 q6mm10 + 3 q6mm12 + 3 q6mm14 + 6 q6mm20 + 3 q7mm11 + 3 q8mm11 + 3 q8mm28 + 3 q10mm11 + 3 q11mm14 + 6 q11mm20 + 3 q12mm32 + 5 r1mm24 + 3 r2mm11 + 3 r2mm23 + 5 r3mm8 + 5 r3mm17 + 6 r3mm18 + 5 r3mm20 + 6 r3mm22 + 5 r3mm24 + 3 r4mm11 + 3 r4mm23 + 5 r8mm11 + 6 r8mm16 + 6 r8mm19);$$

$$e31 = 1/630 (0.0 + 11 q2mm31 + 11 q3mm30 + 5 q6mm26 + 5 q11mm22 + 6 q17mm28 + 6 q24mm32 + 5 r1mm24 + 11 r3mm20 + 3 r6mm23 + 5 r7mm11 + 5 r11mm12 + 6 r11mm13 + 11 r11mm17 + 11 r11mm18 + 11 r11mm22 + 5 r11mm24 + 3 r14mm23 + 6 r16mm16 + 6 r19mm19);$$

$$e32 = 1/630 (0.0 + 5 q6mm19 + 6 q6mm23 + 11 q6mm31 + 5 q11mm18 + 6 q11mm27 + 11 q11mm30 + 6 q21mm28 + 6 q25mm32 + 12 r3mm17 + 11 r3mm18 + 11 r3mm22 + 12 r3mm24 + 5 r9mm23 + 5 r10mm11 + 5 r11mm15 + 12 r11mm20 + 11 r16mm19);$$

$$e33 = 1/630 (0.0 + 24 q6mm20 + 24 q11mm20 + 47 r3mm24 + 23 r8mm11 + 24 r8mm23);$$

$$e34 = 1/630 (0.0 + 12 q6mm31 + 12 q11mm30 + 3 q28mm28 + 3 q32mm32 + 24 r3mm24 + 12 r11mm17 + 12 r11mm18 + 12 r11mm20 + 12 r11mm22 + 24 r11mm24 + 12 r16mm23 + 12 r19mm23);$$

$$e35 = 1/630 (0.0 + 120 r11mm24 + 30 r23mm23);$$

$$e36 = 1/630 (0.0 + 11 q1mm1 + 47 q1mm4 + 23 q4mm5 + 5 q5mm5);$$

$$e37 = 1/630 (0.0 + 24 q1mm4 + 16 q2mm4 + 16 q3mm4 + 8 q4mm4 + 16 q4mm5 + 8 q5mm9 + 8 q5mm13 + 4 q5mm15 + 8 q5mm16 + 4 r1mm1 + 2 r5mm5);$$

$$e38 = 1/630 (0.0 + 2 q1mm1 + 4 q1mm2 + 4 q1mm3 + 4 q1mm4 + 2 q1mm5 + 4 q1mm8 + 4 q1mm9 + 4 q1mm12 + 4 q1mm13 + 4 q1mm15 + 4 q1mm16 + 4 q2mm3 + 4 q2mm4 + 4 q2mm13 + 4 q3mm4 + 4 q3mm9 + 4 q4mm4 + 4 q4mm5 + 4 q4mm7 + 4 q4mm10 + 4 q4mm14 + 4 q4mm16 + 2 q5mm5 + 2 q5mm7 + 2 q5mm8 + 2 q5mm10 + 2 q5mm12 + 2 q5mm14 + 4 r1mm2 + 4 r1mm4 + 2 r2mm5 + 2 r4mm5);$$

$$e39 = 1/630 (0.0 + 4 q1mm4 + 4 q1mm8 + 2 q1mm9 + 4 q1mm12 + 2 q1mm13 + 2 q1mm15 + 5 q1mm16 + 4 q2mm8 + 4 q2mm9 + 4 q3mm12 + 4 q3mm13 + 4 q4mm5 + 4 q4mm6 + 4 q4mm10 + 4 q4mm11 + 4 q4mm14 + 4 q4mm15 + 4 q4mm16 + 4 q5mm16 + 2 q5mm17 + 2 q5mm22 + 2 q5mm24 + 2 q5mm26 + 2 q6mm13 + 4 q7mm9 + 4 q7mm13 + 4 q9mm10 + 2 q9mm11 + 4 q13mm14 + 2 q15mm16 + 4 r1mm3 + 2 r2mm2 + 4 r2mm3 + 4 r3mm4 + 2 r4mm4 + 1 r5mm6 + 2 r5mm7 + 2 r5mm12 + 2 r5mm13 + 1 r5mm14);$$

$$e40 = 1/630 (0.0 + 4 q1mm2 + 4 q1mm3 + 8 q1mm7 + 4 q1mm10 + 4 q1mm14 + 2 q1mm18 + 2 q1mm19 + 2 q1mm21 + 2 q1mm23 + 2 q1mm25 + 2 q1mm27 + 8 q2mm3 + 4 q2mm4 + 4 q2mm5 + 4 q2mm12 + 4 q2mm14 + 4 q3mm4 + 4 q3mm5 + 4 q3mm8 + 4 q3mm10 + 4 q4mm8 + 4 q4mm12 + 4 r1mm2 + 4 r1mm4 + 2 r1mm9 + 2 r1mm10 + 2 r1mm15 + 4 r2mm4);$$

$$e41 = 1/630 (0.0 + 2 q1mm2 + 2 q1mm3 + 1 q1mm6 + 2 q1mm8 + 2 q1mm10 + 1 q1mm11 + 2 q1mm12 + 2 q1mm14 + 2 q1mm17 + 2 q1mm21 + 2 q1mm22 + 2 q1mm23 + 2 q1mm24 + 2 q1mm25 + 2 q1mm26 + 2 q1mm27 + 1 q2mm2 + 2 q2mm4 + 2 q2mm7 + 2 q2mm8 + 2 q2mm11 + 2 q2mm18 + 2 q2mm24 + 1 q3mm3 + 2 q3mm4 + 2 q3mm6 + 2 q3mm7 + 2 q3mm12 + 2 q3mm17 + 2 q3mm19 + 2 q4mm8 + 2 q4mm10 + 2 q4mm12 + 2 q4mm14 + 2 q4mm22 + 2 q4mm26 + 2 q5mm8 + 2 q5mm10 + 2 q5mm12 + 2 q5mm14 + 2 q7mm8 + 2 q7mm10 + 2 q7mm12 + 2 q7mm14 + 2 q8mm10 + 1 q10mm10 + 2 q12mm14 + 1 q14mm14 + 2 r1mm6 + 2 r1mm7 + 2 r1mm12 + 2 r1mm14 + 2 r2mm2 + 2 r2mm3 + 2 r2mm10 + 2 r2mm12 + 2 r2mm13 + 2 r3mm4 + 2 r3mm9 + 2 r4mm4 + 2 r4mm7 + 2 r4mm13 + 2 r4mm15);$$

$$e42 = 1/630 (0.0 + 4 q1mm8 + 4 q1mm12 + 2 q1mm21 + 4 q1mm22 + 2 q1mm23 + 2 q1mm25 + 4 q1mm26 + 2 q1mm27 + 4 q2mm17 + 4 q3mm24 + 4 q4mm10 + 4 q4mm14 + 4 q4mm22 + 4 q4mm26 + 4 q5mm22 + 4 q5mm26 + 4 q6mm8 + 2 q6mm18 + 4 q7mm17 + 4 q7mm24 +$$

4 q10mm17 + 4 q11mm12 + 2 q11mm19 + 4 q14mm24 + 2 r1mm11 + 4 r2mm6 + 4 r2mm11 + 2 r3mm3 + 4 r3mm7 + 4 r3mm12 + 4 r4mm11 + 4 r4mm14 + 2 r6mm10 + 4 r7mm12 + 4 r7mm13 + 2 r9mm11 + 4 r12mm13 + 2 r13mm13 + 2 r14mm15);

e43 = 1/630 (0.0+ 2 q1mm1 + 8 q1mm15 + 16 q2mm3 + 15 q2mm13 + 15 q3mm9 + 16 q4mm16 + 15 q4mm20 + 4 q5mm5 + 8 q5mm20 + 16 r1mm8 + 8 r5mm8);

e44 = 1/630 (0.0+ 2 q1mm4 + 4 q1mm15 + 4 q2mm12 + 4 q2mm13 + 4 q3mm8 + 4 q3mm9 + 2 q4mm6 + 2 q4mm11 + 2 q4mm15 + 8 q4mm16 + 8 q4mm20 + 2 q5mm15 + 4 q5mm16 + 1 q5mm28 + 2 q5mm30 + 2 q5mm31 + 1 q5mm32 + 4 q6mm13 + 4 q7mm9 + 4 q7mm13 + 2 q8mm8 + 4 q9mm10 + 4 q9mm11 + 4 q9mm14 + 4 q9mm20 + 4 q10mm13 + 2 q12mm12 + 4 q13mm14 + 4 q13mm20 + 4 q15mm16 + 4 q16mm16 + 4 r1mm8 + 4 r2mm3 + 4 r2mm8 + 4 r3mm4 + 4 r3mm8 + 4 r4mm8 + 2 r5mm16 + 2 r5mm17 + 2 r5mm18 + 2 r5mm19 + 2 r5mm20 + 2 r5mm22);

e45 = 1/630 (0.0+ 2 q1mm8 + 4 q1mm9 + 2 q1mm12 + 4 q1mm13 + 8 q2mm4 + 4 q2mm13 + 4 q2mm15 + 4 q2mm16 + 8 q3mm4 + 4 q3mm9 + 4 q3mm15 + 4 q3mm16 + 8 q4mm9 + 8 q4mm13 + 4 q5mm8 + 4 q5mm9 + 4 q5mm12 + 4 q5mm13 + 2 q7mm15 + 4 q7mm16 + 4 q9mm13 + 4 q9mm16 + 2 q10mm15 + 4 q10mm16 + 4 q13mm16 + 2 q14mm15 + 4 q14mm16 + 2 r1mm1 + 2 r1mm2 + 2 r1mm4 + 2 r1mm5 + 4 r2mm5 + 4 r4mm5 + 2 r5mm5);

e46 = 1/630 (0.0+ 3 q1mm17 + 1 q1mm22 + 2 q1mm23 + 3 q1mm24 + 1 q1mm26 + 2 q1mm27 + 2 q2mm4 + 4 q2mm8 + 2 q2mm18 + 2 q2mm21 + 2 q2mm22 + 2 q2mm23 + 2 q2mm24 + 2 q3mm4 + 4 q3mm12 + 2 q3mm17 + 2 q3mm19 + 2 q3mm25 + 2 q3mm26 + 2 q3mm27 + 2 q4mm6 + 2 q4mm11 + 2 q4mm17 + 2 q4mm21 + 2 q4mm24 + 2 q4mm25 + 2 q5mm8 + 2 q5mm12 + 2 q5mm17 + 2 q5mm24 + 1 q6mm24 + 2 q7mm18 + 2 q7mm19 + 2 q7mm22 + 2 q7mm26 + 2 q8mm10 + 2 q9mm10 + 2 q9mm22 + 2 q10mm16 + 2 q10mm22 + 1 q11mm17 + 2 q12mm14 + 2 q13mm14 + 2 q13mm26 + 2 q14mm16 + 2 q14mm26 + 1 q15mm22 + 1 q15mm26 + 2 r1mm3 + 1 r1mm6 + 1 r1mm7 + 1 r1mm12 + 2 r1mm13 + 1 r1mm14 + 4 r2mm7 + 2 r2mm10 + 2 r2mm13 + 2 r3mm6 + 2 r3mm14 + 4 r4mm12 + 2 r4mm13 + 2 r4mm15 + 2 r5mm13 + 1 r6mm12 + 1 r6mm13 + 2 r7mm9 + 2 r7mm14 + 2 r9mm12 + 2 r12mm15 + 1 r13mm14);

e47 = 1/630 (0.0+ 2 q1mm2 + 2 q1mm3 + 2 q1mm7 + 2 q1mm9 + 2 q1mm13 + 2 q1mm18 + 2 q1mm19 + 2 q1mm21 + 2 q1mm25 + 5 q2mm3 + 2 q2mm5 + 2 q2mm12 + 2 q2mm13 + 2 q2mm14 + 2 q2mm15 + 2 q2mm16 + 2 q2mm25 + 2 q2mm27 + 2 q3mm5 + 2 q3mm8 + 2 q3mm9 + 2 q3mm10 + 2 q3mm15 + 2 q3mm16 + 2 q3mm21 + 2 q3mm23 + 4 q4mm7 + 2 q4mm8 + 2 q4mm9 + 2 q4mm10 + 2 q4mm12 + 2 q4mm13 + 2 q4mm14 + 2 q4mm18 + 2 q4mm19 + 2 q4mm23 + 2 q4mm27 + 4 q5mm7 + 2 q5mm10 + 2 q5mm14 + 2 q5mm13 + 2 q8mm14 + 2 q8mm16 + 2 q9mm12 + 2 q10mm12 + 2 q10mm14 + 2 q12mm16 + 2 r1mm2 + 2 r1mm4 + 4 r1mm9 + 2 r1mm10 + 2 r1mm15 + 4 r2mm4 + 2 r2mm5 + 2 r2mm15 + 2 r4mm5 + 2 r4mm10);

e48 = 1/630 (0.0+ 1 q1mm8 + 1 q1mm9 + 1 q1mm12 + 1 q1mm13 + 1 q1mm21 + 1 q1mm25 + 1 q2mm4 + 1 q2mm8 + 1 q2mm9 + 1 q2mm12 + 1 q2mm15 + 2 q2mm16 + 1 q2mm18 + 1 q2mm19 + 1 q2mm25 + 1 q2mm26 + 1 q2mm27 + 1 q3mm4 + 1 q3mm8 + 1 q3mm12 + 1 q3mm13 + 1 q3mm15 + 2 q3mm16 + 1 q3mm18 + 1 q3mm19 + 1 q3mm21 + 1 q3mm22 + 1 q3mm23 + 1 q4mm6 + 2 q4mm7 + 1 q4mm8 + 1 q4mm9 + 2 q4mm10 + 1 q4mm11 + 1 q4mm12 + 1 q4mm13 + 2 q4mm14 + 1 q4mm17 + 1 q4mm18 + 1 q4mm19 + 1 q4mm21 + 2 q4mm23 + 1 q4mm24 + 1 q4mm25 + 2 q4mm27 + 1 q5mm8 + 1 q5mm9 + 1 q5mm12 + 1 q5mm13 + 1 q5mm17 + 1 q5mm18 + 1 q5mm19 + 1 q5mm22 + 1 q5mm23 + 1 q5mm24 + 1 q5mm26 + 1 q5mm27 + 1 q6mm12 + 1 q6mm13 + 1 q7mm8 + 1 q7mm9 + 1 q7mm12 + 1 q7mm13 + 1 q7mm21 + 1 q7mm23 + 1 q7mm25 + 1 q7mm27 + 1 q8mm9 + 1 q8mm10 + 1 q8mm11 + 1 q8mm15 + 1 q9mm11 + 1 q9mm14 + 1 q9mm16 + 1 q9mm18 + 1 q9mm24 + 1 q9mm26 + 1 q10mm13 + 1 q10mm15 + 1 q10mm16 + 1 q10mm19 + 1 q10mm21 + 1 q10mm23 + 1 q10mm24 + 1 q12mm13 + 1 q12mm14 + 1 q12mm15 + 1 q13mm16 + 1 q13mm17 + 1 q13mm19 + 1 q13mm22 + 1 q14mm15 + 1 q14mm16 + 1 q14mm17 + 1 q14mm18 + 1 q14mm25 + 1 q14mm27 + 1 q16mm21 + 1 q16mm22 + 1 q16mm25 + 1 q16mm26 + 1 r1mm2 + 1 r1mm4 + 1 r1mm7 + 1 r1mm10 + 1 r1mm12 + 1 r1mm15 + 1 r2mm2 + 2 r2mm3 + 1 r2mm9 + 1 r2mm10 + 1 r2mm12 + 1 r2mm13 + 1 r2mm14 + 2 r3mm4 + 2 r3mm9 + 1 r4mm4 + 1 r4mm6 + 1 r4mm7 + 1 r4mm9 + 1 r4mm13 + 1 r4mm15 + 1 r5mm6 + 1 r5mm7 + 1 r5mm10 + 1 r5mm12 + 1 r5mm14 + 1 r5mm15 + 1 r7mm15 + 1 r9mm10 + 1 r9mm15 + 1 r10mm12 + 1 r10mm13 + 1 r13mm15);

e49 = 1/630 (0.0+ 1 q1mm2 + 1 q1mm3 + 2 q1mm21 + 2 q1mm25 + 4 q2mm3 + 2 q2mm7 + 3 q2mm11 + 2 q2mm13 + 2 q2mm14 + 1 q2mm15 + 2 q2mm18 + 2 q2mm24 + 2 q2mm25 + 2 q2mm27 + 1 q2mm32 + 3 q3mm6 + 2 q3mm7 + 2 q3mm9 + 2 q3mm10 + 1 q3mm15 + 2 q3mm17 + 2 q3mm19 + 2 q3mm21 + 2 q3mm23 + 1 q3mm28 + 4 q4mm20 + 2 q4mm22 + 2 q4mm23 + 2 q4mm26 + 2 q4mm27 + 2 q4mm30 + 2 q4mm31 + 2 q5mm10 + 2 q5mm14 + 4 q5mm20 + 2 q7mm10 + 2 q7mm14 + 2 q8mm13 + 2 q8mm16 + 2 q8mm18 + 2 q8mm20 + 2 q8mm22 + 2 q9mm12 + 2 q10mm20 + 2 q12mm16 + 2 q12mm19 + 2 q12mm20 + 2 q14mm20 + 1 r1mm6 + 2 r1mm8 + 1 r1mm14 + 2 r1mm16 + 2 r1mm17 + 2 r1mm19 + 2 r1mm20 + 2 r2mm8 + 2 r2mm10 + 2 r2mm12 + 2 r2mm22 + 2 r3mm8 + 4 r3mm9 + 2 r4mm7 + 2 r4mm8 + 2 r4mm15 + 2 r4mm18 + 2 r5mm8 + 2 r8mm10 + 2 r8mm13 + 2 r8mm15);

e50 = 1/630 (0.0+ 1 q1mm8 + 1 q1mm12 + 2 q1mm21 + 2 q1mm25 + 2 q2mm12 + 2 q2mm25 + 2 q2mm26 + 2 q2mm27 + 2 q3mm8 + 2 q3mm21 + 2 q3mm22 + 2 q3mm23 + 4 q4mm20 + 2 q4mm22 + 2 q4mm23 + 2 q4mm26 + 2 q4mm27 + 2 q4mm30 + 2 q4mm31 + 2 q5mm22 + 2 q5mm26 + 2 q5mm30 + 2 q5mm31 + 1 q6mm8 + 2 q6mm18 + 2 q7mm17 + 2 q7mm24 + 1 q7mm28 + 1 q7mm32 + 2 q8mm17 + 2 q9mm14 + 2 q9mm26 + 2 q10mm13 + 1 q10mm15 + 2 q10mm17 + 1 q10mm28 + 1 q11mm12 + 2 q11mm19 + 2 q12mm24 + 2 q13mm22 + 1 q14mm15 + 2 q14mm24 + 1 q14mm32 + 2 q16mm22 + 2 q16mm26 + 2 q17mm18 + 2 q17mm20 + 2 q19mm24 + 2 q20mm24 + 2 r1mm8 + 2 r1mm20 + 1 r2mm6 + 3 r2mm11 + 2 r2mm16 + 2 r3mm7 + 2 r3mm12 + 2 r3mm17 + 3 r4mm11 + 1 r4mm14 + 2 r4mm19 + 2 r5mm20 + 2 r6mm8 + 2 r6mm10 + 4 r7mm12 + 2 r7mm13 + 2 r7mm22 + 2 r8mm11 + 2 r8mm14 + 4 r9mm11 + 2 r10mm16 + 2 r12mm13 + 2 r12mm18 + 2 r13mm17 + 2 r13mm18 + 2 r13mm22 + 2 r14mm15 + 2 r15mm19);

e51 = 1/630 (0.0+ 5 q1mm15 + 12 q6mm13 + 12 q9mm11 + 24 q9mm20 + 24 q13mm20 + 12 q15mm16 + 11 q16mm16 + 24 r3mm8 + 5 r5mm23 + 12 r5mm24 + 11 r8mm8);

e52 = 1/630 (0.0+ 1 q1mm9 + 1 q1mm13 + 2 q1mm21 + 2 q1mm25 + 2 q2mm18 + 2 q2mm19 + 2 q3mm18 + 2 q3mm19 + 2 q6mm12 + 1 q6mm13 + 1 q6mm15 + 1 q6mm16 + 2 q6mm25 + 2 q6mm27 + 2 q7mm9 + 2 q7mm13 + 2 q7mm23 + 2 q7mm27 + 2 q8mm11 + 2 q8mm16 + 2 q8mm20 + 2 q8mm21 + 2 q9mm10 + 1 q9mm11 + 1 q9mm15 + 2 q9mm19 + 2 q9mm20 + 2 q9mm23 + 2 q9mm31 + 1 q9mm32 + 2 q10mm16 + 2 q10mm24 + 2 q10mm31 + 1 q11mm15 + 2 q11mm16 + 2 q11mm21 + 2 q11mm23 + 2 q12mm16 + 2 q12mm20 + 2 q12mm25 + 2 q13mm14 + 1 q13mm15 + 2 q13mm18 + 2 q13mm20 + 2 q13mm27 + 1 q13mm28 + 2 q13mm30 + 2 q14mm16 + 2 q14mm17 + 2 q14mm30 + 2 q15mm23 + 2 q15mm27 + 2 q16mm22 + 2 q16mm26 + 1 q16mm28 + 1 q16mm32 + 2 q18mm21 + 2 q19mm25 + 2 r1mm17 + 2 r2mm3 + 2 r2mm10 + 2 r2mm19 + 2 r2mm22 + 2 r3mm4 + 4 r3mm8 + 4 r3mm9 + 2 r3mm10 + 2 r3mm15 + 2 r3mm20 + 2 r4mm15 + 2 r4mm16 + 2 r4mm18 + 1 r5mm6 + 1 r5mm14 + 2 r5mm17 + 2 r5mm18 + 2 r5mm22 + 2 r7mm15 + 4 r8mm9 + 2 r10mm12 + 2 r10mm19 + 2 r13mm20 + 2 r15mm16);

e53 = 1/630 (0.0+ 3 q1mm21 + 3 q1mm25 + 3 q6mm18 + 3 q6mm25 + 3 q6mm27 + 6 q9mm20 + 6 q9mm31 + 3 q11mm19 + 3 q11mm21 +

3 q11mm23 + 6 q13mm20 + 6 q13mm30 + 3 q15mm23 + 3 q15mm27 + 6 q16mm22 + 6 q16mm26 + 6 q17mm20 + 3 q18mm28 + 3 q19mm32 + 6 q20mm24 + 5 r3mm8 + 5 r3mm17 + 5 r3mm20 + 5 r5mm24 + 3 r6mm10 + 6 r7mm22 + 5 r8mm11 + 6 r8mm16 + 6 r8mm19 + 5 r9mm11 + 3 r10mm23 + 6 r12mm18 + 5 r13mm24 + 3 r14mm15 + 3 r15mm23);

e54 = 1/630 (0.0+ 2 q1mm9 + 2 q1mm13 + 3 q1mm18 + 3 q1mm19 + 1 q1mm21 + 1 q1mm25 + 2 q2mm9 + 2 q2mm12 + 2 q2mm13 + 2 q2mm16 + 2 q2mm21 + 2 q2mm27 + 2 q3mm8 + 2 q3mm9 + 2 q3mm13 + 2 q3mm16 + 2 q3mm23 + 2 q3mm25 + 2 q6mm15 + 2 q6mm16 + 1 q6mm25 + 1 q6mm27 + 2 q7mm9 + 2 q7mm13 + 4 q7mm16 + 2 q8mm9 + 2 q8mm14 + 2 q8mm16 + 2 q9mm9 + 2 q9mm10 + 2 q9mm15 + 2 q9mm19 + 2 q9mm23 + 2 q9mm25 + 2 q10mm12 + 2 q10mm16 + 2 q10mm26 + 2 q11mm15 + 2 q11mm16 + 1 q11mm21 + 1 q11mm23 + 2 q12mm13 + 2 q12mm16 + 2 q13mm13 + 2 q13mm14 + 2 q13mm15 + 2 q13mm18 + 2 q13mm21 + 2 q13mm27 + 2 q14mm16 + 2 q14mm22 + 1 q15mm18 + 1 q15mm19 + 1 q15mm23 + 1 q15mm27 + 2 q16mm17 + 2 q16mm24 + 2 r1mm3 + 2 r1mm9 + 1 r1mm10 + 1 r1mm15 + 2 r2mm3 + 2 r2mm5 + 2 r2mm9 + 2 r2mm15 + 2 r3mm4 + 2 r3mm5 + 2 r3mm10 + 2 r3mm15 + 2 r4mm5 + 2 r4mm9 + 2 r4mm10 + 2 r5mm7 + 2 r5mm12 + 2 r5mm13 + 1 r6mm15 + 1 r10mm14);

e55 = 1/630 (0.0+ 2 q1mm6 + 2 q1mm11 + 2 q1mm17 + 2 q1mm20 + 2 q1mm24 + 2 q1mm28 + 2 q1mm30 + 2 q1mm31 + 2 q1mm32 + 2 q2mm3 + 2 q2mm10 + 2 q2mm11 + 2 q2mm12 + 2 q2mm20 + 2 q2mm22 + 2 q2mm24 + 2 q2mm30 + 2 q3mm6 + 2 q3mm8 + 2 q3mm14 + 2 q3mm17 + 2 q3mm20 + 2 q3mm26 + 2 q3mm31 + 2 q6mm11 + 2 q7mm7 + 2 q7mm8 + 2 q7mm10 + 2 q7mm12 + 2 q7mm14 + 2 q7mm20 + 2 q8mm8 + 2 q8mm10 + 2 q8mm14 + 2 q8mm22 + 2 q10mm10 + 2 q10mm12 + 2 q10mm14 + 2 q12mm12 + 2 q12mm14 + 2 q12mm26 + 2 q14mm14 + 2 r1mm16 + 2 r1mm18 + 2 r1mm19 + 2 r1mm22 + 2 r2mm3 + 2 r2mm7 + 2 r2mm8 + 2 r2mm13 + 2 r2mm17 + 2 r2mm18 + 2 r2mm20 + 2 r3mm4 + 2 r3mm6 + 2 r3mm14 + 2 r3mm16 + 2 r3mm19 + 2 r4mm8 + 2 r4mm12 + 2 r4mm13 + 2 r4mm17 + 2 r4mm20 + 2 r4mm22 + 2 r7mm8 + 2 r8mm12);

e56 = 1/630 (0.0+ 2 q1mm17 + 2 q1mm24 + 1 q1mm28 + 3 q1mm30 + 3 q1mm31 + 1 q1mm32 + 2 q2mm12 + 2 q2mm17 + 2 q2mm22 + 2 q2mm24 + 2 q2mm28 + 2 q2mm30 + 2 q3mm8 + 2 q3mm17 + 2 q3mm24 + 2 q3mm26 + 2 q3mm31 + 2 q3mm32 + 2 q6mm22 + 1 q6mm30 + 2 q7mm17 + 2 q7mm22 + 2 q7mm24 + 2 q7mm26 + 2 q8mm14 + 2 q8mm17 + 2 q8mm22 + 2 q10mm12 + 2 q10mm17 + 2 q10mm22 + 2 q10mm26 + 2 q11mm26 + 1 q11mm31 + 2 q12mm24 + 2 q12mm26 + 2 q14mm22 + 2 q14mm24 + 2 q14mm26 + 2 q17mm22 + 2 q24mm26 + 2 r1mm11 + 1 r1mm16 + 1 r1mm18 + 1 r1mm19 + 1 r1mm22 + 2 r2mm11 + 2 r2mm13 + 2 r2mm16 + 2 r2mm20 + 2 r3mm13 + 2 r3mm18 + 2 r3mm22 + 2 r4mm11 + 2 r4mm13 + 2 r4mm19 + 2 r4mm20 + 2 r6mm7 + 2 r6mm11 + 1 r6mm17 + 1 r6mm18 + 1 r6mm20 + 2 r7mm7 + 2 r7mm13 + 2 r7mm16 + 2 r7mm17 + 2 r7mm18 + 2 r8mm14 + 1 r1mm16 + 1 r1mm19 + 2 r12mm12 + 2 r12mm13 + 2 r12mm14 + 2 r12mm17 + 2 r12mm19 + 2 r12mm22 + 2 r13mm13 + 1 r14mm17 + 1 r14mm20 + 1 r14mm22);

e57 = 1/630 (0.0+ 1 q1mm17 + 1 q1mm24 + 2 q1mm28 + 2 q1mm32 + 2 q2mm30 + 2 q2mm31 + 2 q3mm30 + 2 q3mm31 + 2 q6mm12 + 1 q6mm22 + 1 q6mm24 + 2 q6mm30 + 2 q7mm17 + 2 q7mm24 + 2 q7mm30 + 2 q7mm31 + 2 q8mm11 + 2 q8mm20 + 2 q8mm22 + 2 q8mm28 + 2 q10mm17 + 2 q10mm22 + 2 q10mm24 + 2 q10mm31 + 1 q11mm17 + 1 q11mm26 + 2 q11mm31 + 2 q12mm20 + 2 q12mm26 + 2 q12mm32 + 2 q14mm17 + 2 q14mm24 + 2 q14mm26 + 2 q14mm30 + 2 q17mm20 + 2 q20mm24 + 2 q22mm22 + 1 q22mm28 + 2 q26mm26 + 1 q26mm32 + 2 r1mm24 + 2 r2mm11 + 2 r2mm18 + 2 r2mm23 + 2 r2mm24 + 2 r3mm8 + 2 r3mm16 + 2 r3mm18 + 2 r3mm19 + 2 r3mm22 + 2 r3mm24 + 2 r4mm11 + 2 r4mm22 + 2 r4mm23 + 2 r4mm24 + 1 r6mm7 + 1 r6mm11 + 1 r6mm13 + 2 r6mm18 + 2 r7mm17 + 2 r7mm18 + 2 r7mm20 + 1 r7mm23 + 2 r7mm24 + 2 r8mm11 + 2 r8mm16 + 2 r8mm19 + 1 r1mm14 + 2 r11mm16 + 2 r11mm19 + 1 r12mm14 + 2 r12mm17 + 2 r12mm20 + 2 r12mm22 + 1 r12mm23 + 2 r12mm24 + 1 r13mm14 + 2 r13mm17 + 2 r13mm18 + 2 r13mm20 + 2 r13mm22 + 2 r14mm22 + 2 r16mm17 + 2 r16mm20 + 2 r17mm19 + 2 r19mm20);

e58 = 1/630 (0.0+ 3 q1mm28 + 3 q1mm32 + 5 q6mm30 + 5 q11mm31 + 11 q17mm20 + 11 q20mm24 + 6 q22mm22 + 6 q26mm26 + 12 r3mm24 + 6 r6mm18 + 11 r7mm24 + 12 r8mm11 + 11 r8mm23 + 5 r11mm16 + 5 r11mm19 + 11 r12mm24 + 12 r13mm24 + 6 r14mm22 + 6 r17mm23 + 5 r20mm23);

e59 = 1/630 (0.0+ 4 q2mm13 + 4 q2mm18 + 4 q2mm21 + 4 q2mm24 + 4 q2mm27 + 4 q3mm9 + 4 q3mm17 + 4 q3mm19 + 4 q3mm23 + 4 q3mm25 + 2 q6mm15 + 1 q6mm32 + 4 q9mm22 + 4 q9mm23 + 4 q9mm25 + 2 q11mm15 + 1 q11mm28 + 4 q13mm21 + 4 q13mm26 + 4 q13mm27 + 2 q15mm30 + 2 q15mm31 + 4 q16mm17 + 8 q16mm20 + 4 q16mm24 + 2 q18mm18 + 2 q19mm19 + 4 q20mm22 + 4 q20mm26 + 2 r1mm3 + 2 r1mm16 + 2 r1mm17 + 2 r1mm19 + 2 r1mm20 + 2 r3mm6 + 2 r3mm14 + 4 r5mm8 + 4 r5mm13 + 2 r6mm22 + 4 r7mm9 + 4 r7mm10 + 4 r8mm10 + 4 r8mm13 + 4 r8mm15 + 4 r9mm12 + 4 r12mm15 + 2 r14mm18);

e60 = 1/630 (0.0+ 15 q2mm13 + 8 q2mm15 + 15 q3mm9 + 8 q3mm15 + 16 q9mm13 + 16 q9mm16 + 16 q13mm16 + 8 q15mm20 + 15 q16mm20 + 2 r1mm1 + 8 r1mm8 + 4 r5mm5 + 16 r5mm8);

e61 = 1/630 (0.0+ 2 q2mm13 + 2 q2mm15 + 4 q2mm25 + 2 q3mm9 + 2 q3mm15 + 4 q3mm21 + 1 q6mm12 + 2 q6mm13 + 2 q7mm9 + 2 q7mm13 + 2 q7mm15 + 1 q8mm11 + 2 q8mm13 + 1 q8mm15 + 2 q8mm18 + 2 q9mm11 + 2 q9mm12 + 4 q9mm13 + 2 q9mm14 + 4 q9mm16 + 2 q9mm18 + 2 q9mm24 + 2 q9mm26 + 4 q9mm27 + 2 q10mm13 + 1 q10mm15 + 2 q10mm19 + 1 q10mm32 + 1 q12mm15 + 2 q12mm19 + 4 q13mm16 + 2 q13mm17 + 2 q13mm19 + 2 q13mm22 + 4 q13mm23 + 1 q14mm15 + 2 q14mm18 + 1 q14mm28 + 4 q15mm20 + 2 q16mm18 + 2 q16mm19 + 4 q16mm20 + 2 q16mm21 + 2 q16mm23 + 2 q16mm25 + 2 q16mm27 + 2 q16mm30 + 2 q16mm31 + 2 q20mm21 + 2 q20mm23 + 2 q20mm25 + 2 q20mm27 + 1 r1mm2 + 1 r1mm4 + 2 r1mm8 + 2 r2mm8 + 2 r2mm12 + 1 r2mm14 + 2 r3mm8 + 4 r3mm9 + 1 r4mm6 + 2 r4mm7 + 2 r4mm8 + 2 r5mm8 + 2 r5mm10 + 2 r5mm15 + 2 r5mm16 + 2 r5mm17 + 2 r5mm19 + 2 r5mm20 + 2 r8mm10 + 2 r8mm13 + 2 r8mm15 + 2 r9mm10 + 2 r9mm15 + 2 r10mm22 + 2 r15mm18);

e62 = 1/630 (0.0+ 8 q2mm25 + 8 q3mm21 + 8 q9mm26 + 8 q9mm27 + 8 q13mm22 + 8 q13mm23 + 8 q15mm20 + 8 q16mm30 + 8 q16mm31 + 8 q17mm18 + 8 q19mm24 + 4 q20mm28 + 4 q20mm32 + 4 r1mm8 + 8 r5mm20 + 4 r6mm8 + 8 r7mm12 + 4 r8mm11 + 4 r8mm14 + 8 r9mm11 + 8 r10mm16 + 8 r13mm17 + 8 r15mm19 + 8 r18mm22);

e63 = 1/630 (0.0+ 2 q1mm17 + 2 q1mm24 + 1 q1mm28 + 1 q1mm32 + 2 q2mm22 + 2 q2mm23 + 2 q2mm30 + 2 q2mm31 + 2 q3mm26 + 2 q3mm27 + 2 q3mm30 + 2 q3mm31 + 2 q4mm6 + 2 q4mm11 + 2 q4mm28 + 2 q4mm32 + 2 q5mm17 + 2 q5mm24 + 2 q6mm12 + 2 q6mm24 + 2 q7mm8 + 2 q7mm12 + 2 q7mm18 + 2 q7mm19 + 2 q7mm30 + 2 q7mm31 + 1 q8mm8 + 4 q8mm10 + 2 q8mm11 + 2 q8mm21 + 2 q10mm19 + 2 q10mm22 + 2 q10mm23 + 2 q10mm24 + 2 q11mm17 + 1 q12mm12 + 4 q12mm14 + 2 q12mm25 + 2 q14mm17 + 2 q14mm18 + 2 q14mm26 + 2 q14mm27 + 2 q21mm22 + 1 q22mm22 + 2 q25mm26 + 1 q26mm26 + 2 r1mm18 + 2 r1mm22 + 2 r2mm3 + 2 r2mm7 + 2 r2mm17 + 4 r2mm18 + 2 r2mm19 + 2 r3mm4 + 2 r3mm6 + 2 r3mm14 + 2 r3mm16 + 2 r3mm19 + 2 r4mm12 + 2 r4mm16 + 2 r4mm17 + 4 r4mm22 + 4 r6mm13 + 2 r7mm10 + 2 r7mm19 + 2 r7mm20 + 2 r9mm17 + 2 r9mm20 + 2 r10mm13 + 2 r12mm15 + 2 r12mm16 + 2 r12mm20 + 2 r13mm14 + 2 r13mm15);

e64 = 1/630 (0.0+ 12 q2mm3 + 12 q2mm25 + 12 q3mm21 + 12 q4mm18 + 12 q4mm19 + 12 q5mm7 + 12 q8mm27 + 12 q10mm14 + 12

q12mm23 + 12 r1mm9 + 12 r2mm15 + 12 r4mm10);

e65 = 1/630 (0.0 + 2 q2mm12 + 2 q2mm13 + 4 q2mm25 + 2 q3mm8 + 2 q3mm9 + 4 q3mm21 + 4 q4mm18 + 4 q4mm19 + 2 q5mm18 + 2 q5mm19 + 1 q6mm15 + 2 q6mm25 + 4 q7mm16 + 2 q7mm21 + 2 q7mm25 + 2 q8mm9 + 4 q8mm27 + 1 q9mm9 + 2 q9mm18 + 2 q9mm19 + 2 q9mm25 + 2 q10mm21 + 2 q10mm25 + 2 q10mm26 + 2 q10mm27 + 1 q11mm15 + 2 q11mm21 + 2 q12mm13 + 4 q12mm23 + 1 q13mm13 + 2 q13mm18 + 2 q13mm19 + 2 q13mm21 + 2 q14mm21 + 2 q14mm22 + 2 q14mm23 + 2 q14mm25 + 2 q15mm18 + 2 q15mm19 + 2 q16mm17 + 2 q16mm18 + 2 q16mm19 + 2 q16mm24 + 2 q17mm27 + 2 q18mm27 + 2 q19mm23 + 2 q21mm27 + 2 q22mm27 + 1 q23mm23 + 2 q23mm24 + 2 q23mm25 + 2 q23mm26 + 1 q27mm27 + 4 r1mm9 + 2 r2mm3 + 2 r2mm9 + 4 r2mm15 + 2 r3mm4 + 2 r3mm9 + 2 r4mm9 + 4 r4mm10 + 2 r5mm7 + 2 r5mm9 + 2 r5mm12 + 2 r6mm15 + 2 r7mm15 + 2 r9mm10 + 2 r9mm15 + 2 r10mm12 + 2 r10mm13 + 2 r10mm14 + 4 r10mm15 + 2 r13mm15);

e66 = 1/630 (0.0 + 4 q2mm3 + 4 q2mm11 + 4 q2mm25 + 4 q2mm32 + 4 q3mm6 + 4 q3mm21 + 4 q3mm28 + 4 q4mm30 + 4 q4mm31 + 4 q5mm20 + 2 q6mm11 + 2 q7mm7 + 4 q7mm10 + 4 q7mm14 + 4 q8mm18 + 4 q8mm27 + 4 q8mm30 + 4 q10mm14 + 4 q10mm20 + 4 q12mm19 + 4 q12mm23 + 4 q12mm31 + 4 q14mm20 + 2 q20mm20 + 4 r1mm16 + 4 r1mm19 + 4 r2mm17 + 4 r2mm20 + 4 r2mm22 + 4 r3mm9 + 4 r3mm16 + 4 r3mm19 + 4 r4mm17 + 4 r4mm18 + 4 r4mm20 + 4 r8mm10 + 4 r8mm15 + 4 r8mm18 + 4 r8mm22);

e67 = 1/630 (0.0 + 2 q2mm12 + 2 q2mm24 + 2 q2mm25 + 2 q2mm28 + 2 q3mm8 + 2 q3mm17 + 2 q3mm21 + 2 q3mm28 + 4 q4mm30 + 4 q4mm31 + 2 q5mm30 + 2 q5mm31 + 2 q7mm22 + 2 q7mm26 + 2 q7mm28 + 2 q7mm32 + 2 q8mm17 + 2 q8mm27 + 2 q8mm30 + 2 q10mm25 + 2 q10mm26 + 2 q10mm28 + 2 q10mm30 + 2 q12mm23 + 2 q12mm24 + 2 q12mm31 + 2 q14mm21 + 2 q14mm22 + 2 q14mm31 + 2 q14mm32 + 2 q17mm18 + 2 q17mm22 + 2 q17mm30 + 2 q18mm22 + 2 q19mm24 + 2 q19mm26 + 2 q22mm27 + 2 q23mm26 + 2 q24mm26 + 2 q24mm31 + 2 r1mm16 + 2 r1mm19 + 2 r2mm11 + 2 r2mm16 + 2 r2mm20 + 2 r2mm22 + 2 r4mm11 + 2 r4mm18 + 2 r4mm19 + 2 r4mm20 + 1 r6mm11 + 2 r6mm17 + 2 r6mm20 + 1 r7mm7 + 2 r7mm13 + 2 r7mm16 + 2 r7mm17 + 2 r7mm22 + 2 r9mm11 + 2 r9mm13 + 2 r10mm16 + 2 r10mm20 + 1 r11mm14 + 2 r11mm16 + 2 r11mm19 + 1 r12mm12 + 2 r12mm13 + 2 r12mm17 + 2 r12mm18 + 2 r12mm19 + 2 r13mm18 + 2 r13mm22 + 2 r14mm17 + 2 r14mm20 + 2 r15mm19 + 2 r15mm20 + 2 r16mm18 + 2 r17mm18 + 2 r17mm22 + 1 r18mm18 + 2 r19mm22 + 1 r22mm22);

e68 = 1/630 (0.0 + 4 q2mm25 + 4 q3mm21 + 5 q6mm25 + 4 q9mm18 + 4 q9mm19 + 5 q11mm21 + 4 q13mm18 + 4 q13mm19 + 2 q15mm18 + 2 q15mm19 + 4 q16mm18 + 4 q16mm19 + 4 q17mm27 + 4 q18mm21 + 4 q18mm27 + 4 q19mm23 + 4 q19mm25 + 4 q20mm21 + 4 q20mm25 + 4 q21mm27 + 4 q22mm27 + 4 q23mm24 + 4 q23mm25 + 4 q23mm26 + 4 q23mm31 + 2 q23mm32 + 2 q27mm28 + 4 q27mm30 + 2 r1mm9 + 4 r3mm8 + 8 r3mm9 + 4 r5mm17 + 2 r6mm15 + 4 r7mm15 + 4 r8mm9 + 4 r9mm10 + 4 r9mm15 + 4 r10mm12 + 2 r10mm14 + 4 r10mm15 + 4 r10mm19 + 4 r10mm22 + 4 r13mm20 + 4 r15mm16 + 4 r15mm18);

e69 = 1/630 (0.0 + 2 q2mm25 + 2 q2mm32 + 2 q3mm21 + 2 q3mm28 + 2 q6mm25 + 1 q6mm32 + 2 q9mm30 + 2 q9mm31 + 2 q11mm21 + 1 q11mm28 + 2 q13mm30 + 2 q13mm31 + 1 q15mm30 + 1 q15mm31 + 2 q16mm30 + 2 q16mm31 + 2 q17mm18 + 2 q17mm27 + 2 q17mm30 + 2 q18mm22 + 2 q18mm28 + 2 q18mm30 + 2 q19mm24 + 2 q19mm26 + 2 q19mm31 + 2 q19mm32 + 2 q20mm21 + 2 q20mm25 + 2 q20mm28 + 2 q20mm32 + 2 q21mm27 + 2 q22mm27 + 2 q22mm30 + 2 q23mm24 + 2 q23mm25 + 2 q23mm26 + 2 q23mm31 + 2 q24mm31 + 2 q26mm31 + 2 q27mm30 + 1 q28mm30 + 1 q31mm32 + 1 r1mm16 + 1 r1mm19 + 2 r3mm8 + 2 r3mm16 + 2 r3mm19 + 2 r5mm24 + 1 r6mm17 + 1 r6mm20 + 1 r6mm22 + 2 r7mm17 + 2 r7mm20 + 2 r7mm22 + 2 r8mm11 + 2 r8mm16 + 2 r8mm19 + 4 r9mm11 + 2 r9mm18 + 2 r9mm22 + 2 r10mm16 + 2 r10mm20 + 2 r10mm23 + 2 r10mm24 + 3 r11mm16 + 3 r11mm19 + 2 r12mm17 + 2 r12mm18 + 2 r12mm20 + 2 r13mm17 + 2 r13mm20 + 2 r13mm24 + 1 r14mm17 + 1 r14mm18 + 1 r14mm20 + 2 r15mm19 + 2 r15mm20 + 2 r15mm23 + 2 r15mm24 + 2 r16mm17 + 2 r16mm18 + 2 r16mm20 + 2 r17mm18 + 2 r17mm19 + 2 r17mm22 + 4 r18mm22 + 1 r18mm23 + 2 r18mm24 + 2 r19mm20 + 2 r19mm22 + 1 r22mm23 + 2 r22mm24);

e70 = 1/630 (0.0 + 1 q1mm6 + 1 q1mm11 + 2 q1mm28 + 2 q1mm32 + 4 q2mm11 + 4 q2mm20 + 4 q2mm24 + 4 q2mm30 + 4 q3mm6 + 4 q3mm17 + 4 q3mm20 + 4 q3mm31 + 4 q6mm11 + 8 q7mm20 + 4 q8mm20 + 4 q8mm22 + 2 q10mm10 + 4 q10mm20 + 4 q12mm20 + 4 q12mm26 + 2 q14mm14 + 4 q14mm20 + 4 r1mm23 + 4 r1mm24 + 4 r2mm18 + 4 r2mm24 + 2 r3mm6 + 4 r3mm8 + 2 r3mm14 + 4 r3mm16 + 4 r3mm19 + 2 r3mm23 + 4 r4mm22 + 4 r4mm24 + 4 r7mm8 + 4 r8mm8 + 4 r8mm12 + 4 r8mm13 + 4 r8mm17 + 4 r8mm20);

e71 = 1/630 (0.0 + 8 q2mm24 + 4 q2mm28 + 8 q2mm30 + 8 q3mm17 + 8 q3mm31 + 4 q3mm32 + 8 q17mm22 + 8 q20mm22 + 8 q20mm26 + 8 q24mm26 + 2 r1mm11 + 4 r1mm23 + 4 r1mm24 + 4 r6mm11 + 4 r6mm24 + 8 r7mm16 + 8 r7mm18 + 8 r8mm13 + 8 r8mm20 + 4 r11mm14 + 2 r11mm23 + 8 r12mm19 + 8 r12mm22 + 4 r13mm13 + 4 r14mm24 + 4 r17mm17);

e72 = 1/630 (0.0 + 4 q2mm24 + 4 q3mm17 + 2 q4mm6 + 2 q4mm11 + 2 q4mm28 + 2 q4mm32 + 2 q5mm28 + 2 q5mm32 + 4 q6mm24 + 4 q7mm30 + 4 q7mm31 + 4 q8mm20 + 4 q8mm22 + 4 q10mm23 + 4 q10mm30 + 4 q11mm17 + 4 q12mm20 + 4 q12mm26 + 4 q14mm27 + 4 q14mm31 + 4 q18mm20 + 4 q19mm20 + 4 q21mm22 + 4 q25mm26 + 2 r1mm23 + 4 r2mm18 + 4 r2mm24 + 4 r3mm6 + 4 r3mm14 + 4 r3mm23 + 4 r4mm22 + 4 r4mm24 + 4 r7mm8 + 4 r7mm19 + 2 r8mm8 + 4 r8mm12 + 4 r8mm17 + 4 r9mm24 + 4 r10mm18 + 4 r12mm16 + 4 r13mm16 + 4 r13mm19 + 4 r15mm22 + 4 r17mm20 + 2 r20mm20);

e73 = 1/630 (0.0 + 4 q2mm11 + 4 q2mm32 + 4 q3mm6 + 4 q3mm28 + 8 q6mm11 + 8 q7mm20 + 8 q8mm30 + 8 q10mm20 + 8 q12mm31 + 8 q14mm20 + 8 q20mm20 + 4 r1mm23 + 8 r2mm24 + 8 r3mm16 + 8 r3mm19 + 8 r3mm23 + 8 r4mm24 + 8 r8mm17 + 8 r8mm18 + 8 r8mm20 + 8 r8mm22 + 8 r8mm24);

e74 = 1/630 (0.0 + 2 q2mm24 + 2 q2mm32 + 2 q3mm17 + 2 q3mm28 + 1 q6mm12 + 2 q6mm24 + 1 q7mm28 + 2 q7mm30 + 2 q7mm31 + 1 q7mm32 + 1 q8mm11 + 1 q8mm28 + 4 q8mm30 + 1 q10mm28 + 2 q10mm30 + 2 q10mm31 + 1 q10mm32 + 2 q11mm17 + 4 q12mm31 + 1 q12mm32 + 1 q14mm28 + 2 q14mm30 + 2 q14mm31 + 1 q14mm32 + 2 q17mm22 + 2 q17mm30 + 2 q20mm22 + 2 q20mm26 + 2 q20mm30 + 2 q20mm31 + 2 q22mm28 + 2 q22mm30 + 2 q24mm26 + 2 q24mm31 + 2 q26mm31 + 2 q26mm32 + 2 r1mm23 + 1 r2mm11 + 1 r2mm23 + 4 r2mm24 + 2 r3mm16 + 2 r3mm19 + 2 r3mm23 + 1 r4mm11 + 1 r4mm23 + 4 r4mm24 + 2 r6mm11 + 2 r6mm24 + 2 r7mm16 + 2 r7mm18 + 2 r7mm23 + 2 r7mm24 + 2 r8mm18 + 2 r8mm20 + 2 r8mm22 + 2 r8mm24 + 2 r11mm14 + 2 r11mm16 + 2 r11mm19 + 2 r11mm23 + 2 r12mm19 + 2 r12mm22 + 2 r12mm23 + 2 r12mm24 + 2 r13mm16 + 2 r13mm18 + 2 r13mm19 + 2 r13mm22 + 2 r14mm24 + 2 r16mm17 + 2 r16mm18 + 2 r16mm20 + 2 r16mm24 + 2 r17mm17 + 2 r17mm18 + 2 r17mm19 + 2 r17mm20 + 2 r17mm22 + 2 r17mm24 + 2 r18mm18 + 2 r18mm20 + 2 r19mm20 + 2 r19mm22 + 2 r19mm24 + 2 r20mm20 + 2 r20mm22 + 2 r22mm22);

e75 = 1/630 (0.0 + 4 q2mm32 + 4 q3mm28 + 8 q17mm30 + 4 q20mm28 + 4 q20mm32 + 8 q22mm30 + 8 q24mm31 + 8 q26mm31 + 2 r1mm23 + 4 r6mm24 + 8 r7mm24 + 4 r8mm11 + 4 r8mm23 + 8 r11mm16 + 8 r11mm19 + 4 r11mm23 + 8 r12mm24 + 8 r13mm24 + 4 r14mm24 + 8 r16mm18 + 8 r17mm23 + 8 r17mm24 + 8 r18mm24 + 8 r19mm22 + 4 r20mm20 + 8 r20mm23 + 8 r22mm24 + 4 r23mm24);

e76 = 1/630 (0.0 + 2 q6mm25 + 4 q6mm32 + 2 q11mm21 + 4 q11mm28 + 2 q18mm28 + 4 q18mm30 + 4 q19mm31 + 2 q19mm32 + 4 q21mm30

+ 4 q23mm31 + 2 q23mm32 + 4 q25mm31 + 2 q27mm28 + 4 q27mm30 + 4 q28mm30 + 4 q30mm30 + 4 q31mm31 + 4 q31mm32 + 4 r3mm16 + 4 r3mm19 + 4 r3mm23 + 2 r9mm11 + 4 r9mm24 + 2 r10mm23 + 4 r10mm24 + 4 r11mm16 + 4 r11mm19 + 2 r11mm23 + 2 r15mm23 + 4 r15mm24 + 4 r16mm17 + 4 r16mm20 + 4 r16mm22 + 4 r16mm24 + 2 r17mm17 + 4 r17mm18 + 4 r17mm19 + 4 r17mm20 + 4 r17mm22 + 4 r18mm19 + 4 r18mm20 + 4 r18mm22 + 4 r18mm23 + 4 r18mm24 + 4 r19mm20 + 4 r19mm24 + 4 r20mm22 + 4 r20mm24 + 4 r22mm23 + 4 r22mm24 + 2 r24mm24);

e77 = 1/630 (0.0+ 17 q6mm11 + 35 q20mm20 + 36 r3mm23 + 72 r8mm24);

e78 = 1/630 (0.0+ 5 q6mm24 + 5 q11mm17 + 11 q20mm30 + 11 q20mm31 + 6 q22mm28 + 6 q26mm32 + 12 r3mm23 + 3 r6mm11 + 5 r7mm23 + 24 r8mm24 + 3 r11mm14 + 5 r11mm23 + 5 r12mm23 + 5 r13mm23 + 11 r16mm24 + 11 r17mm24 + 6 r18mm18 + 11 r19mm24 + 12 r20mm24 + 6 r22mm22);

e79 = 1/630 (0.0+ 3 q6mm32 + 3 q11mm28 + 6 q28mm30 + 6 q30mm30 + 6 q31mm31 + 6 q31mm32 + 5 r3mm23 + 5 r11mm16 + 5 r11mm19 + 12 r11mm23 + 11 r16mm24 + 5 r17mm23 + 12 r17mm24 + 5 r18mm23 + 11 r18mm24 + 11 r19mm24 + 6 r20mm23 + 11 r20mm24 + 5 r22mm23 + 11 r22mm24 + 11 r23mm24 + 12 r24mm24);

e80 = 1/630 (0.0+ 36 r11mm23 + 72 r23mm24 + 72 r24mm24);

e81 = 1/630 (0.0+ 4 q1mm4 + 2 q1mm15 + 8 q2mm4 + 4 q2mm12 + 4 q2mm13 + 8 q3mm4 + 4 q3mm8 + 4 q3mm9 + 5 q4mm4 + 4 q4mm5 + 8 q4mm7 + 4 q4mm8 + 4 q4mm10 + 4 q4mm12 + 4 q4mm14 + 4 q4mm16 + 4 q5mm9 + 4 q5mm13 + 4 q5mm15 + 4 q5mm16 + 2 q5mm18 + 2 q5mm19 + 2 q5mm21 + 2 q5mm23 + 2 q5mm25 + 2 q5mm27 + 4 q9mm14 + 4 q10mm13 + 2 q16mm16 + 4 r1mm2 + 4 r1mm4 + 4 r2mm4 + 2 r5mm9 + 2 r5mm10 + 2 r5mm15);

e82 = 1/630 (0.0+ 5 q2mm4 + 5 q2mm24 + 5 q3mm4 + 5 q3mm17 + 5 q4mm8 + 5 q4mm10 + 5 q4mm12 + 5 q4mm14 + 5 q4mm22 + 5 q4mm26 + 5 q5mm21 + 5 q5mm23 + 5 q5mm25 + 5 q5mm27 + 5 q7mm8 + 5 q7mm12 + 5 q10mm18 + 5 q14mm19 + 3 r1mm6 + 3 r1mm14 + 3 r2mm2 + 5 r2mm12 + 3 r4mm4 + 5 r4mm7 + 5 r9mm13 + 3 r10mm10 + 3 r15mm15);

e83 = 1/630 (0.0+ 2 q2mm4 + 2 q2mm12 + 4 q2mm13 + 2 q2mm15 + 2 q2mm25 + 2 q3mm4 + 2 q3mm8 + 4 q3mm9 + 2 q3mm15 + 2 q3mm21 + 4 q4mm7 + 2 q4mm8 + 4 q4mm9 + 2 q4mm12 + 4 q4mm13 + 2 q4mm18 + 2 q4mm19 + 2 q5mm9 + 2 q5mm13 + 2 q5mm18 + 2 q5mm19 + 2 q5mm21 + 2 q5mm25 + 4 q7mm15 + 4 q7mm16 + 4 q8mm13 + 2 q8mm14 + 2 q8mm16 + 4 q9mm12 + 4 q9mm13 + 2 q9mm14 + 2 q9mm16 + 2 q9mm27 + 2 q10mm12 + 2 q10mm13 + 2 q10mm15 + 2 q10mm16 + 2 q10mm25 + 2 q10mm27 + 2 q12mm16 + 2 q13mm16 + 2 q13mm23 + 2 q14mm15 + 2 q14mm16 + 2 q14mm21 + 2 q14mm23 + 2 q16mm18 + 2 q16mm19 + 2 q16mm23 + 2 q16mm27 + 2 r1mm2 + 2 r1mm4 + 2 r1mm9 + 5 r2mm4 + 2 r2mm5 + 2 r2mm15 + 2 r4mm5 + 2 r4mm10 + 4 r5mm9 + 2 r5mm10 + 2 r5mm15 + 2 r10mm15);

e84 = 1/630 (0.0+ 1 q2mm4 + 1 q2mm12 + 1 q2mm13 + 1 q2mm15 + 3 q2mm24 + 1 q2mm32 + 1 q3mm4 + 1 q3mm8 + 1 q3mm9 + 1 q3mm15 + 3 q3mm17 + 1 q3mm28 + 1 q4mm6 + 1 q4mm11 + 1 q4mm17 + 2 q4mm20 + 1 q4mm21 + 1 q4mm22 + 1 q4mm23 + 1 q4mm24 + 1 q4mm25 + 1 q4mm26 + 1 q4mm27 + 1 q4mm30 + 1 q4mm31 + 1 q5mm21 + 1 q5mm23 + 1 q5mm25 + 1 q5mm27 + 1 q5mm28 + 1 q5mm30 + 1 q5mm31 + 1 q5mm32 + 1 q6mm13 + 1 q6mm24 + 1 q7mm8 + 1 q7mm12 + 1 q7mm18 + 1 q7mm19 + 1 q7mm21 + 1 q7mm22 + 1 q7mm23 + 1 q7mm25 + 1 q7mm26 + 1 q7mm27 + 1 q8mm8 + 1 q8mm13 + 1 q8mm14 + 1 q8mm16 + 1 q8mm18 + 1 q8mm20 + 1 q8mm22 + 1 q9mm10 + 1 q9mm11 + 1 q9mm12 + 1 q9mm20 + 1 q9mm22 + 1 q9mm24 + 1 q9mm30 + 1 q10mm12 + 1 q10mm16 + 2 q10mm18 + 1 q10mm21 + 1 q10mm23 + 1 q10mm25 + 1 q10mm27 + 1 q10mm30 + 1 q11mm17 + 1 q12mm12 + 1 q12mm16 + 1 q12mm19 + 1 q12mm20 + 1 q12mm26 + 1 q13mm14 + 1 q13mm17 + 1 q13mm20 + 1 q13mm26 + 1 q13mm31 + 1 q14mm16 + 2 q14mm19 + 1 q14mm21 + 1 q14mm23 + 1 q14mm25 + 1 q14mm27 + 1 q14mm31 + 1 q15mm22 + 1 q15mm26 + 1 q16mm21 + 1 q16mm23 + 1 q16mm25 + 1 q16mm27 + 1 q18mm20 + 1 q18mm22 + 1 q19mm20 + 1 q19mm26 + 1 r1mm6 + 1 r1mm14 + 1 r1mm16 + 1 r1mm19 + 1 r2mm3 + 1 r2mm7 + 2 r2mm8 + 1 r2mm10 + 1 r2mm12 + 1 r2mm13 + 1 r2mm17 + 1 r2mm20 + 1 r2mm22 + 1 r3mm4 + 1 r3mm6 + 1 r3mm14 + 1 r3mm16 + 1 r3mm19 + 1 r4mm7 + 2 r4mm8 + 1 r4mm12 + 1 r4mm13 + 1 r4mm15 + 1 r4mm17 + 1 r4mm18 + 1 r4mm20 + 1 r5mm16 + 1 r5mm18 + 1 r5mm19 + 1 r5mm22 + 1 r6mm12 + 2 r7mm8 + 1 r7mm9 + 1 r7mm14 + 2 r8mm12 + 2 r9mm13 + 1 r9mm18 + 1 r9mm22 + 1 r10mm10 + 1 r10mm17 + 1 r10mm18 + 1 r10mm20 + 1 r13mm16 + 1 r13mm19 + 1 r15mm15 + 1 r15mm17 + 1 r15mm20 + 1 r15mm22);

e85 = 1/630 (0.0+ 2 q2mm15 + 2 q2mm32 + 2 q3mm15 + 2 q3mm28 + 4 q6mm13 + 2 q6mm24 + 4 q9mm11 + 4 q9mm20 + 4 q9mm24 + 4 q9mm30 + 2 q11mm17 + 4 q13mm17 + 4 q13mm20 + 4 q13mm31 + 2 q15mm22 + 2 q15mm26 + 4 q16mm21 + 4 q16mm23 + 4 q16mm25 + 4 q16mm27 + 4 q18mm20 + 4 q18mm22 + 4 q19mm20 + 4 q19mm26 + 4 q20mm21 + 4 q20mm23 + 4 q20mm25 + 4 q20mm27 + 1 r1mm6 + 1 r1mm14 + 4 r3mm8 + 4 r3mm16 + 4 r3mm19 + 4 r5mm23 + 4 r5mm24 + 2 r6mm12 + 4 r7mm8 + 2 r7mm14 + 4 r8mm8 + 4 r8mm12 + 4 r8mm13 + 4 r8mm17 + 4 r8mm20 + 4 r9mm18 + 4 r9mm22 + 2 r10mm10 + 4 r10mm24 + 2 r13mm23 + 2 r15mm15 + 4 r15mm24);

e86 = 1/630 (0.0+ 2 q6mm13 + 1 q6mm15 + 4 q6mm24 + 2 q6mm32 + 2 q9mm11 + 2 q9mm32 + 1 q11mm15 + 4 q11mm17 + 2 q11mm28 + 2 q13mm28 + 2 q16mm28 + 2 q16mm32 + 4 q18mm20 + 4 q18mm30 + 4 q19mm20 + 4 q19mm31 + 4 q20mm23 + 4 q20mm27 + 4 q20mm30 + 4 q20mm31 + 4 q21mm22 + 4 q21mm30 + 2 q23mm23 + 4 q25mm26 + 4 q25mm31 + 2 q27mm27 + 2 r3mm6 + 2 r3mm14 + 4 r3mm16 + 4 r3mm19 + 5 r3mm23 + 2 r5mm23 + 4 r7mm19 + 8 r8mm17 + 4 r8mm18 + 4 r8mm20 + 4 r8mm22 + 4 r8mm24 + 8 r9mm24 + 4 r10mm18 + 4 r10mm24 + 4 r12mm16 + 4 r13mm23 + 4 r15mm22 + 4 r15mm24 + 4 r16mm22 + 4 r17mm20 + 4 r18mm19 + 4 r20mm24);

e87 = 1/630 (1.0+ 13 q7mm15 + 13 q8mm13 + 13 q9mm12 + 13 q9mm13 + 13 q10mm25 + 13 q14mm21 + 13 q16mm18 + 13 q16mm19 + 13 q23mm27 + 13 r2mm4 + 13 r5mm9 + 13 r10mm15);

e88 = 1/630 (0.0+ 1 q2mm13 + 1 q2mm24 + 1 q2mm25 + 1 q2mm32 + 1 q3mm9 + 1 q3mm17 + 1 q3mm21 + 1 q3mm28 + 1 q6mm12 + 1 q6mm13 + 1 q6mm15 + 1 q6mm24 + 1 q6mm25 + 1 q6mm32 + 1 q7mm18 + 1 q7mm19 + 1 q7mm21 + 1 q7mm23 + 1 q7mm25 + 1 q7mm27 + 1 q8mm11 + 2 q8mm18 + 1 q8mm21 + 1 q8mm27 + 1 q8mm30 + 1 q9mm11 + 1 q9mm22 + 1 q9mm23 + 1 q9mm25 + 1 q9mm30 + 1 q9mm31 + 1 q9mm32 + 1 q10mm18 + 1 q10mm19 + 1 q10mm21 + 1 q10mm27 + 1 q10mm31 + 1 q10mm32 + 1 q11mm15 + 1 q11mm17 + 1 q11mm21 + 1 q11mm28 + 2 q12mm19 + 1 q12mm23 + 1 q12mm25 + 1 q12mm31 + 1 q13mm21 + 1 q13mm26 + 1 q13mm27 + 1 q13mm28 + 1 q13mm30 + 1 q13mm31 + 1 q14mm18 + 1 q14mm19 + 1 q14mm23 + 1 q14mm25 + 1 q14mm28 + 1 q14mm30 + 1 q15mm30 + 1 q15mm31 + 1 q16mm17 + 2 q16mm20 + 1 q16mm24 + 1 q16mm28 + 1 q16mm30 + 1 q16mm31 + 1 q16mm32 + 1 q17mm27 + 1 q18mm18 + 1 q18mm21 + 1 q18mm27 + 1 q18mm30 + 1 q19mm19 + 1 q19mm23 + 1 q19mm25 + 1 q19mm31 + 1 q20mm22 + 1 q20mm23 + 1 q20mm26 + 1 q20mm27 + 1 q20mm30 + 1 q20mm31 + 1 q21mm22 + 1 q21mm27 + 1 q21mm30 + 1 q22mm30 + 1 q23mm23 + 1 q23mm24 + 1 q23mm25 + 1 q25mm26 + 1 q25mm31 + 1 q26mm31 + 1 q27mm27 + 1 r1mm16 + 1 r1mm19 + 1 r2mm3 + 2 r2mm17 + 1 r2mm19 + 1 r2mm20 + 1 r2mm22 + 1 r3mm4 + 1 r3mm6 + 2 r3mm9 + 1 r3mm14 + 3 r3mm16 + 3 r3mm19 + 1 r4mm16 + 2 r4mm17 + 1 r4mm18 + 1 r4mm20 + 1 r5mm16 + 1 r5mm18 + 1 r5mm19 + 1 r5mm22 + 1 r6mm22 + 1 r7mm9 + 1 r7mm10 + 1 r7mm19 + 1 r7mm20 + 2 r8mm10 + 2 r8mm15 + 2 r8mm18 + 2 r8mm22 + 1 r9mm12

+ 2 r9mm17 + 1 r9mm18 + 2 r9mm20 + 1 r9mm22 + 1 r10mm13 + 2 r10mm17 + 1 r10mm18 + 1 r10mm19 + 1 r10mm20 + 1 r10mm22 + 1 r12mm15 + 1 r12mm16 + 1 r12mm20 + 1 r13mm15 + 1 r13mm16 + 1 r13mm19 + 1 r14mm18 + 1 r15mm16 + 2 r15mm17 + 1 r15mm18 + 1 r15mm20 + 1 r15mm22 + 1 r16mm22 + 1 r18mm19 + 1 r18mm20 + 1 r20mm22);

e89 = 1/630 (0.0+ 5 q6mm25 + 3 q6mm32 + 5 q11mm21 + 3 q11mm28 + 3 q18mm18 + 5 q18mm21 + 5 q18mm27 + 3 q19mm19 + 5 q19mm23 + 5 q19mm25 + 5 q21mm30 + 5 q23mm31 + 5 q23mm32 + 5 q25mm31 + 5 q27mm28 + 5 q27mm30 + 3 q30mm30 + 3 q31mm31 + 5 r3mm9 + 5 r3mm16 + 5 r3mm19 + 5 r9mm17 + 5 r9mm20 + 5 r10mm17 + 5 r10mm19 + 5 r10mm22 + 5 r15mm16 + 5 r15mm17 + 5 r15mm18 + 5 r16mm22 + 5 r18mm19 + 5 r18mm20 + 5 r20mm22);

Bibliography

- [1] Lieven Vandenbergh, and Stephen Boyd: Semidefinite Programming (1996 Society for Industrial and Applied Mathematics SIAM REVIEW, Vol. 38, No. 1, pp. 49-95)
- [2] Michael J. Todd: Conic Programming (<https://people.orie.cornell.edu/miketodd/iccopt.pdf>)
- [3] Chad Wildman: A Proof of Tychonoff's Theorem (<http://www.math.ucsd.edu/~cwildman/qualprep-tychonoff.pdf>)
- [4] Marcel K. De Carli Silva, Fernando Mario De Oliveira Filho, and Cristiane Maria Sato: Flag Algebras: A First Glance (<https://arxiv.org/abs/1607.04741>)
- [5] Bernard Lidicky: Flag Algebras hopefully simple basics <https://kam.mff.cuni.cz/bernard/slides/2014-aim.pdf>
- [6] Andrzej Grzesik: Flag Algebras in Extremal Graph Theory PHD Thesis (<http://ssdm.mimuw.edu.pl/pliki/prace-studentow/st/pliki/andrzej-grzesik-d.pdf>)
- [7] Andrzej Grzesik: On the maximum number of five-cycles in a triangle-free graph (Journal of Combinatorial Theory, Series B, Volume 102, Issue 5, September 2012, Pages 1061-1066)
- [8] Paul Erdos: On some problems in Graph Theory, Combinatorial Analysis and Combinatorial Number Theory(Proc. Conf. Hon. P. Erdos, Cambridge, 1983, pp. 1–17, 1984)
- [9] Kris Sey: The Erdos Pentagon Problem (UWSpace. <http://hdl.handle.net/10012/14274>)
- [10] Andrzej Grzesik: On the maximum number of five-cycles in a triangle-free graph (<https://arxiv.org/abs/1607.04741>)
- [11] Alexander Razborov: Flag Algebra (Journal of Symbolic Logic, 72(4):1239–1282, 2007)
- [12] Alexander Razborov: On the minimal density of triangles in graphs, Combinatorics, Probability and Computing, 17(4), 603-618
- [13] Rahil Baber: Turán densities of hypercubes (arXiv:1201.3587v2)
- [14] Bernard Lidický, Florian Pfender: Semidefinite Programming and Ramsey Numbers: (arXiv:1704.03592)
- [15] Jozsef Balogh, Ping Hu, Bernard Lidicky, Oleg Pikhurko, Balazs Udvari, Jan Volec: Minimum number of monotone subsequences of length 4 in permutations (arXiv:1411.3024)