ON THE ANALYSIS OF SRG$(85, 14, 3, 2)$

by

TIANXIAO ZHAO

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Mathematics
College of Engineering and Physical Sciences
The University of Birmingham
February 7, 2024

ACKNOWLEDGEMENTS

Throughout the writing of this thesis I have received a great deal of support and assistance.

I would first like to thank my supervisor, Professor Sergey Shpectorov, whose expertise was invaluable in formulating the research questions and methodology. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

I would also like to thank my tutors, Professor Chris Parker, Doctor Johannes Carmesin, and Professor Marta Mazzocco, for their valuable guidance throughout my studies. You provided me with the tools that I needed to choose the right direction and successfully complete my dissertation.

Finally, I would like to thank my parents for their wise counsel and sympathetic ear. You are always there for me.

Abstract

We investigate the second smallest unresolved feasible set of parameters of strongly regular graphs, $(v, k, \lambda, \mu) = (85, 14, 3, 2)$. We start with the general properties of the 34-dimensional Euclidean representation of such a graph G, and then focus on the special subgraphs called segments, which are edge complements in the local subgraphs of G. Since local subgraphs are cubic graphs of order 14, we utilize the available classification of all such cubic graphs. After enumerating possible segments, pairs of segments, and triples of segments, we concentrate on a configuration of four segments build around a maximal 3-clique of G. We enumerate all such configuration and eliminate them case by case, using both combinatorial and metric properties of G. Thus we show that there exists no strongly regular graph with parameters $(85, 14, 3, 2)$. The method we use is quite general and we hope that it can be applied in the future in other unresolved cases with small λ and μ.

CONTENTS

INTRODUCTION

Strongly regular graphs were first introduced by R.C. Bose in 1963 [4]. The topic of strongly regular graphs is an area where statistics, Euclidean geometry, group theory, finite geometry, and algebraic combinatorics meet. Strongly regular graphs have very interesting algebraic and combinatorial properties due to their strong regularity conditions. A strongly regular graph has four parameters to describe its properties: the number of vertices v, the valency k, the number of neighbours of any two adjacent vertices, λ, and the number of neighbours of any two non-adjacent vertices, μ. The four parameters are not independent; in fact, they satisfy several condition known as feasibility conditions. For the set of feasible parameters (i.e. the quadruples (v, k, λ, μ) that satisfies the feasibility conditions), an interesting problem arises and it has been on the forefront of research for many years: Does there exist a strongly regular graph with given parameters? For some feasible parameters, there are famous families of graphs, such as, say, *Triangular graphs* $T(n)$, *orthogonal arrays* $OA(k, n)$, *Paley graphs* Paley(n), etc., showing that the given parameters correspond to actual examples of strongly regular graphs (called SRG in short in the remaining of this thesis). Typically, there would be a unique SRG for each such quadruple (v, k, λ, μ). For example, the pentagon is the unique strongly regular graph with parameters $(5, 2, 0, 1)$. In other cases, it has been shown that there are no SRG with a given feasible set of parameters. For example, F.C. Bussemaker, W.H. Haemers, R. Mathon and H.A. Wilbrink [9] show that there are no strongly regular graphs with parameters $(49, 16, 3, 6)$. Finally, there are feasible sets of parameters for which a wildly large number of SRGs have been shown to exist. For example, W.H. Haemers and E. Spence [13] show that there are 167 srg$(64, 18, 2, 6)$ up to isomorphism by complete

enumeration.

A.E. Brouwer maintains the online list [6] of feasible parameters with $n \leq 1300$ where the information about the status of each case is given. As of today, there are only nine sets of parameters with $v \leq 100$ (as shown in the following table)

v	k	λ	μ
68	20	7	5
85	14	3	2
85	30	11	10
85	42	20	21
88	27	6	9
96	35	10	14
99	14	1	2
99	42	21	15
100	33	8	12

for which the existence of the corresponding strongly regular graph has not been decided.

This thesis demonstrate the non-existence of strongly regular graphs in the second smallest of these cases from the algebraic point of view. Namely, we intend to prove the following conjecture:

Conjecture 0.0.1. There is no strongly regular graph with parameters $(85, 14, 3, 2)$.

There are several methods that were employed to show that a particular set of feasible parameters does not yield any actual strongly regular graphs. For quite a few parameter sets, they were eliminated by Krein's conditions (see Section 2.3) or the absolute bound ([22] Theorem 21.4). On the other hand, some sets of parameters are shown to be impossible by various specialized or even ad-hoc methods. In 1983, H.A. Wilbrink and A.E. Brouwer proved that an $\mathrm{srg}(57, 14, 1, 4)$ does not exist [26]. They focused on a coclique in such a graph G, and gave an upper bound of the size of it. Then they split into two cases depending on whether or not G contains a 15 coclique, the largest

possible size. Both cases led to a contradiction. In 1993, W.H. Haemers [20] proved that if a graph G is an $\mathrm{srg}(76, 21, 2, 7)$, then it had to be geometric, that is, coming from a geometry, and drew a contradiction with the known fact that no such geometry existed. Another example is [3]. In 2012, A.V. Bondarenko and D.V. Radchenko gave a complete description of a family of strongly regular graphs with $\lambda = 1$, limiting all possible such graphs. As a corollary, quite a lot of sets of parameters were shown to became impossible.

Some of the results on strongly regular graphs also came from the research on other topics. In 1960, A.J. Hoffman and R.R. Singleton [14] showed the uniqueness of Moore graphs with diameter 2 and 3. In the case of diameter 2, they first computed the eigenvalues of the graph, then considered the case where the eigenvalues were rational. (Note that nowadays we know that they are rational and even integer for any strongly regular graph.) The number of vertices n which satisfy this condition is one of 2, 10, 50, or 3250. They proved that the case $n = 50$ contains a unique graph by analyzing the length of cycles in the graph, and the corresponding block design. Hence the only $\mathrm{srg}(50, 7, 0, 1)$ has been found, now known as the *Hoffman-Singleton graph*. Also, in 2020, A.A. Makhnev [17] proved that the Moore graph with parameters $(3250, 57, 0, 1)$ does not exist. In 1968, J.J. Seidel [19] analyzed a class of strongly regular graphs with $(-1, 1, 0)$-adjacency matrix. He considered such graphs with exactly three distinct eigenvalues, with one of them being 3. He then discussed several different cases of possible spectra of such graphs. In one of the cases ($v = 27$, $\rho_1 = 3$, $\rho_2 = -9$, $\rho_0 = -6$), the only $\mathrm{srg}(27, 16, 10, 8)$ is the Schläfli graph. In 2001, B.D. McKay and E. Spence [20] did an exhaustive enumeration of regular two-graphs on 36 and 38 vertices, which gave a complete enumeration of $\mathrm{srg}(35, 16, 6, 8)$, $\mathrm{srg}(36, 14, 4, 6)$, and $\mathrm{srg}(36, 20, 10, 12)$.

In [1], M. Alfuraidan, I. Sarumi, and S. Shpectorov proved non-existence of $\mathrm{srg}(76, 21, 2, 7)$ by considering the Euclidean representation of the graph, and they show the graph is locally a union of cycles, then manage to get a contradiction in each cases. In our present case, we attempt an exhaustive enumeration using the similar conditions as in [1] from the Euclidean representation to cut off impossible branches in the enumeration tree. The

enumeration in this case is enormous, and so we have to be clever to implement it in such a way so that it can finish in a reasonable time. To minimize the enumeration, we work in a fragment of the graph consisting of four vertex neighbourhoods. We use a decomposition of four steps where in each step we identify a larger and larger part of this fragment of graph, trying to get a contradiction with the combinatorial properties of the graph and the semi-positive definiteness of the Gram matrix of the graph, coming from the Euclidean representation.

Let us now discuss the contents of this thesis. In Chapter 1, we introduce the notation and terminology from graph theory. Starting with the basics of graphs, together with some examples, we discuss the adjacency matrix and spectrum of a graph. Eigenvectors and the corresponding eigenspaces plays an important role in this thesis. After that, we introduce strongly regular graphs, which are our research object. We end this chapter with an introduction of distance regular graphs. The reason why we have this is that we will treat $\mathrm{srg}(85, 14, 3, 2)$ as a distance regular graph of diameter 2, because we want to use the theory of association schemes developed in this wider context.

In Chapter 2, we talk about the association schemes and the Bose-Mesner algebra, which are closely related with distance regular graphs. The first two sections cover some basics. In the theory of the association schemes, every Bose-Mesner algebra has two standard bases coming from the two dual algebra structures, one from the usual matrix multiplication, and the other one from the Hadamard product. This leads to the Krein parameters, which is the content of the third section. In the fourth section we talk about the $P-$polynomial and $Q-$polynomial association schemes, which are a pair of dual properties related with the two different bases. The last section contains the theory of Euclidean representations and provides the details about the cosine sequence.

In Chapter 3, we do some basic analysis of the graph $\Gamma = \mathrm{srg}(85, 14, 3, 2)$. To begin with, we compute the spectrum of Γ, which is $\{1^1, 4^{34}, -3^{50}\}$. We also find the basic properties of Euclidean representation with respect of the eigenvalue $\theta = 4$ of Γ, get the cosine sequence of this graph, which will be used later in this thesis. We then focus on the

neighbourhood subgraphs of Γ and collect some useful facts in the following two sections. For example, we prove that there must be a maximal 3-clique in Γ. As the last part of this chapter, we define a structure called segments, establishing important properties of it.

Chapter 4 contains the first step of our work. We first enumerate all possible segments in the graph Γ. We then introduce amalgams of graphs, and establish the theory of gluing two segments along a handle. We then discuss the enumeration of segment pairs, which are amalgams consisting of two glued segments. We prove that there is a matching between certain vertices in the two segments. In the next section of this chapter, we show how a symmetric semi-positive definite $n \times n$ matrix can be decomposed into the product of a triangular matrix L, a diagonal matrix D, and the transpose L^T. This is called the LDLT decomposition. We need this algorithm to check whether a possible Gram matrix of a set of vectors from our Euclidean representation is semi-positive definite. We realize this in GAP, based on the code contributed by Madeleine Whybrow (the code can be found in Appendix). In the last part of this section, we create a function AddOne. This function is specifically created for this project, and it is useful when we already have the LDLT decomposition of a symmetric matrix A, and then we can quickly do the decomposition of a symmetric matrix A', which formed by adding a row and a column to A, instead of decomposing A' from scratch. Therefore, when we are adding one vertex to a possible subgraph, we will apply this function to check whether or not the new configuration is possible.

Chapter 5 continues the discussion of how our enumeration algorithm works. Firstly, we construct the full list of possible enumeration trees for gluing two segments at the first step. The enumeration tree allows to enumerate all possible matchings on a given order. At this step we only have two trees depending on the type 4 and type 6 matching. After that, at step 2, we consider gluing the third segment to the segment pair with a possible matching inside the pair. This leads to the segment triple structure, which is the amalgam consisting of the segment pair and the third segment. We use similar method

as we did in Chapter 4, but this time since we have two handles to glue, there are more cases to consider for each triple. By similar argument, there should be matchings between the third segment and the first two segments. They are independent but the matching vertex sets in the third segment are always a subset of the eight vertices that are not in either handles. For each vertex, it can be in the matchings of neither sides, or only matching with the first segment, or only matching with the second segment, or matching with both. By ordering these vertices in the order mentioned above and counting the number of vertices of each type, there will be 11 different quad types of segment triples. According to this fact, we construct the full list of possible big enumeration trees for gluing a segment pair and the third segment at the second step. These trees depends on the quad type of the segment triple.

Chapter 6 explains the third and the fourth steps of our work. For the segment triples that pass the first two steps, we aim to add the neighbourhood of the vertex number 13. This is because vertex number 13 is in the handle between the second and the third segment, it has six neighbours inside the segment triple and the 3-clique. Notice that in total there will be eight more vertices. The preliminary of Chapter 6 include the Gram-Schmidt process work, and the relation of it with the LDLT decomposition. Besides, we develop a method allowing the quick computation of the projection of a vertex onto the subspace spanned by the current triple of segments. For the vertices that are not in the segment triple, they need to satisfy several conditions. Firstly, for each vertex, its projection onto the space spanned by the segment triple should be in a reasonable range. Secondly, since this vertex is not adjacent to any vertex in the 3-clique, it should have two neighbours in every segment. Besides, we also have the parameters condition to limit the number of common neighbours of any pair of vertices. Based on these conditions we get a set of all possible cases, and we aim to select eight vertices and add to the segment triple. Since we are selecting eight out of hundreds of possible new vertices, it is not wise to check them one by one. We set a function to check whether two vertices are compatible or not, and we also eliminate cases by the parameter condition of Γ. Finally at step 4, for

the eight newly added vertices, we check the edges between them, and using the condition on common neighbours to eliminate the configuration. For any cases that passes all the checks above, we check the semi-positive definition condition of the Gram matrix on these 38 vertices. Since the dimension of the eigenspace is only 34, the rank of Gram matrix should not exceed 34. This is a very strong condition but so far in the programme we have not seen any configuration reaches this condition.

In the end there is a brief overview of the result and conclusion.

All our GAP code is included in the Appendix.

CHAPTER 1

BACKGROUND

1.1 The basics of graph theory

The study of graphs is often viewed as a study of the modeling of relations between objects.

Definition 1.1.1 (Graph)**.** A *graph* $G = (V, E)$ consists of a set V of *vertices* and a set E of *edges*. The edges are a symmetric, non-reflexive binary relation over V, denoted as $e : x \sim y$ or $e = xy$ where $e \in E$ and $x, y \in V$. We say that G is a graph on V.

Remark. We say a graph G is simple if it has no loops and there is at most one edge between two different vertices. Throughout this thesis the graphs we mentioned are simple graphs unless specifically mentioned.

The following are some important notations and terminologies that will appear in this paper:

- Let G be a graph with sets V as vertices and E as edges. The *vertex set* is referred to as $V(G) = V$ and its *edges set* as $E(G) = E$.
- If $x \sim y$, we say that x is *adjacent* to y or x is a *neighbour* of y, and e is *incident* with x and y. The set of all vertices that are adjacent to a vertex x is called the *neighbourhood* of x, denoted as $N(x)$.

- The *degree* or *valency* of a vertex x is the number of vertices that are adjacent to x, i.e. the size of $N(x)$. We say that a graph G is *k-regular* if every vertex in G has degree k.
- The number of vertices of a graph G is its *order*, and the number of edges is its *size*.
- A graph G of order n with any two vertices adjacent to each other is called a *complete graph* and denoted by K_n.

Definition 1.1.2 (The complement of a graph)**.** Let $G = (V, E)$ be a graph. We say $\bar{G}$ is the complement of G if $V(\bar{G}) = V(G)$ and $e \in E(\bar{G})$ if and only if $e \notin E(G)$.

Definition 1.1.3 (Bipartite graph)**.** Let $G = (V, E)$ be a graph. We say that G is *bipartite* if V can be partitioned into two sets V_1, V_2 such that the edges of G are only between vertices in V_1 and vertices in V_2. If each vertex in V_1 is adjacent to all vertices in V_2, then we say that this is a *complete bipartite graph*, denoted by $K_{m,n}$, where $m = |V_1|$ and $n = |V_2|$.

Definition 1.1.4 (Graph homomorphism)**.** Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. If there exists a mapping $\varphi : V \to V'$, such that if for all $x, y \in V$, $xy \in E$, then $\varphi(x)\varphi(y) \in E'$, then φ is called an *homomorphism*. If if φ is bijective and φ^{-1} is also a graph homomorphism, we say that φ is an *isomorphism*. In this case G and G' are said to be *isomorphic*, denoted as $G \simeq G'$. In case of $G = G'$ we call φ an *automorphism*.

Definition 1.1.5 (Automorphism group of a graph)**.** Let G be a graph, the group of all automorphisms on G is called the *automorphism group* of G, denoted as $\operatorname{Aut} G$.

Definition 1.1.6 (Subgraph)**.** Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. If $V' \subset V$ and $E' \subset E$ then we say that G' is a *subgraph* of G. We say that G' is a *proper subgraph* of G if $G \neq G'$.

Definition 1.1.7 (Induced subgraph)**.** Let $G = (V, E)$ be a graph, and $G' = (V', E')$ be a subgraph of G. If for any $v_1, v_2 \in V'$ and $v_1v_2 \in E$ then $v_1v_2 \in E'$, then we call G' is a *induced subgraph* of G over the vertex set V'.

Definition 1.1.8 (Path)**.** A *path* p from v_1 to v_2 in a graph $G = (V, E)$ is a sequence $p = (e_1, e_2, ..., e_n)$ of edges where e_1 is incident with v_1, e_n is incident with v_2, and for all $1 \leq i \leq n-1$, e_i and e_{i+1} are incident with a same vertex. The *length* of path p is n, denoted as $l(p) = n$.

1.2 Algebraic graph theory

1.2.1 Adjacency matrix

Matrices are very useful for describing and analyzing graphs. In order to represent a graph algebraically, and investigate properties such as the structure of a graph, isomorphic graphs, eigenvalues and eigenvectors, we introduce the concept of the *adjacency matrix*. This section is mainly based on [15]. In this section, for a graph G on n vertices, let $V(G) = \{v_1, v_2, \ldots, v_n\}$ be its vertex set.

Definition 1.2.1. The adjacency matrix A of the graph $G = (V, E)$ is a square zero-one matrix of order $|V|$, the size of the vertex set. Its entries are defined as following:

$$a_{ij} = \begin{cases} 1, & \text{if } v_i \sim v_j, \\ 0, & \text{if } v_i \not\sim v_j. \end{cases}$$

Since G is a simple graph, A is a symmetric matrix with $a_{ii} = 0$, for all i. We also use the notation $A(G)$ to denote the adjacency matrix of G.

The following theorem gives a relation between isomorphic graphs.

Theorem 1.2.2. Let $G = (V, E)$ and $G' = (V', E')$ be two simple graphs with $|V| = |V'| = n$. For a bijection $f : V \to V'$, let the permutation $\sigma = \sigma(f) \in S_n$ be defined as follows:

$$\sigma(i) = j \text{ if and only if } f(v_i) = v'_j.$$

Then f is an isomorphism between G and G' if and only if

$$A(G') = P_\sigma A(G) P_\sigma^{-1},$$

where P_σ is the $n \times n$ permutation matrix with respect to $\sigma = \sigma(f)$.

Proof. Suppose that f is an isomorphism, then by definition, G can be transformed into G' by a bijective mapping between vertices. At the same time, in $A(G)$, rows and columns are permuted respectively. If we denote P_σ as the respective row permutation matrix then P_σ^{-1} is the column permutation matrix and hence $A(G') = P_\sigma A(G) P_\sigma^{-1}$.

On the other hand, suppose $A(G') = P_\sigma A(G) P_\sigma^{-1}$, then consider the following mapping $\sigma : V \to V'$, $v_i \mapsto v_{\sigma(i)}$:

$$a_{ij} = 1 \Leftrightarrow a_{f(i),f(j)} = 1$$

This is an isomorphism between G and G'. □

Recall that diag $\{B_1, \ldots, B_s\}$ denotes a block diagonal matrix with blocks $B_1, \ldots, B_s$, we have the following theorem on graphs with several components:

Theorem 1.2.3. The graph G has s connected components $G_1, \ldots, G_s$ if and only if there exist a permutation matrix P such that:

$$PA(G)P^{-1} = \text{diag}\,\{A(G_1), \ldots, A(G_s)\}$$

$A(G_i)$ are matrix blocks that cannot be block diagonalized into more blocks.

Proof. Let V_i be the vertex set of G_i, then V can be partitioned into $V_1, \ldots, V_s$ such that there are no edges between any V_i and V_j with $i \neq j$, and the number of parts is maximum. Re-number the vertices of G so that the first $|V_1|$ vertices are all the vertices in V_1, then all the vertices in V_2, and so on. Since there are no edges between two different components, the adjacency matrix of the re-numbered graph is a block diagonal matrix. □

Remark. Such form are called the *block triangular form* or *Frobenius form.*

As an application of the adjacency matrix, here we have a very interesting fact:

Theorem 1.2.4. Let $G = (V, E)$ be a graph and $A(G)$ be its adjacency matrix, then a_{ij}^r, the entry of $(A(G))^r$, is the number of paths between x_i and x_j of length r.

Proof. We will prove this by induction. For the case $r = 1$ it is obvious by the definition of edge. For the case $r = 2$, noticing that:

$$a_{ij}^2 = \sum_{k=1}^{n} a_{ik} a_{kj}. \tag{1.1}$$

We can see that a_{ij}^2 has a contribution whenever $a_{ik} = a_{kj} = 1$ for some k, i.e. $v_i \sim v_k$ and $v_k \sim v_j$, which forms a path of length 2.

Now suppose the statement is true for the case $r = m$. We treat $(A(G))^{m+1}$ as $(A(G))^{m+1} = (A(G))^m (A(G))$, then we have:

$$a_{ij}^{m+1} = \sum_{k=1}^{n} a_{ik}^m a_{kj}. \tag{1.2}$$

Similarly, a_{ij}^{m+1} has a contribution whenever $a_{ik}^m = a_{kj} = 1$ for some k, which means there is a path of length m from v_i to v_k, and $v_k \sim v_j$, which forms a path of length $m + 1$. □

1.2.2 The eigenvalues

Now we analyze a graph using its adjacency matrix. As the eigenvalues of a matrix are invariant under conjugation, they describe some properties of the graph, which are invariant under automorphisms and, more generally, renumbering of vertices. Let us recall the basic definitions.

Definition 1.2.5 (Spectrum)**.** Let G be a graph. If $A(G)$ has distinct eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_s$ with multiplicities $m_1, m_2, \ldots, m_s$ then we shall write

$$\mathrm{Spec}(G) = \{\lambda_1^{m_1}, \lambda_2^{m_2}, \ldots, \lambda_s^{m_s}\}.$$

And we call this multiset the *spectrum* of G.

For a matrix A, let $\varphi(A)$ be its characteristic polynomial. We define $\varphi(G) = \varphi(A(G))$ to be the characteristic polynomial of a graph G. We have the following theorem on characteristic polynomials:

Theorem 1.2.6. If G has components $G_1, \ldots, G_k$, then

$$\varphi(G) = \varphi(G_1) \cdot \ldots \cdot \varphi(G_k)$$

This theorem follows immediately from Theorem 1.2.3 and the property of the determinant.

The following lemma and theorem will be useful later in the thesis.

Lemma 1.2.7. Let $A = A(G)$ be an adjacency matrix of graph G. Then all the eigenvalues of A are real numbers. Furthermore, A is diagonalizable.

Proof. Let $Ax = \lambda x$ where λ is an eigenvalue, $x \neq 0$ is an eigenvector with respect to λ. By taking the complex conjugate on both sides we have $\overline{Ax} = \overline{\lambda x}$ hence $A\overline{x} = \overline{\lambda}\overline{x}$. Then we have:

$$\overline{x}^T Ax = \overline{x}^T (Ax) = \overline{x}^T \lambda x = \lambda(\overline{x} \cdot x),$$

and

$$\overline{x}^T Ax = (A\overline{x})^T x = (\overline{\lambda}\overline{x})^T x = \overline{\lambda}(\overline{x} \cdot x).$$

Since $x \neq 0$, we have $\overline{x} \cdot x \neq 0$ hence $\overline{\lambda} = \lambda$ which means λ is a real number.

Now we prove that A is diagonalizable. Let x_1 be an eigenvector of A with respect to the eigenvalue θ, and $x_2 \in \mathbb{R}^n$ is such that $x_1 \perp x_2$, then we have:

$$x_1 \cdot (Ax_2) = x_1^T Ax_2 = x_1^T A^T x_2 = (Ax_1)^T x_2 = \theta x_1^T x_2 = 0$$

Let $\{x_1, \ldots, x_k\}$ be the maximal set of k linearly independent eigenvectors of A, V be the space spanned by these vectors, and $V^\perp$ be its orthogonal space, containing all vectors all

vectors orthogonal to V. Every vector in $v \in V^{\perp}$ is orthogonal to each x_i, and so is Av, hence $V^{\perp}$ is fixed under A. Therefore, we have a linear transformation $A|_{V^{\perp}} : V^{\perp} \to V^{\perp}$. This linear transformation has at least one eigenvector if $V^{\perp}$ is not empty, in which case there is another eigenvector $x_{k+1} \in V^{\perp}$ of A, orthogonal to, and thus linearly independent from, $\{x_1, \ldots, x_k\}$. We get a contradiction with the choice of the set of eigenvectors, hence $V = \mathbb{R}^n$.

Now we have n linearly independent eigenvectors of A, which are a basis for the diagonalization of A. □

Theorem 1.2.8. Eigenspaces with respect to distinct eigenvalues of a real symmetric matrix are orthogonal with respect to the dot product.

Proof. Let A be a real symmetric matrix with eigenvalues $\lambda_1 \neq \lambda_2$, and eigenvectors x_1, x_2 such that $Ax_1 = \lambda_1 x_1$ and $Ax_2 = \lambda_2 x_2$.

As $Ax_1 = \lambda_1 x_1$ we have $x_2^T A x_1 = \lambda_1 x_2^T x_1$, similarly we have $x_1^T A x_2 = \lambda_2 x_1^T x_2$. Since A is symmetric, we have:

$$x_2^T A x_1 = x_2^T A^T x_1 = (Ax_2)^T x_1 = x_1^T (Ax_2) = x_1^T A x_2.$$

Notice that $x_1^T x_2 = x_2^T x_1$, from the above equation we get $\lambda_1 x_1^T x_2 = \lambda_2 x_1^T x_2$. Since $\lambda_1 \neq \lambda_2$, we must have $x_1^T x_2 = 0$ and the statement is proved. □

Let us end this section with the following result.

Theorem 1.2.9. The number of connected components of a k-regular graph G equals to the multiplicity of the eigenvalue k.

Proof. Suppose G has l components $G_1, \ldots, G_l$ and vertex set $V(G) = \{v_1, \ldots, v_n\}$. Let the vectors $x_i \in \mathbb{R}^n$, $i = 1, \ldots, l$, be characteristic vectors of the components, i.e. $x_i(j) = 1$ if $v_j \in G_i$, and $x_i(j) = 0$ otherwise. Notice that different x_i's are orthogonal.

In order to prove that k has multiplicity l, we need to show that: each x_i is an eigenvector with eigenvalue k, and if u is an eigenvector with eigenvalue k then u is a

linear combination of x_i's.

The first claim is obvious so we focus on the second one. We prove it by contradiction. Suppose $u = (u^1, \ldots, u^n)$ is the eigenvector for eigenvalue k that is not a linear combination of x_i's, and with smallest support, i.e. with smallest number of non-zero entries. Without lost of generality, suppose $|u^1|$ is maximal, $x_1 > 0$ and $v_1 \in G_1$. We have:

$$ku^1 = \sum_{(v_1,v_j)\in E(G)} u^j \leq k \max_{(v_1,v_j)\in E(G)} |x^j| \leq ku^1.$$

Notice that the second term in the above inequality has exactly k non-zero terms, hence all of them should be u^1. Apply the same argument to each u^j for which $(v_1, v_j) \in E(G)$ and to all x_i's with $v_i \in G_1$. Hence all entries in G_1 are equal to u^1. Subtracting $u^1 x_1$ from u we will get counterexample with smaller support, a contradiction. □

1.3 Strongly regular graphs

This research is on a class of graphs called *strongly regular graphs*.

Definition 1.3.1. A *strongly regular graph* with integer parameters v, k, λ and μ, denoted as $\mathrm{srg}(v, k, \lambda, \mu)$, is a $k-$regular graph of order v, where any two adjacent vertices have exactly λ common neighbours and any two non-adjacent vertices have exactly μ common neighbours.

Lemma 1.3.2. The complement of a strongly regular graph $G = \mathrm{srg}(v, k, \lambda, \mu)$ is also a strongly regular graph $\overline{G} = \mathrm{srg}(v, v-k-1, v-2k+\mu-2, v-2k+\lambda)$.

Proof. It is easy to see that $\overline{G}$ is regular of valency $(v-k-1)$. For any two vertices v_i, v_j in $\overline{G}$, their common neighbour v_k in $\overline{G}$ is not adjacent to either of them in G. Hence the number of common neighbours of adjacent vertices in $\overline{G}$ is $v-(k-\mu)-(k-\mu)-\mu-2 = v-2k+\mu-2$. And the number of common neighbours of non-adjacent vertices in $\overline{G}$ is $v-(k-\lambda)-(k-\lambda)-\lambda = v-2k+\lambda$. □

We can see that the union of m k-cliques is an $\mathrm{srg}(km, k-1, k-2, 0)$. In this thesis we exclude these trivial cases, i.e. we require an $\mathrm{srg}(v, k, \lambda, \mu)$ and its complement to be connected.

Example 1.3.3. A famous example of strongly regular graph is the Petersen graph, which is $\mathrm{srg}(10, 3, 0, 1)$, with 10 vertices and 15 edges.

Example 1.3.4 (The lattice graph)**.** The *lattice graph* $L_2(m), m \geq 2$. The vertex set of $L_2(m)$ is the set $S \times S$, where $S = \{1, 2, \ldots, m\}$. Two vertices (x, y) and (x', y') are adjacent to each other if and only if they have a common coordinate value, i.e. $x = x'$ or $y = y'$. The lattice graph $L_2(m)$ is an $\mathrm{srg}(m^2, 2(m-1), m-2, 2)$.

Theorem 1.3.5. For an $\mathrm{srg}(v, k, \lambda, \mu)$, if A is its adjacency matrix, we have:

$$AJ = kJ, \qquad A^2 + (\mu - \lambda)A + (\mu - k)I = \mu J, \tag{1.3}$$

where I is the identity matrix and J is the all-1 matrix, both have the same order as A.

Proof. Every vertex is adjacent to k vertices, which means the sum of each row of A is k, hence the first equation holds.

Now we count every coefficient b_{ij} in $B = A^2$: the value of b_{ij} is the number of paths of length 2 from i to j, hence $b_{ii} = k$, $b_{ij} = \lambda$ if $i \sim j$, and $b_{ij} = \mu$ if $i \not\sim j$, $i \neq j$. Hence $B = kI + \lambda(A) + \mu A'$, where A' is the adjacency matrix of the complement graph of G. Since $A' = J - I - A$, we obtain $A^2 = kI + \lambda A + \mu(J - I - A)$, and this leads to the equation in the theorem □

As an application of these equations, we have the following theorem:

Theorem 1.3.6. If there is an $\mathrm{srg}(v, k, \lambda, \mu)$, then its spectrum is $\left\{k^1, r^f, s^g\right\}$, where

r, s, f, g are:

$$
\begin{aligned}
r &= \frac{1}{2}\left(\lambda - \mu + \sqrt{(\lambda-\mu)^2 + 4(k-\mu)}\right), \\
s &= \frac{1}{2}\left(\lambda - \mu - \sqrt{(\lambda-\mu)^2 + 4(k-\mu)}\right), \\
f &= \frac{1}{2}\left(v - 1 - \frac{2k + (v-1)(\lambda-\mu)}{\sqrt{(\lambda-\mu)^2 + 4(k-\mu)}}\right), \\
g &= \frac{1}{2}\left(v - 1 + \frac{2k + (v-1)(\lambda-\mu)}{\sqrt{(\lambda-\mu)^2 + 4(k-\mu)}}\right).
\end{aligned}
$$

Proof. Let A be the adjacency matrix of the graph. Since the graph is regular of degree k, each row has k 1's and the remaining entries are 0. Hence the all-one vector $j := (1, 1, \ldots, 1)^T$ is an eigenvector of A with eigenvalue k. By Theorem 1.2.9, the multiplicity of this eigenvalue is one because the graph is connected. Any other eigenvector a with respect to eigenvalue $\theta \neq k$, by Theorem 1.2.8, is orthogonal to j with respect to the standard dot product. In particular, this means that $j \cdot a = 0$. Hence by applying equation (1.1) we have:

$$
\begin{aligned}
A^2a + (\mu - \lambda)Aa + (\mu - k)Ia &= \mu Ja, \\
\theta^2 a + (\mu - \lambda)\theta a + (\mu - k)a &= 0.
\end{aligned}
$$

So we get

$$
\theta^2 + (\mu - \lambda)\theta + (\mu - k) = 0, \text{ since } v \neq 0. \tag{1.4}
$$

This equation has two solutions:

$$
r, s = \frac{1}{2}\left(\lambda - \mu \pm \sqrt{(\lambda-\mu)^2 + 4(k-\mu)}\right). \tag{1.5}
$$

Let f and g be the multiplicities of r and s. Then we have $1 + f + g = v$ and $\mathrm{tr}(A) = k + fr + gs = 0$.

Solving this system of linear equations in f and g, we obtain:

$$f = \frac{s - vs - k}{r - s} = \frac{1}{2}\left(v - 1 - \frac{2k + (v-1)(\lambda - \mu)}{\sqrt{(\lambda - \mu)^2 + 4(k - \mu)}}\right),$$
$$g = v - 1 - f = \frac{1}{2}\left(v - 1 + \frac{2k + (v-1)(\lambda - \mu)}{\sqrt{(\lambda - \mu)^2 + 4(k - \mu)}}\right).$$

□

Remark. f and g in Theorem 1.3.6 must be non-negative integers. This is one of the feasibility conditions for strongly regular graphs. More conditions can be found in [23]. There is a table [6] of feasible parameter arrays for $v \leq 1300$ including all available results.

1.4 Distance regular graphs

This research is on strongly regular graph, but here we would like to give a brief introduction in the area of distance regular graphs, since strongly regular graph is a special kind of distance regular graph, and some of theorems about distance regular graphs are useful in Section 2.

First, recall the definition of path (Definition 1.1.8) let us define the *distance* and *diameter* of a graph.

Definition 1.4.1. Let G be a connected simple graph, the *distance* between two vertices x and y in G is the length of the shortest path between them, and it is denoted by $d(x, y)$. The *diameter* $d = \max_{x,y \in V(G)} d(x, y)$ is the maximum distance in the graph.

Now we can define *distance regular graph.*

Definition 1.4.2 (Distance regular graphs)**.** A connected graph G with diameter D is *distance regular* if for all $i = 0, 1, \ldots, d$, there exists numbers a_i, b_i, c_i such that for any pair of vertices x, y with $d(x, y) = i$, there are a_i neighbours of y having distance i to x, b_i neighbours of y having distance $i + 1$ to x, and c_i neighbours of y having distance $i - 1$ to x. These numbers a_i, b_i, c_i are the *intersection numbers.* Obviously $c_0 = b_d = 0$.

Definition 1.4.3. Let G be a distance regular graph of diameter d. Define matrices $A_0, A_1, \ldots, A_d$ as following: $A_0 = I$, and for $k = 1, ..., d$, we have $A_k(i, j) = 1$ if $d(i, j) = k$, otherwise $A_k(i, j) = 0$. These matrices are called the *distance matrices* of G.

Example 1.4.4. A famous family of distance regular graphs are the *Johnson graphs* $J(n, k)$. The vertices of $J(n, k)$ are all the k-subsets of an n-set, and two vertices are adjacent if and only if their intersection as sets has exactly $k - 1$ elements. It is easy to show that two sets that meet in $k - i$ elements are at distance i in $J(n, k)$. Consider arbitrary vertices x, y in $J(n, k)$ with $d(x, y) = i$, the vertices that are adjacent to y and has distance i to x are those vertices whose intersection with x as sets have $k - i$ elements, and intersection with y as sets have $k-1$ elements. The number of such vertices is $a_i = i(n - i - k)$. The vertices that are adjacent to y and have distance $i + 1$ to x are those whose intersection with x as sets has $k - i - 1$ elements, and intersection with y as sets has $k - 1$ elements. The number of such vertices is $b_i = (k - i)(n - i - k)$. The vertices that are adjacent to y and have distance $i - 1$ to x are those whose intersection with x as sets has $k - i + 1$ elements, and intersection with y as sets have $k - 1$ elements. The number of these is $c_i = ki$. Since a_i, b_i, c_i are independent with the choice of vertices x, y, it shows that $J(n, k)$ is distance regular.

It is clear that $a_i + b_i + c_i = k$, for all $i = 0, 1, \ldots, d$. Therefore, since we can express a_i via the other intersection numbers, a distance regular graph can be described by the intersection array:

$$\{b_0, b_1, \ldots, b_{d-1}; c_1, c_2, \ldots, c_d\}$$

In this array, all parameters are positive integers, since $b_d = c_0 = 0$ are not included.

Example 1.4.5. The bipartite graph $K_{3,3}$ is a distance regular graph, with the intersection array $\{3, 2; 1, 3\}$.

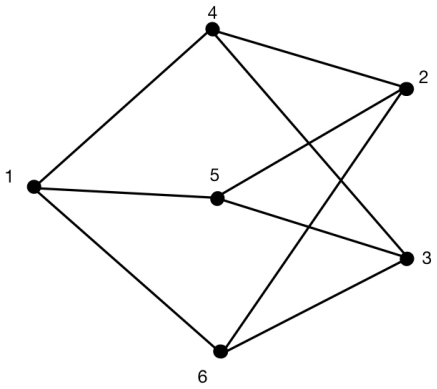


Figure 1.1: $K_{3,3}$ decomposed with respect to a vertex

The parameters of a distance regular graph have some useful properties:

Lemma 1.4.6. Let G be a distance regular graph with valency k and diameter d. For a fixed vertex v, let k_r be the number of vertices in G that at distance r from v. Then the following hold:

(1) $k_{i-1}b_{i-1} = k_i c_i$,

(2) if k_i is odd then a_i is even,

(3) $1 = c_1 \leq c_2 \leq \cdots \leq c_d$,

(4) $k = b_0 \geq b_1 \geq \cdots \geq b_{d-1}$,

(5) if $i + j \leq d$ then $c_j \leq b_i$.

Proof. For (1), let $N_i(v)$ be the set of vertices that are at distance i from v. Consider the number of edges between $N_i(v)$ and $N_{i-1}(v)$. The set $N_i(v)$ contains k_i vertices and each of them contributes c_i edges. Hence the total number of edges is $k_i c_i$. On the other hand, $N_{i-1}(v)$ contains k_{i-1} vertices and each contributes b_{i-1} edges. Therefore $k_i c_i = k_{i-1}b_{i-1}$, and (1) holds.

For (2), consider the subgraph induced on $N_i(v)$. It is a regular graph of valency a_i, and order k_i. Then the number of edges of this subgraph is $\dfrac{K_i a_i}{2}$, which is an integer. Hence if k_i is odd then a_i is even.

For (3), let $w \in N_{i+1}(v)$, $u \in N_i(w)$, and $u \sim v$. Then $w \in N_i(u)$. Any vertex adjacent to w and at distance $i-1$ from u is at distance i from v. Therefore $N_1(w) \cap N_{i-1}(u) \subset N_1(w) \cap N_i(u)$. Notice that $|N_1(w) \cap N_{i-1}(u)| = c_i$ and $|N_1(w) \cap N_i(u)| = c_{i+1}$, we get $c_i \leq c_{i+1}$ and (3) is proved. Similarly we can get $b_i \geq b_{i+1}$ and prove (4).

For (5), let u, v, w be vertices such that $d(u,v) = i$, $d(u,w) = i+j$ and $d(v,w) = j$. Any vertex x that is adjacent to v and satisfies $d(x,w) = j-1$, is at distance $i+1$ from u. Hence $c_j \leq b_i$. □

The relations (1) allow to compute the number k_i of vertices at distance i from any fixed vertex:

Corollary 1.4.7. For $i = 2, \ldots, d$ we have:

$$k_i = k\frac{b_1 \cdot \ldots \cdot b_{i-1}}{c_2 \cdot \ldots \cdot c_i}.$$

Moreover, the following *distance distribution diagram* is helpful.

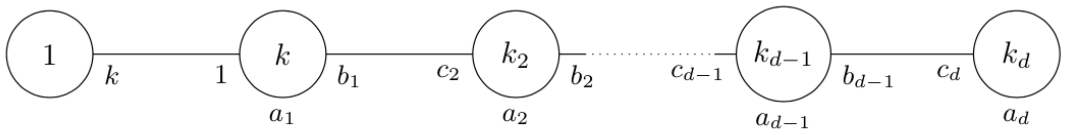


Figure 1.2: The distance distribution diagram of a distance regular graph

This diagram shows the view of a distance regular graph from an arbitrary vertex: Starting from a fixed vertex v, it has valency k, and the next k in the second circle denotes the number of vertices that are at distance one with v. For each vertex in the second circle, it has $c_1 = 1$ edge with v, a_1 edges with other vertices in the second circle, and b_i edges with vertices that are at distance two with v, which is in the third circle. The number of vertices in the third circle is k_2, and the corresponding a_2, b_2 ,and c_2 are labeled. For the last circle (the $(d+1)$-th circle), $b_d = 0$ is not shown on the diagram.

In the remainder of this section we prove a theorem on distance regular graphs which is useful in Chapter 2.

Theorem 1.4.8. Let G be a distance regular graph and $A_0, \ldots, A_d$ are its distance matrices. Then we have:

$$A_1 A_i = b_{i-1} A_{i-1} + a_i A_i + c_{i+1} A_{i+1}, \text{for } 1 \leq i \leq d-1.$$

Proof. Recall that $A_1 = A(G)$ is the adjaceny matrix of G. The $ij-$th entry of $A_1 A_i$ is the number of vertices v that satisfy $d(i,v) = 1$ and $d(j,v) = i$. All possible cases are $d(i,j) = i-1, i, i+1$, with the number of vertices are b_{i-1}, a_i, c_{i+1}, and the statement is proved. □

By applying Theorem 1.4.8 recursively, since $A_0 = I$ and A_1 are polynomials of A_1, of degree zero and one respectively. We can conclude that the distance matrices A_i are polynomials p_i of degree i.

CHAPTER 2

ASSOCIATION SCHEMES AND THE BOSE-MESNER ALGEBRA

The content of this chapter is based on [11].

2.1 Association schemes

Definition 2.1.1 (Association scheme)**.** An association scheme of order n and rank d (also called a d-association scheme) is a set $\mathcal{A} = \{A_0, \ldots, A_d\}$ of $0,1$-matrices of size $n \times n$ with the following properties:

(1) $A_0 = I$, where I is the identity matrix,

(2) $\sum_{i=0}^{d} A_i = J$, where J is the all-1 matrix,

(3) $A_i^T \in \mathcal{A}$, for all i,

(4) $A_i A_j = A_j A_i$,

(5) $A_i A_j \in \text{span}(\mathcal{A})$, for all $0 \leq i, j \leq d$.

The elements $A_1, \ldots, A_d$ in $\mathcal{A}$ are called *classes*. By condition (4), here we defined a symmetric association scheme. For condition (5), it means that $A_i A_j$ is a linear combination of $A_0, A_i, \ldots, A_d$.

Besides the matrix product, the following Hadamard product also plays an important role.

Definition 2.1.2. The *Hadamard product* of two matrices A, B, denoted as $A \circ B$, is defined to be:

$$(A \circ B)_{ij} := A_{ij} \cdot B_{ij}.$$

The Hadamard product is associative and commutative, with J as its identity element.

Example 2.1.3. The strongly regular graphs are association schemes with two classes. In this case we have that $A_0 = I$, A_1 is the adjacency matrix and A_2 is the adjacency matrix of the complementary graph.

Lemma 2.1.4. The matrices $A_0 = I$, A_1 are linearly independent.

Proof. Notice that all A_k's are $0,1$-matrices. For an fixed entry, say the (i,j)-th entry, there is exactly one matrix A_k such that $A_k(i,j) = 1$ and for all $l \neq k$, $A_l(i,j) = 0$. Therefore, A_k cannot be the sum of other matrices. □

By (5) we know that the algebra generated by $A_0, \ldots, A_d$ has dimension $d+1$. Besides, since all A_i commute with each other, we know $J = \sum_{i=0}^{d} A_i$ commutes with A_i. Therefore, for each i, the rows and columns of A_i has the same sum.

The condition (4) can also be expressed as: There exist constants p_{ij}^k such that $A_i A_j = \sum_{k=0}^{d} p_{ij}^k A_k$. The parameters p_{ij}^k are called the *intersection numbers* of the scheme.

Lemma 2.1.5. The intersection numbers are non-negative integers.

Proof. Let $A_i A_j = \sum_{k=0}^{d} p_{ij}^k A_k$. The (x,y)-th entry of $A_i A_j$ is the number of z such that $A_i(x,z) = A_j(z,y) = 1$, and this number is p_{ij}^w where w is the index such that $A_w(x,y) = 1$. Therefore, the intersection numbers are non-negative integers. □

The number of points of this scheme is $n = \sum_{i=0}^{d} p_{ii}^0$.

Now let us think about the relation between a distance regular graph and an association scheme.

Lemma 2.1.6. A distance regular graph G of diameter d can be viewed as a d-association scheme: the set $\mathcal{A}$ is formed by the distance matrices$A_0, \ldots, A_d$.

Proof. The distance matrices $A_0, \ldots, A_d$ clearly satisfy conditions (1), (2), and (3). According to Theorem 1.4.8, $A_i = p_i(A(G))$ are polynomials of $A(G)$, which again show that they commute with each other. Therefore they form a d-association scheme. □

2.2 The Bose-Mesner algebra

Recall the Definition 2.1.1 of a symmetric association scheme. From the condition (2), we know that the A_i's are linearly independent. From the condition (4) and (5), by [2], they generate a commutative algebra of dimension $d+1$. This algebra is called the *Bose-Mesner algebra* of an association scheme $\mathcal{A} = \{A_0, \ldots, A_d\}$. We denote this algebra as $\mathfrak{A}$. It was introduced and studies by Bose and Menser [5]. In this thesis, we view the Bose-Mesner algebra as being defined over $\mathbb{R}$.

Since $\mathcal{A} = \{A_0, \ldots, A_d\}$ is closed under the Hadamard product, the Bose-Mesner algebra is also closed under the Hadamard product. In other words, it is an algebra with respect to Hadamard product.

The matrices $A_0, \ldots, A_d$ form a basis for this algebra. In the remaining part of this section we will find another basis for $\mathfrak{A}$, which is a set of matrix idempotents.

Definition 2.2.1 (Idempotent and nilpotent)**.** An $n \times n$ matrix E is called an *idempotent* if $E^2 = E$. We say that two idempotents E_1 and E_2 are *orthogonal* if $E_1E_2 = 0$. An $n \times n$ matrix N is called a *nilpotent* if $N^k = 0$ for some k.

Example 2.2.2. For example, the identity matrix I and the all 0 matrix O are idempotents. The following matrix N is a nilpotent since $N^3 = 0$:

$$N = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

Lemma 2.2.3. Let $\mathfrak{B}$ be a commutative matrix algebra with identity over an algebraically closed field, then each matrix $A \in \mathfrak{B}$ is a linear combination of idempotents.

Proof. Let $\varphi(x)$ be the minimum polynomial of A:

$$\varphi(x) = \prod_{i=1}^{k} (x - \theta_i)^{m_i},$$

where θ_i is the i-th eigenvalue and m_i is its multiplicity. If we define

$$\varphi_i(x) := \frac{\varphi(x)}{(x - \theta_i)^{m_i}},$$

then $\varphi_1(x), \ldots, \varphi_k(x)$ are a set of co-prime polynomials. Hence there exist polynomials $f_1(x), \ldots, f_k(x)$ such that

$$\sum_{i=1}^{k} f_i(x)\varphi_i(x) = 1.$$

For $x = A$, we get

$$\sum_{i=1}^{k} f_i(A)\varphi_i(A) = I. \tag{2.1}$$

Notice that $\varphi_i(A)\varphi_j(A) = 0$ for $i \neq j$. Multiply with $f_i(A)\varphi_i(A)$ both sides of (2.1). Then we get

$$(f_i(A)\varphi_i(A))^2 = f_i(A)\varphi_i(A),$$

which means that $f_i(A)\varphi_i(A)$ is an idempotent. Let $E_i = f_i(A)\varphi_i(A)$, then $E_iE_j = 0$ for $i \neq j$. Moreover, $\varphi(x)|(x - \theta_i)^{m_i}\varphi_i(x)$, hence $(A - \theta_i I)^{m_i}E_i = 0$, which implies $[(A - \theta_i I)E_i]^{m_i} = 0$. Since B contains no non-zero nilpotent elements, $(A - \theta_i I)E_i = 0$. So we have that $AE_i{=}\theta_i E_i$.

Therefore, (2.1) can be rewrited as

$$E_1 + \cdots + E_k = I.$$

Then we have

$$A = AE_1 + \cdots + AE_k = \theta_1 E_1 + \cdots + \theta_k E_k,$$

which is the expression of A as the linear combination of idempotents. □

Now consider a relation between two idempotents, E_1 and E_2, in $\mathfrak{A}$. If $E_2E_1 = E_1 = E_1E_2$, we say that E_1 is contained in E_2, denoted as $E_1 \leq E_2$. We claim that this relation $\leq$ is a partial order of idempotents.

Lemma 2.2.4. The relation $\leq$ is a partial order of idempotents in $\mathfrak{A}$.

Proof. We need to show that $\leq$ is reflexive, anti-symmetric, and transitive. Let E_1, E_2, E_3 be distinct idempotents in $\mathfrak{A}$.
Reflexive: $E_1^2 = E_1$ hence $E_1 \leq E_1$.
Anti-symmetric: Suppose $E_1 \leq E_2$ and $E_2 \leq E_1$, then $E_1 = E_1E_2 = E_2$.
Transitive: Suppose $E_1 \leq E_2$ and $E_2 \leq E_3$, then $E_1E_3 = (E_1E_2)E_3 = E_1(E_2E_3) = E_1E_2 = E_1$, hence $E_1 \leq E_3$. □

We define the *minimal idempotent* to be a minimal element of a set of non-zero idempotents with respect to the partial order $<$. We claim that such minimal idempotents do exist. Let E_1, E_2 be two different idempotents with $E_1 \leq E_2$, then $E_2(I - E_1) \neq 0$ but $E_1(I - E_1) = 0$. It follows that the column space of E_1 is a proper subspace of the column space of E_2. Therefore, if we have a chain of idempotents $E_1 \leq \cdots \leq E_m$ then we must have $m \leq n + 1$. Hence the minimal idempotents exist.

The following theorem provides the existence of the basis we want to find.

Theorem 2.2.5. Let $\mathfrak{B}$ be a commutative matrix algebra with identity over an algebraically closed field, and without non-zero nilpotent elements. Then $\mathfrak{B}$ has a basis of pairwise orthogonal idempotents.

Proof. By Lemma 2.2.3, we only have to show that each idempotent is a sum of minimal idempotents, hence $\mathfrak{B}$ is spanned by those minimal idempotents. Let E be an idempotent, and E_0 be the sum of distinct minimal idempotents F_i such that $F_i \leq E$. We claim that

$E_0 = E$, otherwise $E - E_0$ is an idempotent and is contained in E. Therefore, there is a minimal idempotent F contained in $E - E_0$, which is also contained in E, but not in the sum E_0, a contradiction. □

Remark. Notice that a non-zero linear combination of orthogonal idempotents is not a nilpotent, hence this condition is also necessary. We call a commutative algebra *semi-simple* if the only nilpotent of it is 0.

Before we apply Theorem 2.2.5 to Bose-Mesner algebras, we first recall some statements of linear algebra:

Lemma 2.2.6. For a real matrix A, $\mathrm{tr}(A^T A) = 0$ if and only if $A = 0$.

Proof. It is clear that if $A = 0$ then $A^T A = 0$ hence $\mathrm{tr}(A^T A) = 0$.

Assume that $\mathrm{tr}(A^T A) = 0$. Notice that:

$$(A^T A)(i, j) = \sum_k A(k, i)A(k, j),$$

we have:

$$0 = \mathrm{tr}(A^T A) = \sum_i \sum_k A(k, i)^2.$$

The right hand side of the above equation is 0 only if $A(k, i)$ for all k, i. □

Now we are able to apply Theorem 2.2.5 to Bose-Mesner algebras.

Theorem 2.2.7. Let $\mathfrak{A}$ be the Bose-Mesner algebra of an association scheme, then $\mathfrak{A}$ has a basis of matrix idempotents $E_0, \ldots, E_d$, satisfying:

(1) $E_i E_j = \delta_{ij} E_i$.

(2) The columns of E_i are eignevectors for each matrix in $\mathfrak{A}$.

(3) $\sum_{i=0}^{d} E_i = I$.

(4) $E_i^T = E_i$.

Proof. For (1), let $N \in \mathfrak{A}$, $N^2 = 0$ be a nilpotent. Then by Lemma 2.2.6 we have

$$\mathrm{tr}((N^T N)^T (N^T N)) = \mathrm{tr}((N^T N))^2 = 0. \tag{2.2}$$

By (2.2) we can see $N^T N = 0$ then $\mathrm{tr}(N^T N) = 0$ hence $N = 0$. This shows that $\mathfrak{A}$ satisfies the condition in Theorem 2.2.5, hence $\mathfrak{A}$ has a basis of orthogonal idempotents $E_0, \ldots, E_d$.

For (2), let $A \in \mathfrak{A}$. Then there are $a_i \in \mathbb{C}$ such that $A = \sum_{i=0}^{d} a_i E_i$. Since E_i's are orthogonal to each other, we have $AE_i = a_i E_i$, for any $0 \leq i \leq d$. This equations shows that the column vectors of E_i are eigenvectors of A, with respect to the eigenvalue a_i.

For (3), let $A = I$ in the above argument. Then a_i's are the eigenvalues of I, hence $a_i = 1$, and we have $\sum_{i=0}^{d} E_i = I$.

For (4), since $E_i^T \in \mathfrak{A}$, there are $a_i \in \mathbb{C}$ such that $E_i^T = \sum_{i=0}^{d} a_i E_i$, hence $E_i^T E_k = a_k E_k$, for all$0 \leq k \leq d$. For the case $k = i$, since $\mathrm{tr}(E_i^T E_i) > 0$, $\mathrm{tr}(E_i) > 0$, we know $a_i \neq 0$. But E_i^T is a minimal idempotent, hence for the case $k \neq 0$, $a_k = 0$. Therefore E_i^T is a scalar multiple of E_i, so we must have $E_i^T = E_i$.

□

Since A_i commute, they can be diagonalized at the same time. Thus we find a decomposition of the space $\mathbb{R}^n$ as a direct sum of $d+1$ eigenspaces. Let $\{E_0, \ldots, E_d\}$ be the basis for $\mathcal{A}$, then the dimension of each eigenspaces are $f_i = \mathrm{rank}E_i = \mathrm{tr}E_i$. Since both $\{E_0, \ldots, E_d\}$ and $\{A_0, \ldots, A_d\}$ are bases of $\mathcal{A}$, there exists transition matrices P and Q such that:

$$A_i = \sum_{j=0}^{d} P(j,i) E_j \text{ and } E_i = \frac{1}{n} \sum_{j=0}^{d} Q(j,i) A_j.$$

We call P and Q the *eigenmatrices* of the association scheme.

Lemma 2.2.8. The matrices P and Q has the following properties:

(1) $P(i,0) = Q(i,0) = 1$,

(2) $P(0,j) = p^0_{jj}$, $Q(0,j) = f_j$,

(3) $P(i,j)P(i,k) = \sum_l p^l_{jk} P(i,l)$,

(4) $f_i P(i,j) = p^0_{ii} Q(j,i)$, $\sum_i f_i P(i,j)P(i,k) = \delta_{jk} p^0_{jj} n$, $\sum_i p^0_{ii} Q(i,j)Q(i,k) = \delta_{jk} f_j n$,

(5) $P(i,j)Q(k,i) = \sum_l p^k_{jk} Q(l,i)$,

(6) $\sum_i P(i,k) = \sum_i p^i_{ik}$.

Proof. Noticing that $A_0 = I$ and $E_0 = \frac{1}{n}J$, we get (1). Since $A_i J = p^0_{ii} J$ and $\mathrm{tr}E_j = f_j$, we obtain (2). Notice that $A_j A_k = \sum_l p^l_{jk} A_l$, express A_i in terms of E_i will get (3).

Notice that $\sum_i f_i P(i,j)P(i,k) = \mathrm{tr}(A_j A_k) = \delta_{jk} n p^0_{jj}$, which is the second part of (4). Since $PQ = nI$, the other two parts are equivalent to the second part.

By (3) and the first part of (4), we get (5).

For (6), since we have:

$$n\sum_i P(i,k) = \sum_{i,j} Q(j,i)P(i,j)P(i,k) = \sum_{i,j,l} p^l_{jk} Q(j,i)P(i,l) = n\sum_i p^j_{jk},$$

both sides divided by n will get (6). □

2.3 The Krein parameters

We have shown that the Bose-Mesner algebra $\mathcal{A}$ is closed under the Hadamard product, and A_i form a basis of minimal idempotents corresponding to Hadamard product. Hence there exist constants q^k_{ij} such that:

$$E_i \circ E_j = \frac{1}{n}\sum_{k=0}^{d} q^k_{ij} E_k. \tag{2.3}$$

These constants q^k_{ij} are called the *Krein parameters*.

Lemma 2.3.1. The Krein parameters q^k_{ij} of an association scheme satisfy:

(1) $q_{0j}^k = \delta_{jk}$,

(2) $q_{ij}^0 = \delta_{ij} f_j$,

(3) $q_{ij}^k = q_{ji}^k$,

(4) $\sum_l q_{ij}^l q_{lk}^m = \sum_l q_{il}^m q_{jk}^l$,

(5) $q_{ij}^k f_k = q_{ik}^j f_j$.

Proof. Let $\sum(A) = \sum_{i,j} A_{ij}$ be the sum of all entries of the matrix A, and J be the matrix with all components is 1. Then we have:

$$\begin{aligned}
\mathrm{tr}(MN^T) &= \sum_{i=1}^n (MN^T)(i,i) \\
&= \sum_{i=1}^n \left(\sum_{j=1}^n M(i,j) N^T(j,i) \right) \\
&= \sum_{i=1}^n \sum_{j=1}^n M(i,j) N(i,j) \\
&= \sum (M \circ N),
\end{aligned}$$

and

$$JAJ = \sum(A) J,$$

and

$$\begin{aligned}
\sum(E_i) J &= J E_i J \\
&= n E_0 E_i J \\
&= \delta_{0i} n J \qquad \text{(by Theroem 2.2.7(1))},
\end{aligned}$$

Since $E_0 = n^{-1} J$, (1) holds.

Noticing that $q_{ij}^0 = \sum(E_i \circ E_j) = \mathrm{tr}(E_i E_j) = \delta_{0i} n J$, hence (2) holds.

The statement (3) is obvious since $E_i \circ E_j = E_j \circ E_i$.

For (4), let us compute $E_i \circ E_j \circ E_k$ in two ways: If we compute as $(E_i \circ E_j) \circ E_k$ then the coefficient of E_m is $\sum_l q_{ij}^l q_{lk}^m$; if we compute it as $E_i \circ (E_j \circ E_k)$ then the coefficient of

E_m is $\sum_l q_{il}^m q_{jk}^l$.

Letting $m = 0$ in (4), we get (5). □

The constants q_{ij}^k may not be integers but they are always non-negative, which is known as the *Krein conditions*.

2.4 P- and Q-polynomial association schemes

Recall that in Section 1.4 we consider a graph G and the matrices A_i which carry the distance relation. In this case, $p_{ij}^k = 0$ if one of i, j, k is greater than the sum of the other two, and $p_{ij}^k \neq 0$ for $i = j + k$. Such schemes are called *metric* and are the same objects as distance regular graphs.

Similarly, a *cometric* scheme is a scheme which $q_{ij}^k = 0$ if $i > j + k$, and $q_{ij}^k \geq 0$ for $i = j + k$.

Definition 2.4.1. An association scheme is called *P-polynomial* if there exist polynomials ϕ_k, $\deg(\phi_k) = k$, with real coefficients, such that there are real numbers x_i satisfying $P(i, k) = \phi_k(x_i)$.

Similarly, an association scheme is called *Q-polynomial* if such polynomials satisfy $Q(i, k) = \phi_k(x_i)$ for some ordering of E_i.

By Lemma 2.2.5(3) we have:

$$\sum_i f_i \phi_j(x_i)\phi_k(x_i) = \sum_i f_i P(i, j)P(i, k) = \delta_{jk} p_{jj}^0 n,$$

which leads to the conclusion that the polynomials ϕ_k form a set of orthogonal polynomials.

We have the following eqivalence between these two properties:

Theorem 2.4.2. An association scheme is metric if and only if it is P-polynomial, is cometric if and only if it is Q-polynomial.

Proof. Suppose $\mathcal{A}$ is metric, we have

$$A_1 A_i = p_{1i}^{i-1} A_{i-1} + p_{1i}^{i} A_i + p_{1i}^{i+1} A_{i+1}.$$

Notice that $p_{1i}^{i+1} \neq 0$, which means A_{i+1} is a linear combination of A_1, A_{i-1} and A_i, hence for all i there exists a polynomial ϕ_i of degree i such that $A_i = \phi_i(A_1)$. Furthermore, we have

$$P(i,j)E_i = A_j E_i = \phi_j(A_1)E_i.$$

Therefore, $P(i,j) = \phi_j(P(i,1))$.

Now suppose that $\mathcal{A}$ is P-polynomial, then we have orthogonal polynomials ϕ_i such that the following relation holds [21]:

$$a_{i+1}\phi_{i+1}(x) = (b_1 - x)\phi_i(x) + c_{1-1}\phi_{i-1}(x).$$

Hence we have

$$P(i,1)P(i,j) = -a_{j+1}P(i,j+1) + b_j P(i,j) + c_{j-1}P(i,j-1) \quad \forall 0 \leq i \leq d.$$

Notice that $P(i,1)P(i,j) = \sum_k p_{ij}^k P(ik)$, we must have $p_{1j^k} = 0$ for $|k-j| > 1$. Inductively we are able to show $\mathcal{A}$ is metric.

The proof for the cometric case uses a similar argument. □

P-polynomial association schemes are related with distance regular graphs by the following theorem:

Theorem 2.4.3. If $\mathcal{A}$ is a P-polynomial association scheme, then its Hadamard idempotents are distance matrices of a distance regular graph.

Proof. Suppose that the corresponding graphs of $\mathcal{A}$ are $G_1, \ldots, G_d$, and $A_1, \ldots, A_d$ are their adjacency matrices. Let ϕ_k be polynomials with degree k such that $A_i = \phi_i(A_1)$, and

$\mathcal{P}$ be the vector space spanned by all ϕ_k's. We define a mapping f over $\mathcal{P}$ as following:

$$f(\phi, \psi) = \mathrm{tr}(\phi(A_1)\psi(A_1)).$$

This is an inner product over $\mathcal{P}$, and the polynomials ϕ_k's are orthogonal to each other under the inner product f. Since $x\phi_k = \sum_{i=0}^{k+1} \alpha_i \phi_i$ for some coefficients α_i and if $i \leq k-2$ then $f(x\phi_k, \phi_n) = 0$, hence $x\phi_k$ is a linear combination of ϕ_{k-1}, ϕ_k and ϕ_{k+1}, for all k. Hence G_1 is a distance regular graph, with distance matrices $A_1, \ldots, A_d$. □

A P-polynomial association scheme, as the discussion above, can be treated as a combinatorial realisation of some pairwise orthogonal polynomials. An important result has been shown by Leonard [16]: the orthogonal polynomials corresponding to the association schemes which are both P- and Q-polynomial are either Askey-Wilson polynomials, or some limiting cases.

2.5 The Euclidean representation of a graph

In this section, we introduce the Euclidean representation of graphs, specifically of distance regular graphs, together with the theory developed based on it. This section is mostly based on the book of Brouwer, Cohen, and Neumaier [7].

A representation of a distance regular graph is a mapping of its vertices into the eigenspaces of the corresponding association scheme. We will show that this mapping has the property that the angles between the images are determined by the distance of their corresponding vertices.

2.5.1 Representation of a graph

Let V be a vector space, and $V = U \oplus W$ a decomposition of V. For every $v \in V$, there is a unique expression $v = u_v + w_v$ where $u_v \in U$ and $w_v \in W$. Let $E : V \to U$ defined by $v \mapsto u_v$ be a mapping. This mapping E is called a *projection*.

We first prove the following theorem on the relation between projections and idempotents.

Theorem 2.5.1. (1) Every projection is an idempotent.

(2) Every idempotent from $\mathrm{End}(V, V)$ is a projection.

Proof. To prove (1), let E be a projection from V to its subspace U, such that $E(v) = u$ and let $u^{\perp}$ be the remaining such that $v = u + u^{\perp}$. Then $E(E(v)) = E(E(u + w)) = E(u) = E(u + 0) = u = E(v)$, for all $v \in V$. Hence $E = E^2$ is an idempotent.

To prove (2), let $E \in \mathrm{End}(V, V)$, $E = E^2$ which is an idempotent. Let $U = \mathrm{Im}(E)$ and $W = \mathrm{Ker}(E)$. We claim that $V = U \oplus W$: For any $v \in V$, let $u = E(V)$ and $w = v - E(v)$. Then $u \in U$, and $E(w) = E(v - E(v)) = E(v) - E(E(v)) = E(v) - E(v) = 0$. Furthermore, let $v \in U \cap W$. Since $v \in U$, there exists some $x \in V$ such that $E(x) = v$. Hence $0 = E(v) = E(E(x)) = E(x) = v$. We conclude that $V = U + W$ is a direct sum, in other words, $V = U \oplus W$. □

Let U_E and W_E be the two subspaces corresponding to the projection E. Notice that $E_1 \leq E_2$ is equivalent to $U_{E_1} \subset U_{E_2}$ and $W_{E_2} \subset W_{E_1}$. Therefore, we have the following corollary:

Corollary 2.5.2. If $E_1 \leq E_2$ and they have the same rank then $E_1 = E_2$.

Now let G be a graph with vertex set $\{v_1, v_2, \ldots, v_n\}$ and adjacency matrix A. Let θ be an eigenvalue of A with multiplicity m, with the corresponding eigenspace S_θ. Let $e_1, e_2 \ldots, e_n$ be vectors in $\mathbb{R}^n$ which correspond to $v_1, v_2, \ldots, v_n$. We call the space spanned by $e_1, e_2 \ldots, e_n$ as V_G. In other words, this space V_G is spanned by the vertices of the graph G. Now we can give the definition of the Euclidean representation of a graph:

Definition 2.5.3. The projection $E_\theta : V_G \to S_\theta$ is a *Euclidean representation* of G if $\sum_{j \sim i} E_\theta(e_j) = \theta E_\theta(e_i)$.

By Theorem 2.5.1, these mappings are idempotents.

If we consider the association scheme of G, then by Theorem 2.2.5, it has a basis of orthogonal idempotents. In fact, the projections E_θ are exactly the basis E_i we mentioned above. Firstly, for all eigenvalues θ, these projections are orthogonal since the images are in the different eigenspaces, which are orthogonal to each other by Theorem 1.2.8. Secondly, all the eigenspaces form a decomposition of $\mathbb{R}^n$, so these projections are a basis for the space. Since there is not a natural order of E_θ's, in the remaining of this thesis, we will use E_θ with different θ as the notation of these idempotents.

Example 2.5.4. Consider the cube graph, it has 8 vertices and 12 edges, with the spectrum $\{-3^1, -1^3, 1^3, 3^1\}$. The following figure demonstrates the Euclidean representation with respect to the eigenvalue $\theta = 1$. Notice that for each vertex, its image is the sum of the images of its neighbours.

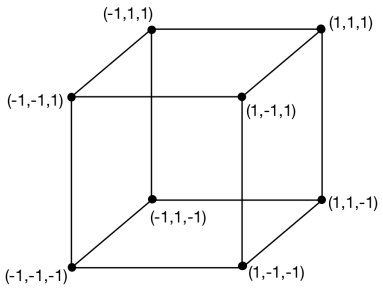


Figure 2.1: A representation of the cube graph

Theorem 2.5.5. For any two vertices i, j of G, and for any non-negative integer r, we have

$$(A^r)_{ij} = \sum_{\theta \in S_\theta} \langle E_\theta(e_i), E_\theta(e_j) \rangle \theta^r \tag{2.4}$$

In order to prove this, we prove a general theorem on spectral decomposition.

Theorem 2.5.6. Let A be a real symmetric matrix with the set of all eigenvalues spec(A). An eigenvalue $\theta \in$ spec(A) has multiplicity $m(\theta)$ and corresponding eigenspace S_θ. Let U_θ be the matrix which columns are a basis of the eigenspace of θ. Then for any non-negative

integer r, we have:

$$A^r = \sum_{\theta \in \mathrm{ev}(G)} \theta^r U_\theta U_\theta^T \tag{2.5}$$

Proof. Consider the sum $U = \sum_{\theta \in \mathrm{ev}(G)} U_\theta U_\theta^T$. Since θ run through the eigenvalues of A, we have $U^2 = U$, which means all eigenvalues of U should be either 0 or 1. Besides, we have

$$\mathrm{tr}(U) = \sum_\theta \mathrm{tr}(U_\theta U_\theta^T) = \sum_\theta m(\theta),$$

which draw to the conclusion that all eigenvalues of U are 1, hence it is the identity matrix. Multiply A^r on both sides of $I = \sum U_\theta U_\theta^T$, and notice the fact that $A^r U = \sum U_\theta U_\theta^T = \theta^r U = \sum U_\theta U_\theta^T$, the theorem is proved. □

It is clear that the $ij-$entry of $U_\theta U_\theta^T$ is exactly $\langle E_\theta(e_i), E_\theta(e_j)\rangle$, hence Theorem 2.5.5 follows.

Now we consider the inner product of two images $\langle E_\theta(e_i), E_\theta(e_j)\rangle$. We have the following lemma:

Lemma 2.5.7. Let v_i, v_j be two vertices of G. Then $\langle E_\theta(e_i), E_\theta(e_j)\rangle$ is determined by $d(v_i, v_j)$ only.

In order to prove this lemma, recall the kth distance matrix $A_k(G) = A_k$ of G are $n \times n$ matrices. In distance regular graphs, $A_k A_h$ is a linear combination of $A_1, A_2, \ldots, A_d$.

Proof. By the previous argument, $A_k A_h$ is a linear combination of distance matrices $A_1, A_2, \ldots, A_d$, hence so is $A^2 = A_1 A_1$, and by induction it is easy to see that there exist constants $c(r, t)$:

$$A^t = \sum_{r=0}^{d} c(r, t) A_r.$$

Let i, j be two vertices of distance d in G, then the ij-entry of A^t is $c(d, t)$. By Theorem 2.5.5, we draw the conclusion that:

$$c(d, t) = \sum_{\theta \ \in \mathrm{ev}(G)} \langle E_\theta(e_i), E_\theta(e_j)\rangle \theta^t.$$

Fixing i, j, and letting t run through $0, 1, \ldots, d$, we will get a linear system satisfied by inner products $\langle E_\theta(e_i), E_\theta(e_j)\rangle$. Notice that coefficient matrix of this system is a Vandermonde matrix, which means this matrix is non-singular and hence $\langle E_\theta(e_i), E_\theta(e_j)\rangle$ are determined by the values of θ and $c(r,t)$. The latter are only determined by $d(i,j)$ and the statement is proved. □

2.5.2 The cosine sequence

Now let E_θ be a representation with respect to θ such that all image vectors $E_\theta(e_i)$ are scaled to be unit vectors. By Lemma 2.5.7, there are real numbers $w_i, 0 \leq i \leq d$ such that for all $x, y \in V(G), d(x,y) = i$, we have:

$$\langle E_\theta(e_x), E_\theta(e_y)\rangle = w_i. \tag{2.6}$$

We call this sequence $w_0, w_1, \ldots, w_d$ *the cosine sequence* of graph G with respect to the eigenvalue θ. It is clear that $w_0 = 1$ and $|w_i| \leq 1$.

Now fix two vertices $x, y \in V(G)$ with $d(x,y) = i$, among the neighbours of y, there are c_i vertices z_j has distance $i-1$ with x, and a_i vertices u_j has distance i with x, and b_i vertices v_j has distance $i+1$ with x. We have:

$$\theta E_\theta(e_v) = \sum E_\theta(e_{z_j}) + \sum E_\theta(e_{u_j}) + \sum E_\theta(e_{v_j}).$$

By taking the inner product with $E_\theta(u)$ on both sides of the above equation, we have:

$$\theta w_i = c_i w_{i-1} + a_i w_i + b_i w_{i+1}, \quad 0 \leq i \leq d, \tag{2.7}$$

where we define $w_{-1} = w_{d+1} = 0$. The cosine sequence is uniquely determined by θ.

From (2.7) we have:

$$w_1 = \frac{\theta}{k}, w_2 = \frac{\theta^2 - a_1\theta - k}{kb_1},$$

In the case of G is a $\mathrm{srg}(v, k, \lambda, \mu)$, $a_1 = \lambda$ and $b_1 = \mu$, hence

$$w_1 = \frac{\theta}{k}, w_2 = \frac{\theta^2 - \lambda\theta - k}{k\mu}. \tag{2.8}$$

2.5.3 Injectivity

In this section we aim to determine when a representation of a distance regular graph is not injective.

Lemma 2.5.8. Let G be a distance regular graph of valency k, diameter d, and θ be one of its eigenvalues. Then E_θ is not injective if and only if one of the following holds:

(1) $\theta = k$,

(2) $\theta = -k$ and G is bipartite,

(3) the number of eigenvalues greater than θ is even, and G is antipodal.

Here by *antipodal* we mean the vertices at distance d from a given vertex are all at distance d from each other.

Proof. For the first case we have $w_1(\theta) = 1$ hence u_θ is constant.

For the second case, $w_1(\theta) = -1$. For any two adjacent vertices i, j, we have $E_\theta(i) + E_\theta(j) = 0$, which means E_θ is constant on the neighbourhood of any vertex. Therefore there are no cycles of odd length in G, hence G is bipartite.

Now we assume that G is antipodal. Notice that 'being at distance d or zero' induces an equivalence relation on the vertices of G, and the equivalence classes are called antipodal classes. The eigenspace of A_1 with respect to θ is also the eigenspace of A_d, with some eigenvalue λ. If we denote the set of vertices that are at distance d of i as $D(i, d)$, then we have $\sum_{j \in D(i,d)} E_\theta(j) = \lambda E_\theta(i)$, hence $w_d = \dfrac{\lambda}{k_d - 1}$. G_d now consists of disjoint complete graphs of k_d vertices, therefore its eigenvalues are $k_d - 1$ and -1, and $w_d = 1$ or $w_d = \dfrac{-1}{k_d - 1}$. If θ is the $r-$th largest eigenvalue then $(-1)^{r+1} w_d(\theta)$ is non-negative, therefore $w_d = 1$ if and only if the number of eigenvalues greater than θ is even.

Now we prove that these three conditions are necessary. Let G be a graph of valency k and diameter k, and $E_\theta(i) = E_\theta(j)$ for two vertices i, j with $d(i,j) = \delta$. Then $w_r = 1$ hence $E_\theta(i) = E_\theta(j)$ for all vertices i, j with $d(i,j) = \delta$. In other words, E_θ is constant on the components of G_δ. If G_δ is connected then $\theta = k$, otherwise either G_2 or G_d is not connected.

Suppose G_2 is not connected. In this case G is bipartite. Since $\theta E_\theta(i) = \sum_{j \sim i} E_\theta(j)$, we have $\theta E_\theta(i) = k E_\theta(j)$ if $i \sim j$. Dually $\theta E_\theta(j) = k E_\theta(i)$, hence $\theta^2 = k^2$.

Suppose G_d is not connected. In this case G is antipodal. By similar argument we can conclude that the number of eigenvalues greater than θ is even.

□

We call u_θ *locally injective* if it takes different values on the neighbourhood of any vertex. Notice that the union of disjoint complete graphs of same size is a distance regular graph, we have the following corollary:

Corollary 2.5.9. Let G be a distance regular graph with valency k, and not a union of disjoint complete graphs of same size. Let $\theta \neq \pm k$ be an eigenvalue of G, then u_θ is locally injective.

CHAPTER 3

INITIAL ANALYSIS OF SRG$(85, 14, 3, 2)$

We intend to prove Conjecture 0.0.1 by a complete enumeration. Suppose that there exist an srg$(85, 14, 3, 2)$, and we call such a graph Γ. In this chapter, we will have a look at the algebraic and combinatorial properties of Γ.

3.1 The Euclidean representation of Γ

We first compute the spectrum of Γ.

Theorem 3.1.1. The spectrum of Γ is $\{14^1, 4^{34}, -3^{50}\}$.

Proof. By Theorem 1.3.6, we have:

$$r = \frac{1}{2}\left(3 - 2 + \sqrt{(3-2)^2 + 4(14-2)}\right) = 4,$$

$$s = \frac{1}{2}\left(3 - 21\sqrt{(3-2)^2 + 4(14-2)}\right) = -3,$$

$$f = \frac{1}{2}\left(85 - 1 - \frac{2 \times 14 + (85-1)(3-2)}{\sqrt{(3-2)^2 + 4(14-2)}}\right) = 34,$$

$$g = \frac{1}{2}\left(85 - 1 + \frac{2 \times 14 + (85-1)(3-2)}{\sqrt{(3-2)^2 + 4(14-2)}}\right) = 50.$$

□

Recall that in Theorem 1.2.8 we have proved that eigenspaces with respect to distinct eigenvalues of a real symmetric matrix are orthogonal with respect to the dot product. Besides, as discussed in Section 2.5.1, we will use V_Γ to denote the space spanned by the vertices of Γ.

Now consider the eigenspace U_θ with respect to the eigenvalue $\theta = 4$. It is a subspace of $V \cong \mathbb{R}^{85}$ of dimension 34, and the projection $E_\theta : V_\Gamma \to U_\theta$ is a Euclidean representation of Γ. By (2.8), the cosine sequence of this representation consists of $w_1 = \dfrac{\theta}{k} = \dfrac{2}{7}$ and $w_2 = \dfrac{\theta^2 - \lambda\theta - k}{k\mu} = -\dfrac{1}{14}$. Hence for two distinct vertices $v_1, v_2 \in V(G)$, we have:

$$\langle E_\theta(v_1), E_\theta(v_2)\rangle = \begin{cases} \dfrac{2}{7}, & \text{if } v_1 \sim v_2; \\ -\dfrac{1}{14}, & \text{if } v_1 \not\sim v_2. \end{cases} \tag{3.1}$$

Now we introduce the Gram matrix and its important property that is useful in the following analysis.

Let $B = A^T A$ be a real $n \times n$ matrix where A is a real $m \times n$ matrix, consider the quadratic form $q(x) = x^T B x = (Ax)^T Ax$. Noticing that $q(x)$ is semi-positive definite, hence B is a semi-positive definite matrix. Let $a_1, a_2, \ldots, a_m$ be the column vectors of A, then B has the form:

$$\begin{bmatrix} \langle a_1, a_1\rangle & \cdots & \langle a_1, a_n\rangle \\ \vdots & \ddots & \vdots \\ \langle a_n, a_1\rangle & \cdots & \langle a_n, a_n\rangle \end{bmatrix}$$

We call any matrix of this form a *Gram matrix*. By the above argument, any Gram matrix obtained in this way is semi-positive definite.

For a graph G, given its Euclidean representation E_θ, the matrix $\mathrm{Gr}(i,j) = \langle E_\theta(v_i), E_\theta(v_j)\rangle$ is a Gram matrix.

By equation (3.1) and the fact that all the vectors $E_\theta(v_i)$, $1 \le i \le n$, are unit normal vectors, we know the fact that all entries in $\mathrm{Gr}(i,j)$ that are not on the diagonal are either $\dfrac{2}{7}$ or $-\dfrac{1}{14}$, depending on whether the vertices i and j are adjacent or not. The diagonal

entries are all 1.

3.2 The local graph

In this section we will focus on the subgraph induced by the 14 neighbours of a vertex v in Γ.

Lemma 3.2.1. Let $N(v)$ be the set of vertices that are adjacent to a vertex v. The subgraph N induced by $N(v)$ is a cubic graph of order 14. What's more, any two non-adjacent vertices $x, y \in N(v)$ have at most one common neighbour in N, and any two adjacent vertices $x, y \in N(v)$ have at most two common neighbours in N.

Proof. For a vertex v in Γ, $|N(v)| = 14$. Any vertex $x \in N(v)$ is adjacent to v so x has three common neighbours with v, in other words, x has valency three in N.

For any two non-adjacent vertices $x, y \in N(v)$, they have one more common neighbour other than v, hence they have at most one common neighbour in N.

Similarly, for any two adjacent vertices $x, y \in N(v)$, they have two more common neighbours other than v, hence they have at most two common neighbours in N. □

By the above property, we are able to analyze the structure of the local graph N, and first get the following conclusion.

Proposition 3.2.2. The subgraph N induced by the neighbourhood of a vertex v is either a connected cubic graph of order 14, or a union of a connected cubic graph of order ten and a complete graph K_4.

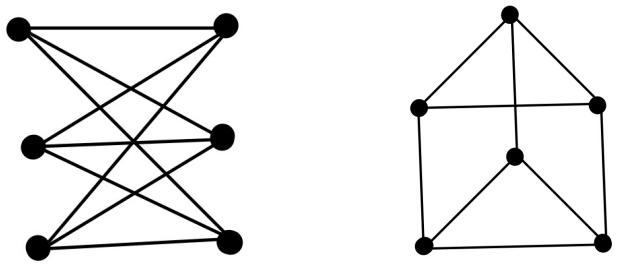
Proof. In a cubic graph K, $3|V(K)| = 2|E(K)|$, hence $|V(K)|$ must be even, and obviously it should be at least four. Furthermore, if $|V(K)| = 4$ then K is isomorphic to the complete graph K_4. Therefore, if a cubic graph of order 14 is not connected, then it must be in one of the following cases:

(1) two connected components: K_4 and one of order 10;

(2) two connected components: one of order 6 and one of order 8;

(3) three connected components: two K_4's and one of order 6.

We claim that any order 6 cubic graph cannot be a component of H. By the table of simple cubic graphs [25], there are only two cubic graphs on six vertices: $K_{3,3}$ and the prism graph Y_3 (also known as the box product of K_2 and K_3), as shown:



(a) $K_{3,3}$ (b) Y_3

Figure 3.1: $K_{3,3}$ and the prism graph Y_3

However, in both graphs there are non-adjacent vertices that have more than one common neighbour, which contradicts Lemma 3.2.1. Hence the only possible case is (1). □

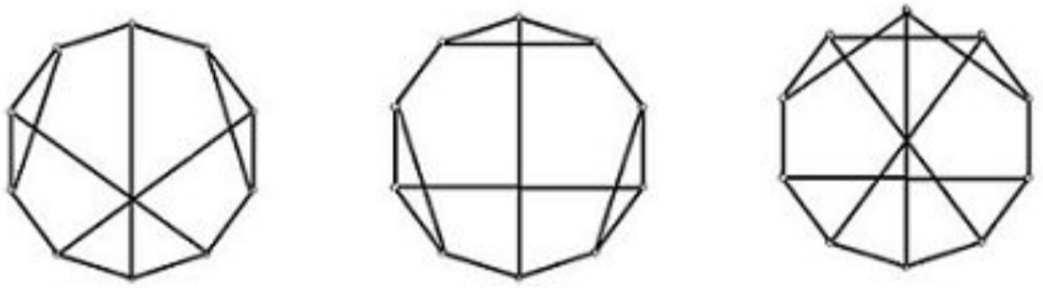
The connected cubic graphs of order up to 14 were enumerated in [8]. It follows from the table there that there are 509 connect cubic graphs of order 14 and 19 connected cubic graphs of order 10. Hence according to Lemma 3.2.1, the local graph H is one of 528 cubic graphs.

Proposition 3.2.3. Among the 528 cubic graphs above, only 39 satisfy the property that any two non-adjacent vertices have at most one common neighbour.

Definition 3.2.4. The 39 graphs mentioned in Proposition 3.2.3 are called *good graphs*.

Proposition 3.2.3 was verified in GAP [10] by going through the list of all known cubic graphs of order 14. The codes of these cubic graphs can be found in the database [18]. The resulting 39 possible local graphs H have been saved in the file `goodgraphs.g`.

Out of these possible 39 cubic graphs, 36 are connected and three are the sum of K_4 and one of the graphs T_1, T_2 or T_3 below:



(a) T_1 (b) T_2 (c) T_3

Figure 3.2: The three 10-order possible cubic graphs

Notice that T_3 is isomorphic to the Petersen graph.

Among the 39 good graphs, the last three are the disconnected ones.

If the local graph H is disconnected, with components $K \cong K_4$ and T_i, then v together with the four vertices with K form a 5-clique. In other words, we have the following statement:

Proposition 3.2.5. The local graph N at v is disconnected if and only if v is contained in a 5-clique in Γ.

3.3 Useful facts

In this section, we prove several statements about the representation of Γ onto the eigenspace U_4 satisfying the equation (3.1):

Lemma 3.3.1. If $w \in N(v)$, then the projection of w onto the span $\{v\}$ is $\frac{2}{7}v$.

Proof. Indeed, $\text{proj}_v w = \frac{\langle w, v\rangle}{\langle v, v\rangle} v = \frac{2/7}{1} v = \frac{2}{7}v$, since $w \sim v$. □

Theorem 3.3.2. This representation $E_4 : V_\gamma \to U_4$ is a Euclidean representation. In other words, for a vertex $v \in V(\Gamma)$, the sum of all the neighbours of v is $4v$.

Proof. As $|N(v)| = 14$, we have:

$$t = \sum_{w \in N(v)} \left(w - \frac{2}{7} v \right) = \left(\sum_{w \in N(v)} w \right) - 4v.$$

Then:

$$\begin{aligned} \langle t, t \rangle &= \left\langle \sum_{w \in N(V)} w - 4v, \sum_{w \in N(V)} w - 4v \right\rangle \\ &= \left\langle \sum_{w \in N(V)} w, \sum_{w \in N(V)} w \right\rangle - \left\langle 4v, \sum_{w \in N(V)} w \right\rangle - \left\langle \sum_{w \in N(V)} w, 4v \right\rangle \\ &\quad + \langle 4v, 4v \rangle \end{aligned}$$

For the local graph N, it has 21$= \dfrac{14 \times 3}{2}$ edges, and 70 non-edges. We also know $\langle v, w \rangle = \dfrac{2}{7}$ by (3.1), and $\langle v, v \rangle = 1$. Hence we have:

$$\left\langle \sum_{w \in N(V)} w, \sum_{w \in N(V)} w \right\rangle = 21 \times 2 \times \frac{2}{7} + 70 \times 2 \times \left(-\frac{1}{14} \right) + 14 = 16.$$

Moreover:

$$\begin{aligned} \langle t, t \rangle &= 16 - 4 \times \frac{2}{7} \times 14 - 4 \times \frac{2}{7} \times 14 + 16 \\ &= 0, \end{aligned}$$

which leads to the conclusion that $t = \sum_{w \in N(V)} w - 4v = 0$, which is $\sum_{w \in N(V)} w = 4v$. □

Another useful fact about Γ is:

Theorem 3.3.3. Every edge $(a, b) \in E(\Gamma)$ that is not contained in a 5-clique is contained in a maximal 3-clique.

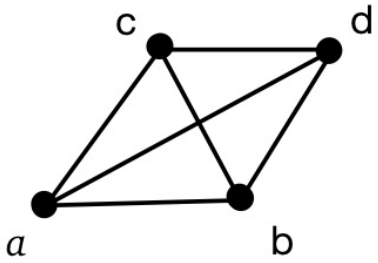
Proof. Let c be a common neighbour of a and b, then $\{a, b, c\}$ forms a 3-clique. Now

suppose that every such 3-clique is contained in a 4-clique, then there exists a vertex d that forms a 4-clique that contains the $\{a, b, c\} - 3-$clique.

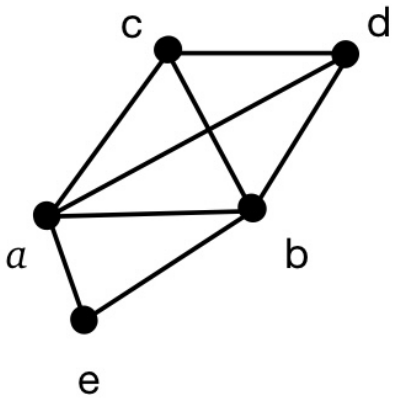
Since a and b are adjacent, they have three common neighbours, two of them are c and d, we call the other one vertex e.

Notice that $\{a, b, e\}$ also form a 3-clique, hence it must be contained in a 4-clique, as our assumption. However, a and b have only three common neighbours, this leads to a fact that the vertices d and e are adjacent to all of a, b, and c. Therefore d and e must be adjacent to each other, which leads to the conclusion that the vertices a, b, c, d, e form a 5-clique, a contradiction.

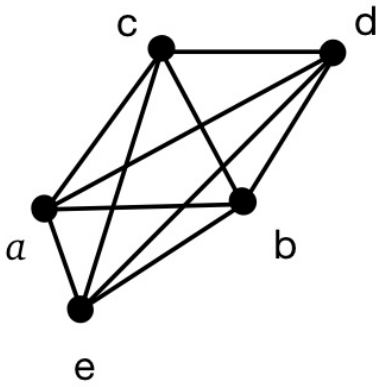
The figures following shows the process of our argument.



(a) Step 1: take a 4-clique



(b) Step 2: add the last common neighbour



(c) Step 3: claim that this must be a 5-clique

□

Corollary 3.3.4. Every vertex $a \in V(\Gamma)$ is contained in a maximal 3-clique.

Proof. Suppose not, then by Theorem 3.3.3, we can see that the neighbours of a must be divided into groups of four, in order to form 5-cliques. But $|N(a)| = 14$, which cannot be divided by 4. This contradiction means that a must be contained in a maximal 3-clique. □

Theorem 3.3.3 and Corollary 3.3.4 can also be justified directly by enumerating all 39 good graphs. But here we would like to give a combinatorial proof.

3.4 Segments

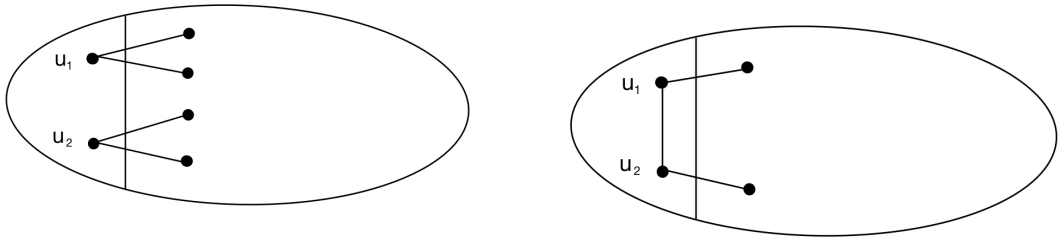
Now starting from a maximal 3-clique with vertices A, B, and C, the local graph of each of these vertices contains the other two, which appear as a pair of adjacent vertices. We will focus on the remaining twelve vertices in the neighbourhood.

Definition 3.4.1. Let S be a graph that is obtained from a good graph G by removing two adjacent vertices v_1 and v_2 which are not contained in a 3-clique of G. The sets H_1

and H_2 are the neighbourhoods of v_1 and v_2 in S, respectively. A *segment* is a triple consisting of the graph S and H_1, H_2, which are called *handles* of this segment.

Remark. Noticing that G is a cubic graph, v_1 have three neighbours in G. One of them is v_2, so H_1 has two vertices. Similarly, H_2 also has two vertices. Besides, since v_1 and v_2 are not contained in a 3-clique of G, we know that H_1 and H_2 are disjoint.

Lemma 3.4.2. Let S be a segment and $H = \{u_1, u_2\}$ be one of the handles. There are two cases: either there is no edges between u_1 and u_2, or there is an edge between u_1 and u_2. In the non-edge case, both u_1 and u_2 have two neighbours in $S \setminus H$. In the edge case, both u_1 and u_2 have one more neighbour in $S \setminus H$.



(d) non-edge handle (e) edge handle

Figure 3.3: The two cases in Lemma 3.4.2

Proof. The vertex u_1 has three neighbours in the original good graph, one of them is removed to get the segment, so there are two neighbours left. If u_1 and u_2 are not adjacent, then both of them have two neighbours in $S \setminus H$. Else they are neighbours of one another, so both of them have one more neighbour. □

Definition 3.4.3. In the two cases, we call H an *edge handle* or a *non-edge handle*, respectively. According to the types of handles, a segment can be one of the following three different types: edge and edge, edge and non-edge, non-edge and non-edge.

Lemma 3.4.4. Let S be a segment, $H_1 = \{u_1, u_2\}$ and $H_2 = \{w_1, w_2\}$ be the two handles. For any $i, j \in \{1, 2\}$, u_i and w_j are not adjacent.

Proof. Let G be the good graph that S is obtained from. Let (v_1, v_2) be the edge removed from G. Then H_1 is in the neighbourhood of v_1 and H_2 is in the neighbourhood of v_2. Consider the vertices v_1 and w_1, they are not adjacent in G. By Proposition 3.2.3, they have no more than one common neighbour in the good graph G, and they already have one, namely v_2. Therefore, as a neighbour of v_1, u_1 cannot be adjacent to w_1. Similar argument works for the other pairs of vertices. □

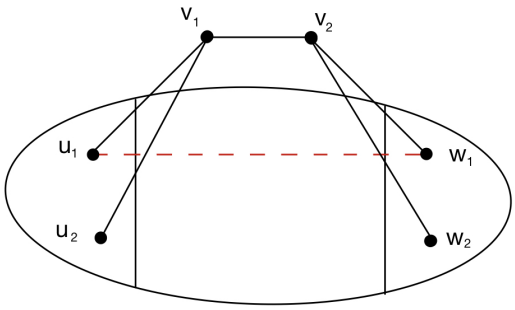


Figure 3.4: u_1 and w_1 are not adjacent, as an example

In the future enumeration, it is useful for us to treat two handles in a reasonable order. Before we do so, let us first define the isomorphism between segments.

Definition 3.4.5. Let S be a segment with handles H_1 and H_2, S' be another segment with handles H_1' and H_2'. A mapping φ is called a *segment isomorphism* if φ is an isomorphism from S to S' such that maps H_1 to H_1' and H_2 to H_2'.

Theorem 3.4.6. There are in total 478 segments up to isomorphism. Among them there are 19 segments with two edge handles, 2×78 segments with one edge handle and one non-edge handle, and 303 segments with two non-edge handles.

Here is how we find segments up to isomorphism: we go through the list of good graphs. For each good graph G, we compute the automorphism group $\mathrm{Aut}(G)$. Then we find the orbits of $\mathrm{Aut}(G)$ on the ordered edges (v_1, v_2) in G which are not contained in a 3-clique. These orbits correspond to segments up to isomorphism. Thus we take one ordered edge in the orbit and form a segment by removing that edge.

All the segments we need are recorded in the file `segment_data.g`. We keep 400 segments in total, excluding the segments with handles of type non-edge/edge. Furthermore, the segments in the file are ordered that first come with two edge handles, then with one edge and one non-edge handles, and finally with two non-edge handles.

Next we give the definition of cores in segments.

Definition 3.4.7. Let S be a segment, $H_1 = \{u_1, u_2\}$ and $H_2 = \{w_1, w_2\}$ be the handles of S. The *core* of S with respect to H_1 is the set of vertices in $V(S) \setminus (H_1 \cup H_2)$ that are not adjacent to u_1 or u_2. Symmetrically define the core with respect to H_2. Denote the cores as K_1, K_2, respectively.

Theorem 3.4.8. For an edge handle, the corresponding core has six vertices; for a non-edge handle, the the corresponding core has four vertices.

Proof. If u_1 and u_2 are non-adjacent then they cannot have any common neighbours in the segment. In fact, u_1 and u_2 are in the neighbourhood of a vertex in Γ, and they are both adjacent to v_1, a vertex removed from the goodgraph. Therefore, they already have two common neighbours and cannot have anymore in the segment. Hence they both have two distinct neighbours, and the remaining four vertices forms the corresponding core.

If u_1 and u_2 are adjacent, suppose that they have a common neighbour x in the segment. Consider the vertex v_1 we removed from the good graph G. The vertices v_1 and x are not adjacent in G, they should have at most one common neighbour. This contradicts with the fact that v_1 and x have two common neighbours u_1 and u_2 at this moment. Hence in this case, u_1 and u_2 also have different neighbours. □

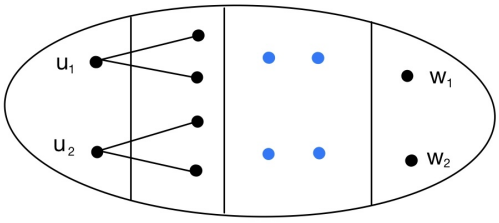


Figure 3.5: A non-edge handle have core of size 4

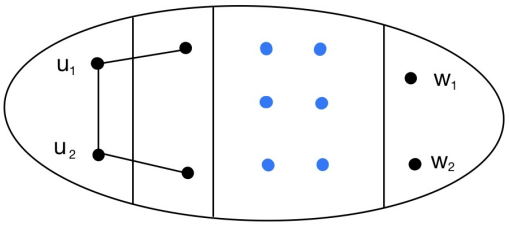


Figure 3.6: An edge handle have core of size 6

Remark. Now in order to make the type of handles look more intuitive, we call an edge handle as type 6 handle, and call a non-edge handle as type 4 handle.

Finally let us explain how vertices in the segment are ordered in the file `segments.g`. The list of the vertices starts with the two vertices from H_1 then the two vertices from H_2. The remaining eight vertices are split into four groups. First the vertices that are neither in the cores K_1 nor in K_2, then the vertices from $K_2 \setminus K_1$, then the vertices from $K_1 \setminus K_2$, and finally from $K_1 \cap K_2$. The following figure shows how this ordering works:

H_1	2
H_2	2
Vertices that are not in a core	a
$K_2 \setminus K_1$	b
$K_1 \setminus K_2$	c
$K_1 \cap K_2$	d

By such grouping, we can assign more detailed types to segments:

Theorem 3.4.9. Considering the eight vertices in a segment S_A that are not contained in either handles. We assign four parameters (a, b, c, d) to each segment, where a is the number of vertices that are neither in K_1 nor in K_2, b is the number of vertices from $K_2 \setminus K_1$, c is the number of vertices from $K_1 \setminus K_2$, and d is the number of vertices that from $K_1 \cap K_2$. Then there are 11 different kinds of segments:

Three of them are of type 6 and 6: $(2, 0, 0, 6)$, $(1, 1, 1, 5)$, $(0, 2, 2, 4)$.

Three of them are of type 6 and 4: $(2, 0, 2, 4)$, $(1, 1, 3, 3)$, $(0, 2, 4, 2)$.

Five of them are of type 4 and 4: $(4, 0, 0, 4)$, $(3, 1, 1, 3)$, $(2, 2, 2, 2)$, $(1, 3, 3, 1)$, $(0, 4, 4, 0)$.

Proof. A simple fact is that $a + b + c + d = 8$. For type 6 and 6 segments, we have $b + d = c + d = 6$. For type 6 and 4 segments, we have $b + d = 4$ and $c + d = 6$. For type 4 and 4 segments, we have $b + d = c + d = 4$. It is not hard to list all different cases. □

Remark. We call the parameters (a, b, c, d) the *quad type* of the segment. Here is a table of number of segments of each quad type:

type	quad type	number of segments
6-6	$(2, 0, 0, 6)$	0
	$(1, 1, 1, 5)$	5
	$(0, 2, 2, 4)$	14
6-4	$(2, 0, 2, 4)$	9
	$(1, 1, 3, 3)$	35
	$(0, 2, 4, 2)$	34
4-4	$(4, 0, 0, 4)$	4
	$(3, 1, 1, 3)$	23
	$(2, 2, 2, 2)$	146
	$(1, 3, 3, 1)$	102
	$(0, 4, 4, 0)$	28

Table 3.1: Table of quad types

Notice that there are no segments of quad type $(2, 0, 0, 6)$, by enumeration.

CHAPTER 4

RECOVERING THE STRUCTURE OF Γ

This chapter describes our main idea to attack Conjecture 0.0.1. By Theorem 3.1.1, the eigenspace E_θ of the eigenvalue $\theta = 4$ has dimension 34. We want to recover a large induced subgraph G of Γ. Since the Euclidean representation of G is an embedding via E_θ, the Gram matrix of G should also have rank no greater than 34. When the order of G is small, we will check whether the Gram matrix of G is semi-positive definite, at this time the rank condition is quite weak. As we add more vertices to G, the rank condition will become stronger and stronger. When the order of G approaches 34, the rank condition will become even stronger, and will eliminate 99% of all cases. When the the order of G exceeds 34, either the case is eliminated by the rank condition, or we get a very strong condition that some of the vectors are linearly dependent.

4.1 Segments in the graph Γ

In this section, we will talk about the segments in the graph Γ.

We start from a maximal 3-clique with vertices A, B, and C. By Corollary 3.3.4 we know such a 3-clique exists. The neighbourhood of each vertex is one of the good graphs. Noticing that $N(A) \setminus \{B, C\}$ is a segment, we call it Σ_A. The two handles of Σ_A are the common neighbours of A and B, call them c_1 and c_2, and the common neighbours of A and C, call them b_1 and b_2. Symmetrically, we call the common neighbours of B and C

as a_1 and a_2, and call the other two segments Σ_B and Σ_C, respectively.

Here is a picture of such structure:

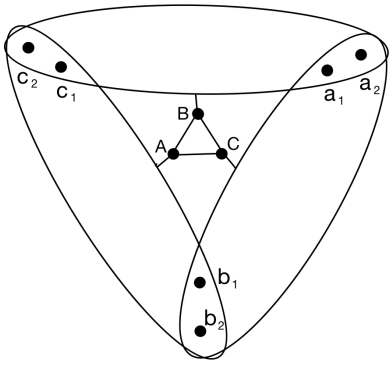


Figure 4.1: The maximal 3-clique and the three segments

In Figure 4.1, recall that we are excluding the segments with handles of type non-edge/edge, or in other words, type 4-6. Since the segments Σ_A, Σ_B, and Σ_C are symmetric with respect to vertices A, B, and C, which are also symmetric, we may always assume that the edge in handles appears first between $\{c_1, c_2\}$ then $\{b_1, b_2\}$ and finally $\{a_1, a_2\}$.

Moreover, after we go over all good graphs, we find the following fact:

Lemma 4.1.1. In every good graph G, there is always an edge $(v_1, v_2) \in E(G)$ such that (v_1, v_2) is not contained in a 3-clique, and the segment formed by deleting the edge (v_1, v_2) from G is either type 6-4 or 4-4.

The subgraph shown in Figure 4.1 is the structure we want to recover. However, we do not have to consider all the vertices.

Lemma 4.1.2. The Euclidean representation of vertices A, B, and C are linear combinations of the representation of vertices in Σ_A, Σ_B, and Σ_C.

Proof. Let σ_A be the sum of vectors in Σ_A. Similarly we have σ_B and σ_C. By Theorem

3.3.2, we have:

$$\sigma_A + B + C = 4A$$
$$\sigma_B + C + A = 4B$$
$$\sigma_C + A + B = 4C$$

Solving this linear system we get:

$$A = \frac{1}{10}(3\sigma_A + \sigma_B + \sigma_C)$$
$$B = \frac{1}{10}(\sigma_A + 3\sigma_B + \sigma_C)$$
$$C = \frac{1}{10}(\sigma_A + \sigma_B + 3\sigma_C)$$

□

By Lemma 4.1.2, we will only recover the vertices in $\Sigma_A \cup \Sigma_B \cup \Sigma_C$, as the 3-clique does not contribute to the rank of the Gram matrix.

Now we have a look at the properties of these joining segments.

Lemma 4.1.3. The vertices c_1 and c_2 cannot be adjacent to any vertex in Σ_C.

Proof. Let $c \in \{c_1, c_2\}$. The vertex c is not adjacent to C because the 3-clique $\{A, B, C\}$ is maximal. Hence they only have two common neighbours, which are A and B. Therefore, c cannot be adjacent to any vertex in Σ_C. □

Remark. We also have similarly conclusions: a_1 and a_2 are not adjacent to any vertex in Σ_A, b_1 and b_2 are not adjacent to any vertex in Σ_B.

Since the structure of three segments is symmetric, we can first focus on the relation between two segments. For two segments that share a common handle like Σ_A and Σ_B, we define the segment pair as following:

Definition 4.1.4. Let Σ_A and Σ_B be two segments with a common handle $H = \{c_1, c_2\}$. The other handle of Σ_A is $H_A = \{b_1, b_2\}$ and the other handle of Σ_B is $H_B = \{a_1, a_2\}$. A *segment pair* P_{AB} is a triple consisting of the graph $\Sigma_A \cup \Sigma_B$ and H_A and H_B, which are called handles of this segment pair.

Before we go deep into the analysis of the structure of segment pairs, let us first come up with a method to find out all possible gluings for two graphs, instead of just scanning all possibilities.

4.2 Amalgam

In this section, we analyze the joining structure of segments, and show the theory of gluing segments together. We first discuss amalgams of graphs.

Definition 4.2.1. An amalgam of graphs is a pair of graph homomorphisms $N_1 \xleftarrow{f_1} M \xrightarrow{f_2} N_2$, such that each $f_i : M \to N_i$, $i = 1, 2$, is an isomorphism of M onto an induced subgraph of N_i.

Definition 4.2.2. We say two amalgams $\mathcal{G}$: $N_1 \xleftarrow{f_1} M \xrightarrow{f_2} N_2$ and $\mathcal{G}'$: $N_1' \xleftarrow{f_1'} M' \xrightarrow{f_2'} N_2'$ are isomorphic, if there exist three graph isomorphisms $\phi_i : N_i \to N_i'$, $i = 1, 2$, and $\phi : M \to M'$ such that the following diagram commutes:

$$\begin{array}{ccccc} N_1 & \xleftarrow{f_1} & M & \xrightarrow{f_2} & N_2 \\ \downarrow{\scriptstyle \phi_1} & & \downarrow{\scriptstyle \phi} & & \downarrow{\scriptstyle \phi_2} \\ N_1' & \xleftarrow{f_1'} & M' & \xrightarrow{f_2'} & N_2' \end{array}$$

We will also use the following weaker equivalence on amalgams:

Definition 4.2.3. We say two amalgams $\mathcal{G}$: $N_1 \xleftarrow{f_1} M \xrightarrow{f_2} N_2$ and $\mathcal{G}'$: $N_1' \xleftarrow{f_1'} M' \xrightarrow{f_2'} N_2'$ have the same *type*, if for $i = 1, 2$, there exist graph isomorphisms $\phi_1 : N_i \to N_1'$ such that $\phi_i(\mathrm{Im} f_i) = \mathrm{Im} f_i'$.

To avoid going too deep into the theory of amalgams, here we directly show the result that we want:

Theorem 4.2.4. Let $\mathcal{G}$: $N_1 \xleftarrow{f_1} M \xrightarrow{f_2} N_2$ be an amalgam. For $i = 1, 2$, let A_i be the subgroup of $\mathrm{Aut}\, N_i$ that stabilize $\mathrm{Im} f_i$. Let ϕ_i be the mapping from A_i to $\mathrm{Aut}\, M$ that $\phi_i(\psi) = f_i \psi f_i^{-1}$, and let $B_i = \mathrm{Im}\phi_i$. Then there exist a bijection between the isomorphism classes of amalgams of the same type as $\mathcal{G}$ and the double cosets of B_1, B_2 in $\mathrm{Aut}\, M$.

Proof. The proof of this theorem is based on [12].

If an amalgam has the same type as $N_1 \xleftarrow{f_1} M \xrightarrow{f_2} N_2$, then it must be isomorphic to some amalgam $N_1 \xleftarrow{g_1 f_1} M \xrightarrow{g_2 f_2} N_2$, where g_1 and g_2 are elements in $\mathrm{Aut}\, M$. This amalgam is isomorphic to $N_1 \xleftarrow{f_1} M \xrightarrow{g_1^{-1} g_2 f_2} N_2$, hence we only have to consider for what condition will amalgams with this form be isomorphic. In other words, we need to find automorphisms ψ_1, ψ_2, φ such that:

(1) $f_1 \psi_1 = \varphi f_1$

(2) $g_1 f_2 \psi_2 = \varphi g_2 f_2$

By our definition, (1) is equivalent to $\psi \in B_1$ and (2) is equivlent to $g_1^{-1} \psi g_2 \in B_2$. Therefore, both conditions can hold if and only if $B_1 \cap g_1^{-1} B_2 g_2 \neq \varnothing$, which is equivalent to $B_2 g_1 B_1 = B_2 g_2 B_1$. Hence we have proved the theorem.

□

4.3 Gluing

Now, as an application of Theorem 4.2.4 to our main problem, we consider the following problem: for two vertices in Γ, the structure of their neighbourhoods can be easily found. Since they have some common neighbours, the structure of these two vertices' neighbourhoods can be viewed as gluing the two neighbourhoods such that keep the common neighbours invariant. Same argument works for similar gluing cases. And here we get an important conclusion direct from Theorem 4.2.4:

Theorem 4.3.1. Suppose that Σ_1 and Σ_2 are two graphs that have the set $H = \{H_1, \ldots, H_n\}$ as their common subgraphs. Let A_i be the subgroup of $\mathrm{Aut}(\Sigma_i)$ that stabilize all H_j's.

The number of non-isomorphic graphs that are formed by gluing Σ_1 and Σ_2 via H is the number of double-cosets of A_1 and A_2 on the automorphism group of H.

This leads to the 'gluing' function of our codes.

The 'gluing' function is to glue two graphs with respect to two families of subgraphs, and the output is the set of all possible gluings up to isomorphism. Suppose that we are gluing two graphs G_A and G_B with two families of subgraphs S_A and S_B. Firstly, we check whether the two induced subgraphs with respect to vertices in S_A and S_B, call them H_A and H_B, are isomorphic or not. If not, then we get an empty set, otherwise there are isomorphisms $\tau : H_B \to H_A$. Now for an element in the automorphism group $\sigma \in \mathrm{Aut}(H_A)$, we take those that maps S_A to S_B and stabilize all the subgraphs. After that, according to Theorem 4.3.1, we compute the stabilizer of $\mathrm{Aut}(G_A)$ and $\mathrm{Aut}(G_B)$ over the sets S_A and S_B, denoted as A_1 and A_2, then by Theorem 4.3.1, all possible gluings up to isomorphism can be counted by the double cosets.

Now come back into the case of segments. The two segments Σ_A and Σ_B have common vertices c_1 and c_2. Therefore, for two segments, we need to have them glued along one of the handles. Notice that a type 4 handle can only glue with another type 4 handle, and a type 6 handle can only glue with another type 6 handle. This gluing will give us a segment pair.

Recall that we have dropped all the segments of type 4-6, and we are always gluing the first handles from both segments. Therefore, a segment pair can only consist of two segments of type 6-6, or a segment of type 6-6 and a segment of type 6-4, or two segments of type 6-4, or two segments of type 4-4.

Theorem 4.3.2. Let Σ_A be a segment, $H_A = \{c_{a1}, c_{a2}\}$ be a handle of Σ_A. Let Σ_B be another segment with a handle $H_B = \{c_{b1}, c_{b2}\}$ of the same type as H_A. Then, up to isomorphism, the number of segment pairs formed by the amalgam that glue Σ_A to Σ_B along the handles H_A and H_B depends on the number of orbits of $\mathrm{Aut}\,\Sigma_A$ on H_A.

In our case, if $\mathrm{Aut}\,A$ switches the vertices in the handle H, then there is only one way

to glue the segment Σ_A with another segment along the handle H. We have the following conclusion by computation:

Theorem 4.3.3. In total there are 86333 different segment pairs. Among them there are 281 segment pairs consist of two segments of type 6-6, 2249 segment pairs consist of a segment of type 6-6 and a segment of type 6-4, 4851 segment pairs consist of two segments of type 6-4, 78952 segment pairs consist of two segments of type 4-4.

We did all these enumeration in GAP. All the segment pairs are stored in the file `segment_pairs.g`. Each segment pair contains the first segment Σ_1 and the second segment Σ_2. Besides, there is also an image set showing the mapping of the handle of Σ_1 into the handle of Σ_2 while gluing.

Gluing two type 6 handles will add 6 edges for matching as shown in Theorem 4.5.1. Comparing with gluing type 4 handles which only adds 4 edges, gluing type 6 handles has more cases to compute. Therefore, in the future when we are about to glue three segments together, we would always like to glue type 6 handles first. This is also what we have done in the code, that arranges segments in the order type 6 and 6, then type 6 and 4, then type 4 and 4. And when gluing two segments at this step, we always glue the first handle with another first handle.

There might be a small chance that two segment pairs are isomorphic. Such cases come from gluing two segments that only differ by taking orders of handles, and these segments have symmetric structure with respect to the handles. However, eliminating them will cost more time than just to go ahead, so we will keep the isomorphic copies, if there are any.

4.4 Start gluing

4.4.1 The LDLT decomposition

For this section, we would to say thank you to Madeleine Whybrow, who provided this algorithm.

The LDLT decomposition is a variant of the classical Cholesky decomposition [24], which is used for solving the linear system $Ax = b$ by factoring the positive definite real matrix A into $A = LDL^T$, where L is a lower unit triangular matrix and D is a diagonal matrix. In other words, L is a lower triangular matrix with all its diagonal entries are 1. In this section we will introduce the the LDLT decomposition and give a proof of its correctness. The code we are using is in the Appendix.

The LDLT decomposition is useful in our research. We will see that the inner product of the images of an Euclidean representation will form a Gram matrix, which should be semi-positive definite. When constructing possible structures of Γ, we apply the LDLT decomposition on its corresponding Gram matrix. If the decomposition fails, then that case cannot happen.

In order to construct the LDLT decomposition, we need the following theorem on Schur complement:

Theorem 4.4.1. Let $A = \begin{bmatrix} X & Y \\ Y^T & Z \end{bmatrix}$ be an $n \times n$ symmetric matrix, X and Z are square matrices, X is positive definite. Then A is semi-positive definite if and only if $Z - YX^{-1}Y^T$ is semi-positive definite. The matrix $Z - YX^{-1}Y^T$ is called the *Schur complement* of X in A.

Proof. By the Spectral Theorem, since X is positive definite, there exist an orthogonal matrix P such that $P^TXP = P^{-1}XP = Q$, where Q is a positive definite diagonal matrix.

Now we have A is semi-positive definite if and only if

$$B = \begin{bmatrix} P^T & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} X & Y \\ Y^T & Z \end{bmatrix} \begin{bmatrix} P & 0 \\ 0 & I \end{bmatrix} = \begin{bmatrix} Q & YP \\ P^T Y^T & Z \end{bmatrix} \tag{4.1}$$

is semi-positive definite.

Observe that $P^T XP = Q$ and $P^T = P^{-1}$, we have $X = PQP^T$, then we have $X^{-1} = (P^T)^{-1}Q^{-1}P^{-1} = PQ^{-1}P^T$.

Now we have a quadratic form defined with respect to B:

$$f_B(x) = x^T Bx = x_1^T Q x_1 + 2x_1^T YPx_2 + x_2^T Z x_2, \tag{4.2}$$

$x = (x_1, x_2)$ is a vector where x_1 has the same dimension with X and x_2 has the same dimension with Z. If we treat the function f_B as $f_B(x_2)$ (in other words, we fix x_1), since $\frac{df_B(x_1)}{dx_1} = 2Qx_1 + 2YPx_2$, $\frac{d^2 f_B(x_1)}{(dx_1)^2} = 2Q$ and Q is positive definite, we conclude that f_B is a convex function with respect to x_1, with the minimum achieved when $\overline{x_1} = -Q^{-1}P^T Y^T x_2$. By plugging $\overline{x_2}$ into (4.2), and noticing that $X^{-1} = PQ^{-1}P^T$, we can draw the conclusion that f_B is non-negative if and only if $f_{B'}$, where $B' = Z - YX^{-1}Y^T$, is non-negative.

Therefore, A is semi-positive definite if and only if $Z - YX^{-1}Y^T$ is semi-positive definite. □

Theorem 4.4.2. Let A be an $n \times n$ semi-positive definite matrix. Then we are able to factor A as $A = LDL^T$, where L is an $n \times n$ lower triangular matrix, with all its diagonal entries $L(i,i) = 1$ for all i, and D is an $n \times n$ diagonal matrix with non-negative entries.

Proof. This theorem shows that the so-called LDLT decomposition works. We prove it by induction over n:

For the base case $n = 1$, we have $L = \begin{bmatrix} 1 \end{bmatrix}$ and $D = A$ which is trivial.

Now we assume that the theorem holds for the case where A is $n \times n$, consider the

case that A is $(n+1) \times (n+1)$, in the form:

$$A = \begin{bmatrix} a_{11} & v^T \\ v & A' \end{bmatrix}. \tag{4.3}$$

We first assume that $a_{11} > 0$. By Theorem 4.4.1, $A' - \dfrac{vv^T}{a_{11}}$ is semi-positive definite. Hence by the induction hypothesis, there exist some lower unitriangular L and non-negative diagonal D such that $A' - \dfrac{vv^T}{a_{11}} = L'D'(L')^T$. Now we have:

$$A = \begin{bmatrix} 1 & 0 \\ v/a_{11} & L' \end{bmatrix} \begin{bmatrix} a_{11} & 0 \\ 0 & D' \end{bmatrix} \begin{bmatrix} 1 & v^T/a_{11} \\ 0 & (L')^T \end{bmatrix}, \tag{4.4}$$

which is the LDLT decomposition of A that we need.

Now consider the case that $a_{11} = 0$. By the induction hypothesis, there exist some L' unit-lower triangle and D' non-negative diagonal such that $A' = L'D'(L')^T$, then we have:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & L' \end{bmatrix} \begin{bmatrix} a_{11} & 0 \\ 0 & D' \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & (L')^T \end{bmatrix}, \tag{4.5}$$

which is the LDLT decomposition of A that we need. And this completes the proof of the theorem. □

Here is the pseudocode of the algorithm:

Input: $A = A(i,j)$, $1 \le i, j \le n$, a symmetric matrix.

Output: Diagonal matrix D and lower-triangular matrix L, $L(i,i) = 1$.

Set $L(i,j)$ and $D(i,i)$ to the null matrix.

for $i = 1$ to n **do**

$D(i,i) = A(i,i) - \sum_{j=1}^{i-1} L(i,j)^2 D(j,j)$

if $D(i,i) = 0$ **then**

for $j = i + 1$ to n **do**

$S = \sum_{k=1}^{i-1} L(j,k)L(i,k)D(k,k)$

if $A(j,i) = S$ **then**

$L(j,i) = 0$

else

return false

end if

end for

else

for $j = i + 1$ to n **do**

$S = \sum_{k=1}^{i-1} L(j,k)L(i,k)D(k,k)$

$L(j,i) = (A(j,i) - S)/D(i,i)$

end for

$L(i,i) = 1$

end if

end for

4.4.2 A remark about the algorithm

In the algorithm there is a line that we justify whether the given matrix A is semi-positive definite or not. This is because the input is the possible local Gram matrix of the graph, and if it is not semi-positive definite, such local graph does not exist. However it is not quite clear the reason why in that case we can claim that A is not semi-positive. So we need the following lemma:

Lemma 4.4.3. If $D(i,i) = 0$ and $A(j,i) \neq \sum_{k=1}^{i-1} L(j,k)L(i,k)D(k,k)$ for some $i + 1 \leq j \leq n$ then A is not semi-positive definite.

Proof. Assume that A is semi-positive definite. By Theorem 4.2, there is a decomposition $A = LDL^T$. Since $D(i,i) = 0$, for any arbitrary $i + 1 \leq j \leq n$, we have:

$$
\begin{aligned}
A(j,i) &= \sum_{k=1}^{n} (LD)(j,k) L^T(k,i) \\
&= \sum_{k=1}^{n} (LD)(j,k) L(i,k) \\
&= \sum_{k=1}^{n} L(i,k) \sum_{s=1}^{n} L(j,s) D(s,k). \qquad (4.6) \\
&= \sum_{k=1}^{n} L(j,k) D(k,k) L(i,k) \\
&= \sum_{k=1}^{i-1} L(j,k) L(i,k) D(k,k)
\end{aligned}
$$

Here the fourth equality since $D(s,k) = 0$ for $s \neq k$. The fifth equality holds since $L(i,k) = 0$ for $k > i$.

Hence, if A is semi-positive definite, then we must have

$$
A(j,i) = \sum_{k=1}^{i-1} L(j,k) L(i,k) D(k,k),
$$

for all $i+1 \leq j \leq n$.

Otherwise A is not semi-positive definite. □

4.4.3 Extending the LDLT decomposition by one dimension

In this section, we assume that we have done the LDLT decomposition for an $n \times n$ matrix A, and we would like find a fast algorithm do the LDLT decomposition for an $(n+1) \times (n+1)$ matrix B that is formed by adding a row and a column to A.

Let $A = LDL^T$ be a symmetric semi-positive definite matrix with its LDLT decomposition of order n. Let B be the symmetric matrix formed by adding the $(n+1)-$th row and $(n+1)-$th column to A. Since B is symmetric, we denote the new-added row vector as r, and the corresponding column vector as r^T. Let $B = L'D'L'^T$ be its possible LDLT decomposition.

If we apply the whole LDLT decomposition algorithm on B, we notice that $L'(i,j) =$

$L(i,j)$ and $D'(i,j) = D(i,j)$ for all $1 \leq i,j \leq n$. Since they are already know, we can omit the re-computation and just calculate the final row and column of L' and D':

For $L'(n+1,i)$, $1 \leq i \leq n$, by the LDLT decomposition algorithm, if $D'(i,i) = 0$ then if

$$R(i) - \sum_{j=1}^{i-1} L'(n+1,j)L'(i,j)D'(j,j) = 0,$$

we have $L'(n+1,i) = 0$; else if

$$R(i) - \sum_{j=1}^{i-1} L'(n+1,j)L'(i,j)D'(j,j) \neq 0,$$

then such decomposition do not exist by Lemma 4.4.3. If $D'(i,i) \neq 0$ then we have

$$L'(n+1,i) = \frac{1}{D'(i,i)} \left(R(i) - \sum_{j=1}^{i-1} L'(n+1,j)L'(i,j)D'(j,j) \right).$$

What is more, we have $L'(n+1,n+1) = 1$ and

$$D'(n+1,n+1) = R(n+1) - \sum_{i=1}^{n} L'(n+1,i)L'(n+1,i)D'(i,i).$$

Therefore, in the case of expanding the matrix, instead of going over the whole algorithm again, we may only compute $L'(n+1,i)$ and $D'(n+1,n+1)$, which will save a lot of time. This is the 'AddOne' function in our code, shown in the Appendix A.2.

4.5 Induced subgraph on a segment pair

Now we are able to glue two segments along a handle of the same type to get a segment pair. In order to recover the induced subgraph of these vertices, we need to justify all the edges in a segment pair.

Theorem 4.5.1. Let P be a segment pair with two segments Σ_A and Σ_B. And let their common handle be $H = \{c_1, c_2\}$. Let K_A be the core in Σ_A corresponding to H and K_B

be the core in Σ_B corresponding to H. Then the only vertices in $\Sigma_A \setminus H$ that are adjacent to vertices in $\Sigma_B \setminus H$ are the vertices in K_A, the only vertices in $\Sigma_B \setminus H$ that are adjacent to vertices in $\Sigma_A \setminus H$ are the vertices in K_B. Furthermore, each vertex has exactly one neighbour in the other core.

Proof. Let $v \in K_A$. Since v and B are non-adjacent, they should have two common neighbours, one of which is A. As C cannot be adjacent to v, the other common neighbour of v and B must be in Σ_B. Similar for vertices in K_B.

On the other hand, let $v \in \Sigma_A$ that is adjacent to a vertex $c \in H$. The vertices v and B are not adjacent, and they have two common neighbours c and A, so v cannot have any more neighbours in Σ_B. □

Corollary 4.5.2. The vertices in the cores of Σ_A and Σ_B with respect to the common handle H form a matching.

By Theorem 3.4.8, we know that if $c_1 \not\sim c_2$ then the size of core is four, if $c_1 \sim c_2$ then the size of core is six.

After gluing the first two segments, we will go through all possible matchings between the cores and check whether the Gram matrix of that graph is semi-positive definite, as discussed in Section 3.1.

Theorem 4.5.3. If the matching between the cores is type 4 then there are 24 possible matchings. If it is type 6 then there are 720 possible matchings.

Proof. Every vertex in one core need one neighbour in the other core without any further restriction. So for a type 4 matching there are $4! = 24$ possibilities and for a type 6 matching there are $6! = 720$ possibilities. □

For all the possible segment pairs equipped with a matching, we then glue them with the third segment, adding matchings and check the Gram matrix again. Details will be discussed in the next chapter. Our general idea of eliminating impossible cases is just like this.

CHAPTER 5

RECOVERING THE FULL INDUCED SUBGRAPH

5.1 The enumeration tree

As we discussed in Theorem 4.5.1, there is a matching between vertices in the cores of a segment pair. Definitely it is not appropriate to consider each matching independently. What we do in our code will be explained in this section.

First, recall that we have recorded all segments by differing the order of their handles. Details were discussed in Section 3.4. When gluing segments, we always glue the first handle of the first segment with the first handle of the second segment, the image of the mapping is contained in the record of the segment pair. This will make us deal with type 6 matching before type 4 matching. We know the fact that there are much more possibilities in type 6 matching than type 4 matching, hence we can check the Gram matrix with fewer dimension at this step, which will cost less time in general.

Secondly, instead of applying each matching to the two cores directly, we create an enumeration tree.

Definition 5.1.1. The *enumeration tree* is a tree that the computation about matchings between two segments will go along with. At each node of this tree, the following information is recorded: the current level, the image vertex, the next brother, and the first child.

The current level shows the ordinal of the vertex in the core that will be matched. It

starts at 1 and the maximum value is the size of the core.

The image vertex denotes the vertex in core of the second segment that will be matched at this step. The value of which is the ordinal of this vertex in the corresponding core.

The next brother denotes the node in this enumeration tree which is the next possible case for matching the current vertex. The value of which is the ordinal of the next brother's node in the tree, is 0 if this node has no more brothers.

The first child denotes the node in this gluing tree which is the first possible case matching the next vertex after the current vertex. The value of which is the ordinal of the first child's node in the tree, is 0 if this node has no more children.

The local structure of such tree looks like this:

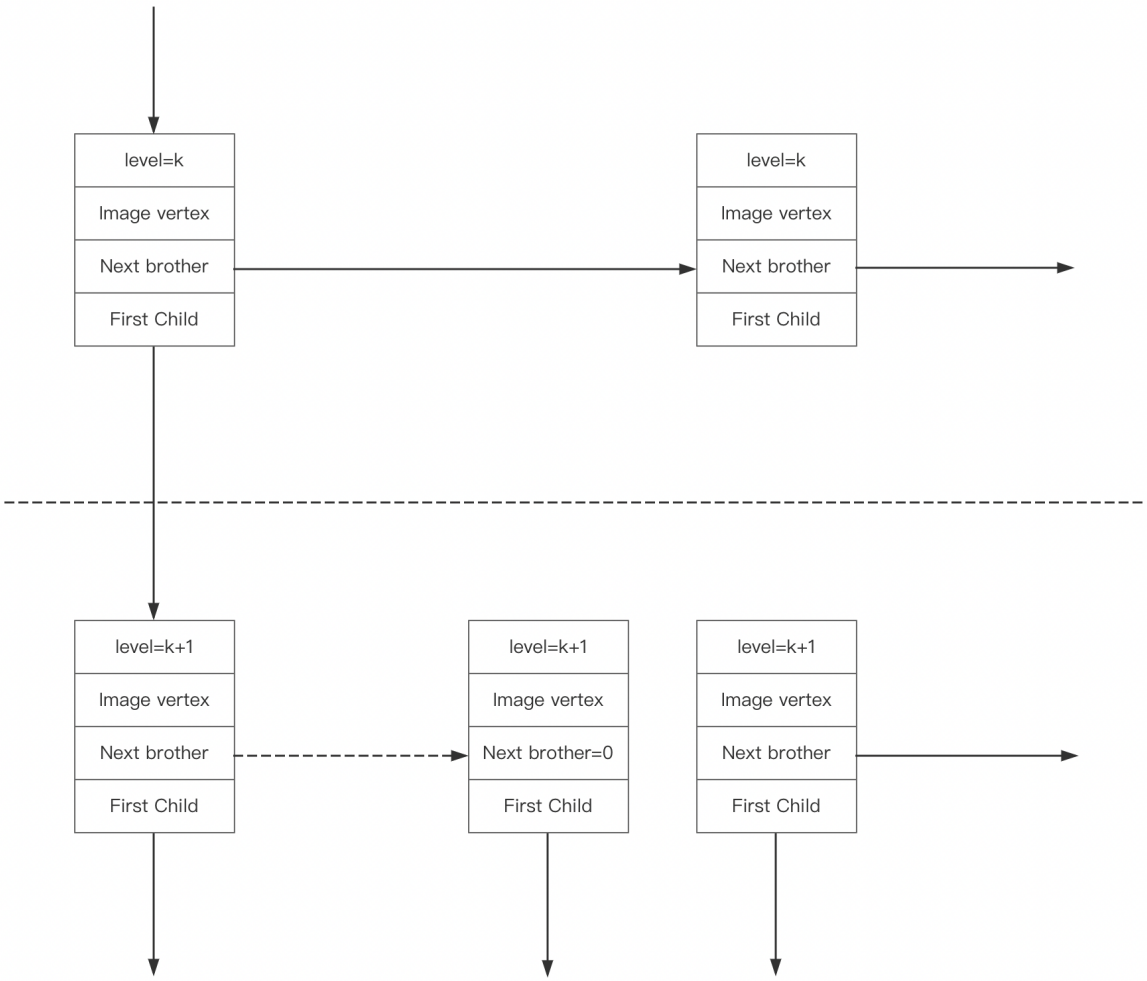


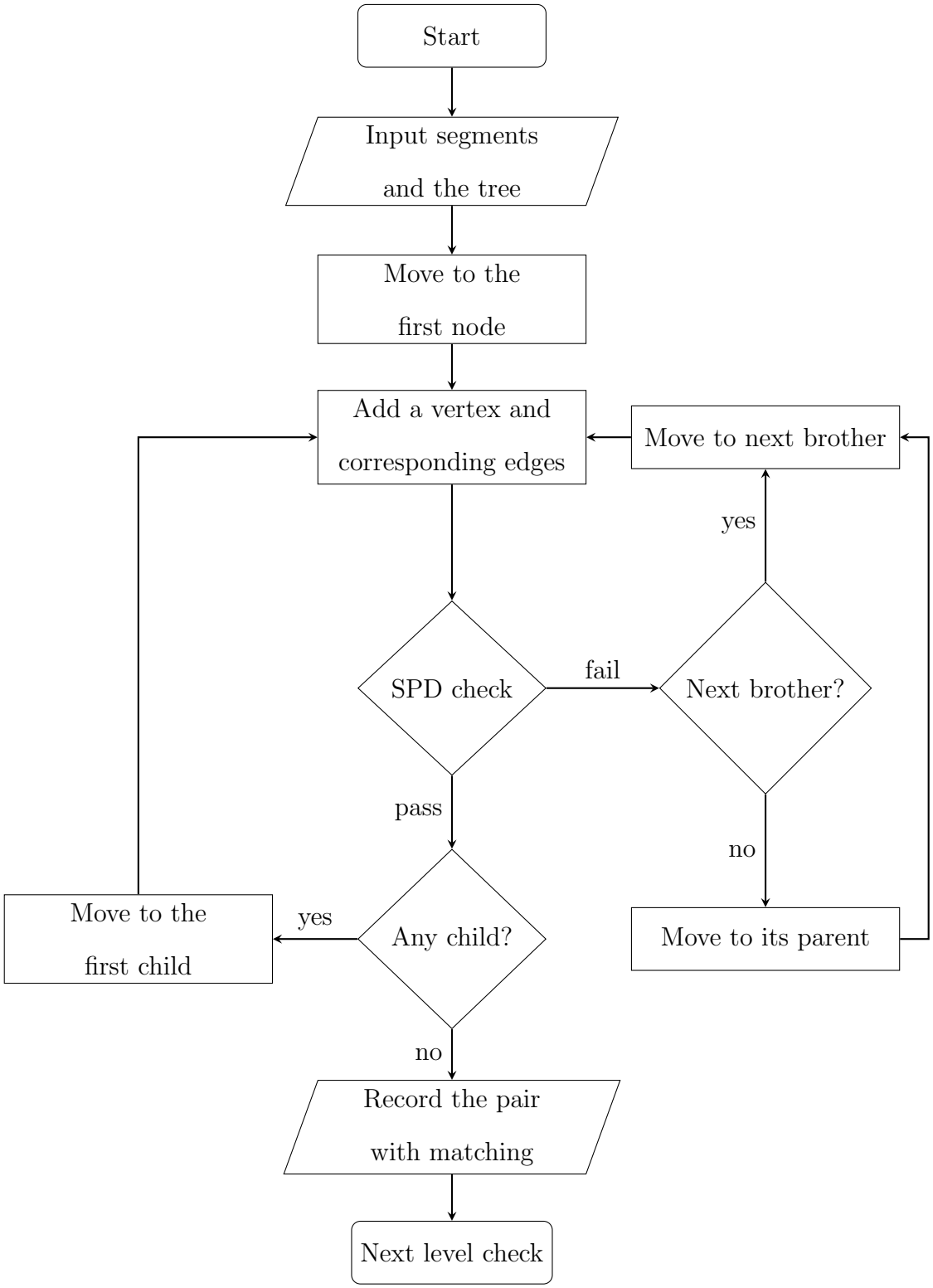
Figure 5.1: Local structure of the enumeration tree

We still need to explain the order of each node that we save in the file. Starting from the first node, at which we are mapping the first vertex to the first vertex. Then we keep looking for the first child until the last level. It is obvious that any node on the last level does not have any children or brothers. Then we go back to the upper level and look for the next brother. Whenever there is no next brother, we will go one level up and look for the next brother, then go over all its children. After we traverse all branches of the last node at level 1, the tree is completed.

Noticing that we only have type 4 and type 6 gluing in this step between two segments, hence we will have two gluing trees working for these two cases. The enumeration tree for type 4 has 64 nodes and for type 6 has 1956 nodes, and they are restored in the file `small_trees.g`.

Now we explain how the enumeration tree works. The gluing will start from the first node of the corresponding tree, the vertex and the edge for matching are added as the information at the current node. If the current structure passes the check, we move to the next child of this node, otherwise we move to the next brother. Then we repeat the checking. If the check fails at a node and there are no more brothers, it means that the whole branch is done, we move back to the parent's next brother. If the check is passed at a node and there are no more children, it means we have successfully glued two segments with a matching. The result will be recorded for the next step.

In order to make the algorithm more clear, here is a flow chart for this procedure:



As a brief summary, there are a significant amount of cases get eliminated at this step when we glue the amalgam of two segments and add matchings (about 60% to 80% depending on the segment pair).

5.2 The segment triple structure

Now we complete the recovery by gluing the third segment to a segment pair. Apart from the two handles, there are eight vertices in the third segment, which will be involved into the matchings with both the other two segments. As discussed in Section 3.4, these vertices are recorded in specific order. Furthermore, by Theorem 3.4.9, all of them are subdivided into further cases.

Firstly, we give the definition of a segment triple.

Definition 5.2.1. Let P be a segment pair with handles H_A and H_B. Let S be a segment with handles H_1 and H_2, where H_1 and H_A are of the same type, and H_2 and H_B are of the same type. We call the graph T comprising of the graph P and the graph S by gluing H_1 with H_A and H_2 with H_B a *segment triple*.

In order to enumerate all segment triples, Theorem 4.3.1 still works for this case.

Theorem 5.2.2. Let P be a segment pair with handles H_{p1} and H_{p2}, S be a segment with handles H_{s1} and H_{s2}. The handles H_{p1} and H_{s1} are of the same type, H_{p2} and H_{s2} are of the same type. Then up to isomorphism, the number of segment triples T formed by the amalgam of P and S is 4, 2 or 1, depending on the number of orbits of $\mathrm{Aut}\, P$ on H_{p1} and H_{p2}.

Proof. Recall that in the gluing of this step, the order of two handles is fixed. That is, we are only able to glue H_{p1} with H_{s1}, and glue H_{p2} with H_{s2}. Therefore, the automorphism group of vertices in $H_{p1} \cup H_{p2}$ is $C_2 \times C_2$. The number of orbits on this group can only be 4, 2, or 1. □

Remark. In principle, we should compute segment triples as in Theorem 5.2.2. However, in most cases, the automorphism group $\mathrm{Aut}\, P$ of a segment pair P is trivial. Therefore, instead of making the code too complicated, we just simply gluing the two pair of handles in all four ways. This might give us very few isomorphic segment triples, but almost does not increase any work.

Before moving forward, let us use a new way of view on the segment triples in order to reduce our work. Recall that in Theorem 3.4.8 we have shown an edge handle have core of size 6 and a non-edge handle have core of size 4. A matching between two size 6 cores will have $6! = 720$ different cases, which is too many comparing with a matching between two size 4 cores, who has $4! = 24$ different cases. Therefore, if we can reduce the number of type 6 matchings, especially the 6-6 types, while enumerating, we will save much time.

In order to achieve such goal, we first view the segment triple T as three independent segments Σ_A, Σ_B and Σ_C. Since each segment comes from a goodgraph, which are ordered in the file `goodgraph.g`, we select the segment Σ with the smallest ordinal of the corresponding goodgraph G. To get a segment from G, we select an edge and delete it, as the definition. We have the following lemma:

Lemma 5.2.3. In every goodgraph G there exist an edge not contained in a 3-clqiue. By removing such edge we will get a segment Σ that is either type 4-4 or 6-4.

In fact, Lemma 5.2.3 is proved by enumeration, going over all 39 goodgraphs we have. The codes of this procedure are attached in the Appendix A.3. For 36 of them we are able to get type 4-4 segments, only for the remaining three of them the best we can get are type 6-4 segments.

As T is symmetric with respect to the three segments, we can always make Σ be the first one in the triple, i.e. Σ_A. Notice that Σ_A is the first occurrence in the list of segments, and is not type 6-6. By our ordering, both Σ_B and Σ_C are not type 6-6. Such selection reduces a lot time, since now we are only focusing on 39 different segments as Σ_A, and none of them are type 6-6.

In the file `segment_triples.g`, all the segment triples are recorded as several lists with respect to a fixed segment pair. In each list, every record contains the segment pair form the file `segment_pairs.g` and the third segment, and the images of the mapping from the handles of the segment pair to the handles of the third segment.

Starting from a segment pair P formed by Σ_A and Σ_B, we add matchings to the vertices in the cores K_A and K_B along the enumeration tree as introduced in Section 5.1.

For the segment pair P that is equipped with a feasible matching, we start adding vertices of Σ_C. For the two cores in Σ_C, we call the one that will match with vertices in Σ_A as K_1 and the other one K_2. We add the vertices of Σ_C in the same order with the quad type: the vertices that are in the neither cores, then the ones only in K_2, then the ones only in K_1, and finally those in $K_1 \cap K_2$.

According to the above analysis, we also need to construct an enumeration tree for adding vertices with matchings, but different from the previous one. We call this one a big enumeration tree.

5.3 The big enumeration tree

The big enumeration tree we construct will be similar to what we had in Section 5.1.

Definition 5.3.1. A *big enumeration tree* is a tree that the computation about matchings between a segment pair and a segment will go along with. At each node of this tree, the following information is recorded: the current level, two image vertices, the next brother, and the first child.

Most of the information is similar to the enumeration tree. At a node of a big enumeration tree, there are two image vertices, showing how the object vertex will be matched with the vertices in the two cores of the segment pair.

Remark. All the eight vertices in Σ_C that are not in either handles might be matched with vertices in P. Hence the maximum level of a big enumeration tree is eight. The local structure of which looks like this:

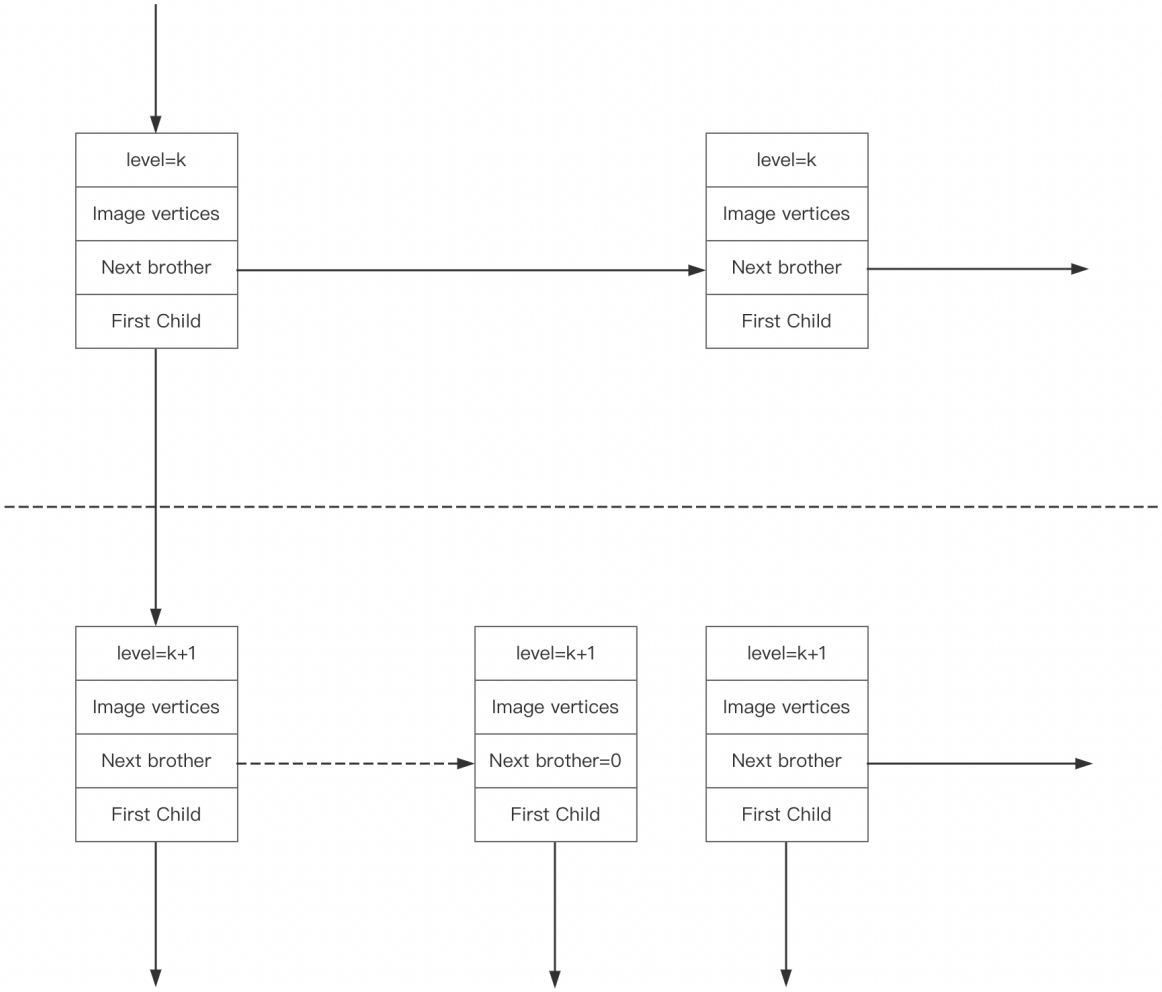


Figure 5.2: Local structure of a big enumeration tree

In order to not have too many different enumeration trees, as shown in Lemma 3.4.9, we record the vertices in the third segment in the order as the quad types (a, b, c, d). Then we can just construct enumeration trees for the listed different quad types regardless of what exactly is the segment pair.

When computing all possible matchings between P and Σ_C along the enumeration tree with respect to Σ_C, the algorithm for this step is almost same as the algorithm for Step 1, which is introduced in Section 5.1. However, there is still something different:

The current level in the big enumeration tree denotes the vertex in cores of Σ_C that will be matched at this step. The value of which is the ordinal of this vertex in the vertex set of $\Sigma_C \setminus (H_1 \cup H_2)$.

The image vertices in the big enumeration tree denote the vertices in the matching set of the first and the second segment that will be matched at this step. The values of which are the ordinal of these vertices in the corresponding matching set. If the related object vertex does not need a matching in one of the sets, then the value here will be 0.

As a summary, over 99.9% cases are eliminated at this step. In fact, quite a lot cases are found to be impossible even at level 6 or level 7. Those surviving cases will move to next level check and get dealt with methods in the next chapter.

CHAPTER 6

ELIMINATING CASES

6.1 Vertices that are not in the segment triple

As we discussed in Section 5.2, a segment triple T contains 30 vertices in total, not counting the three initial vertices A, B, and C, which are contained in the linear span of T. Now we want to add more vertices to T. Let $\hat{T} = T \cup \{A, B, C\}$.

We want to limit the possibilities for the set of neighbours in T of a vertex $x \notin \hat{T}$. We have the following theorem:

Theorem 6.1.1. Suppose that $x \notin \hat{T}$, and Σ is a segment in T. Then:

1. The vertex x has exactly two neighbours, say y and z, in Σ.

2. If y and z are adjacent, then they have at most one common neighbour in T.

3. If y and z are not adjacent, then they have no common neighbours in T.

Proof. Without lost of generality, we may assume that $\Sigma = \Sigma_A$.

Since x is not adjacent to A, x should have two common neighbours with A. As x is not adjacent to B and C, it means that x should have two neighbours in $N(A) \setminus \{B, C\} = \Sigma_A = \Sigma$. We call them y and z. This proves the first statement.

Recall that the graph Γ has parameters $\lambda = 3$ and $\mu = 2$. The two vertices y and z already have two common neighbours A and z. If they are adjacent to each other then

they can have at most one common neighbour in T since $\lambda = 3$. If they are not adjacent, then they cannot have any more common neighbours because $\mu = 2$. □

Remark. Note that each pair of segments are glued along a handle, consisting of two vertices. It is possible for x to have neighbours in the handle. In this case, the total number of neighbours of x in this segment triple T may less than six.

Lemma 6.1.2. For a vertex x not in $\hat{T}$, the number of neighbours of x in T is at least three.

The lower bound of three is achieved when x has one neighbour in each of the three handles in T. We will denote by $N_T(x)$ the set of neighbours of x in T.

Corollary 6.1.3. If x and x' are distinct vertices not in $\hat{T}$, then $N_T(x) \neq N_T(x')$.

Proof. Suppose by contradiction that $N_T(x) = N_T(x')$. If $|N_T(x)| \geq 4$ then it is impossible since x and x' cannot have four or more common neighbours as $\lambda = 3$ and $\mu = 2$. If $|N_T(x)| = 3$ then these three vertices in $N_T(x)$ are in three handles. We take two of them, say u and v, then $u \not\sim v$ by Lemma 3.4.4. Therefore they can have only two common neighbours, but they are adjacent to x, x', and one of A, B, or C, a contradiction. □

Therefore, the $52 (= 85 - 30 - 3)$ vertices x all have distinct sets of $N_T(x)$. Using this we will identify vertices outside of $\hat{T}$ with their sets $N_T(x)$.

Given a segment triple T that is still alive after the previous steps, we next enumerate all possible sets $N_T(x)$ of size between three and six, using the properties from Corollary 6.1.3.

We first create an array called `comm` which counts for each pair of vertices (u, v) in T that how many common neighbours of u and v must have in $\Gamma \setminus \hat{T}$. In other words, we count the number of common neighbours of u and v in $\hat{T}$ and subtract this number from either 3 (if $u \sim v$) or 2 (if $u \not\sim v$). If this difference for some pair (u, v) is negative then this already eliminate the segment triple T. Otherwise, we create an array `downs` of all possible additional vertices. For this we select a pair of vertices in each segment, verify that the pairs intersect with handles in a consistent way:

Lemma 6.1.4. Let X and Y be two elements from $\{A, B, C\}$. Let P_X and P_Y be the pairs of vertices that are adjacent to x in Σ_X and Σ_Y respectively. Let $H_{XY} = \Sigma_X \cap \Sigma_Y$ be the handle between the two segments. Then we must have $P_X \cap H_{XY} = P_Y \cap H_{XY}$.

Indeed, if, say, $u \in P_X \cap H_{XY}$ but $u \notin P_Y \cap H_{XY}$, then x has three neighbours in Σ_Y, a contradiction.

We go through all pairs of P_A, P_B, and P_C. We check whether they are compatible according to Lemma 6.1.4. After that, for any pair of vertices in $P_A \cup P_B \cup P_C$, they should not exceed the maximum number of common neighbours after we add x. Otherwise this x is impossible to add, and we remove it from `downs`.

While selecting elements in `downs`, for each pair of vertices we select, we check the number of common neighbours of them together with the condition that whether they are adjacent or not. If they have enough common neighbours then this is impossible. In fact, this condition eliminate quite a lot cases.

For the vertices still left in `downs`, we will check them against the projection. Details will be explained in the next section.

6.2 Projection of new vectors

For all vertices that are not in $\hat{T}$, each of them should have 14 neighbours, some of them are in T, and the remaining ones are outside of $\hat{T}$. In total, there are $52 = 85 - 3 - 30$ vertices that are not in $\hat{T}$, each of them should have enough non-neighbours. In order to justify whether two vertices are able to be adjacent with each other or not, we need the following theorem:

Theorem 6.2.1. Let U be the linear space spanned by the vectors of the vertices in the segment triple T, and $U^\perp$ be the orthogonal space of U. For two vertices a and b that are not in U, let v_a and v_b be the vectors corresponding to a and b. Then we have:

1. The vertices a and b can be adjacent to each other only if

$$\left|\frac{2}{7} - \langle \mathrm{proj}_U v_a, \mathrm{proj}_U v_b \rangle\right| \leq |\mathrm{proj}_{U^\perp} v_a| \cdot |\mathrm{proj}_{U^\perp} v_b|$$

2. The vertices a and b can be non-adjacent to each other only if
$$\left|-\frac{1}{14} - \langle \mathrm{proj}_U v_a, \mathrm{proj}_U v_b \rangle\right| \leq |\mathrm{proj}_{U^\perp} v_a| \cdot |\mathrm{proj}_{U^\perp} v_b|$$

Proof. By the equation (3.1) we know that if a and b are adjacent then $\langle v_a, v_b \rangle = \frac{2}{7}$ and if they are non-adjacent then $\langle v_a, v_b \rangle = -\frac{1}{14}$.

Now we decompose v_a and v_b by projecting them to U and $U^\perp$. Then we have:

$$v_a = \mathrm{proj}_U v_a + \mathrm{proj}_{U^\perp} v_a$$
$$v_b = \mathrm{proj}_U v_b + \mathrm{proj}_{U^\perp} v_b$$

and hence

$$\langle v_a, v_b \rangle = \langle \mathrm{proj}_U v_a, \mathrm{proj}_U v_b \rangle + \langle \mathrm{proj}_{U^\perp} v_a, \mathrm{proj}_{U^\perp} v_b \rangle.$$

In the case that $a \sim b$, $\langle v_a, v_b \rangle = \frac{2}{7}$ and hence we have:

$$\frac{2}{7} - \langle \mathrm{proj}_U v_a, \mathrm{proj}_U v_b \rangle = \langle \mathrm{proj}_{U^\perp} v_a, \mathrm{proj}_{U^\perp} v_b \rangle.$$

By taking the module on both sides, we have:

$$\left|\frac{2}{7} - \langle \mathrm{proj}_U v_a, \mathrm{proj}_U v_b \rangle\right| = \left|\langle \mathrm{proj}_{U^\perp} v_a, \mathrm{proj}_{U^\perp} v_b \rangle\right|.$$

Therefore, we conclude that

$$\left|\frac{2}{7} - \langle \mathrm{proj}_U v_a, \mathrm{proj}_U v_b \rangle\right| \leq |\mathrm{proj}_{U^\perp} v_a| \cdot |\mathrm{proj}_{U^\perp} v_b|.$$

In the case $a \not\sim b$, by similar argument, we have:

$$\left|-\frac{1}{14} - \langle \mathrm{proj}_U v_a, \mathrm{proj}_U v_b \rangle\right| \leq |\mathrm{proj}_{U^\perp} v_a| \cdot |\mathrm{proj}_{U^\perp} v_b|.$$

□

Remark. One may notice that the conditions look much weaker than the exact equations. The reason why we are using this condition is that we are not able to compute $\langle \text{proj}_{U^\perp} v_a, \text{proj}_{U^\perp} v_b \rangle$ directly since we have very little information about $U^\perp$. However, $|\text{proj}_{U^\perp} v_a|^2 = |v_a|^2 - |\text{proj}_U v_a|^2$ can be computed by the information we have.

By the condition given in Theorem 6.2.1, some of the possible vertices generated by triple-pairs are no longer available since they cannot have enough neighbours or non-neighbours. These two conditions turn into two functions in our code: `AreGoodNeighbours` and `AreGoodNonNeighbours`, together a function called `AreCompatibleVerts` will determine a pair of vertices that satisfy at least one of the conditions. After removing these impossible vertices, we do such check again to see if any vertices become impossible at this time. This check will stop when no more new vertices become impossible, or the number of total possible vertices is less than 52, which cannot make a 85-order graph anymore.

Now we look back to the vertices in the segment triple T.

Theorem 6.2.2. For a pair of vertices v_1 and v_2 in T that are from different segments, their common neighbours that are not in T should meet the condition in Theorem 6.1.1.

Remark. Recall that any two non-adjacent vertices have two common neighbours and two adjacent vertices have three common neighbours in Γ. For each segment triple T with matchings, let v_1 and v_2 be two vertices. Then the number of common neighbours of v_1 and v_2 in $\Gamma \setminus T$ are already determined.

Therefore, we plan to go over all such pair of vertices and check whether they can have enough common neighbours after the previous check. Once there is a pair of vertices cannot be satisfied, it means that the whole structure, i.e. the segment triple with matchings between them, is impossible.

6.3 Ways to add more vertices

By gluing the segment triple structure, we already have a Gram matrix for about dimension 30: there are 33 vertices in total, and by Lemma 4.1.2, we do not have to consider

the vertices A, B, and C. As a result of computation, we noticed that almost all Gram matrices at this point have rank 29 or 30.

Noticing that the eigenspace U_θ with respect to the eigenvalue $\theta = 4$ has dimension 34. If we can add a few more vertices that are linearly independent with the vertices we already have, then the case will reach a contradiction soon.

We have introduced how we justify whether two vertices can be neighbours or non-neighbours in Theorem 6.2.1. For the 30 vertices in the segment triple T, they have some neighbours not in $\hat{T}$. Now let us focus on these vertices.

Firstly, we compute the sets of possible neighbours for these further vertices, as mentioned in the remark of Theorem 6.2.2. These sets are recorded as `downs` in our code. Each record contains the possible vertex delineate by its neighbours in the segment triple, without matchings yet.

Secondly, for each segment triple that passes the check after gluing and matchings, we go over all vertices in `downs`. In order to add one vertex v from `downs` to T, the vertex v should satisfy the following condition:

Lemma 6.3.1. For any vertices t_1 and t_2 in T, after adding the vertex v to T, t_1 and t_2 should have no more than three common neighbours if they are adjacent, or no more than two common neighbours if they are not adjacent.

Proof. Clearly these vertices are part of graph Γ and should satisfy the parameter conditions. □

If adding v leads to a contradiction with Lemma 6.3.1, then v is no longer possible. All vertices in `downs` that are still possible in case of segment triple T, will be recorded in `verts`, together with its neighbours and the adjacency vector.

Now we are ready for the iterative part of the level 3 check. Such things will be explained in the following sections. We first set up some basic theory in linear algebra.

6.4 Gram-Schmidt process

Theorem 6.4.1. (Gram-Schmidt process)

Suppose $v_1, v_2, ..., v_n$ are linearly independent vectors in a vector space V. Define $e_1 = v_1$, and define $e_2, ..., e_n$ inductively as following:

$$e_i = v_i - \frac{\langle v_i, e_1 \rangle}{\langle e_1, e_1 \rangle} e_1 - \cdots - \frac{\langle v_i, e_{i-1} \rangle}{\langle e_{i-1}, e_{i-1} \rangle} e_{i-1}.$$

Then $e_1, e_2, ..., e_n$ are orthogonal vectors in V, and

$$\text{span}\{v_1, v_2, ..., v_m\} = \text{span}\{e_1, e_2, ..., e_m\}$$

for all $m = 1, 2, ..., n$.

Proof. We will prove this theorem by induction on i. Starting with the case $i = 1$, $\text{span}\{v_1\} = \text{span}\{e_1\}$ is trivial.

Now suppose that the statement holds for all $1 \leq k < i$, we need to show that $\text{span}\{v_1, v_2, ..., v_i\} = \text{span}\{e_1, e_2, ..., e_i\}$ also holds.

For a fixed $1 \leq k < i$, we have:

$$\begin{aligned} \langle e_i, e_k \rangle &= \left\langle v_i - \frac{\langle v_i, e_1 \rangle}{\langle e_1, e_1 \rangle} e_1 - \cdots - \frac{\langle v_i, e_{i-1} \rangle}{\langle e_{i-1}, e_{i-1} \rangle} e_{i-1}, e_k \right\rangle \\ &= \langle v_i, e_k \rangle - \langle v_i, e_k \rangle \\ &= 0. \end{aligned}$$

By arguments above we claim that $e_1, e_2, ..., e_i$ forms a set of orthogonal vectors.

What's more, since e_i is a linear combination of $e_1, e_2, ..., e_i$, we know the fact that $\text{span}\{v_1, v_2, ..., v_i\} \subset \text{span}\{e_1, e_2, ..., e_i\}$. On the other hand, both sets are linearly independent by our hypothesis and the property that they are orthogonal, hence they are equal. □

Remark. In general, the standard Gram-Schmidt process will get a orthonormal basis,

while in Theorem 6.4.1 we do not require normality, since we do not have to normalize all the vectors in our following computation.

When realizing the Gram-Schmidt process in our problem, cases we are facing may not be a positive definite matrix but a semi-positive definite matrix, i.e. some of the vectors may be linearly dependent. To make our theory more complete, let U_θ be the eigenspace spanned by the vertex vectors $v_1, ..., v_n$, equipped with the bilinear form $\langle x, y\rangle = x \operatorname{Gr} y^T$ where Gr is the Gram matrix. And let U' be a vector space of dimension n, with basis $\{v'_1, ..., v'_n\}$. We have the following conclusion:

Theorem 6.4.2. Applying the Gram-Schmidt process on $U' = \operatorname{span}\{v'_1, ..., v'_n\}$ with respect to the bilinear form $\langle x, y\rangle = x \operatorname{Gr} y^T$ will create an orthogonal basis and zero vectors.

Proof. We will just apply the same computation as in Theorem 6.4.1 if the vector v_i is linearly independent with $v_1, ..., v_{i-1}$. If we can complete the Gram-Schmidt process until the end then the theroem is automatically proved. Otherwise, suppose that the first vector that is linearly dependent with the previous ones is v_j, then the corresponding e_j we get from Gram-Schmidt process will be a zero vector. In such case, e_j will not involved in the further computation, since we don't want a $\langle e_j, e_j\rangle = 0$ appears in the denominator.

By the Gram-Schmidt process, all the non-zero vectors we get are orthogonal. □

Lemma 6.4.3. Define a homomorphisim $\phi : U' \to U, v'_i \mapsto v_i$, and equip U' with the same bilinear form $\langle \cdot, \cdot\rangle$. Let $K = \operatorname{Ker}\phi$. Suppose $U' \subset W$ and W is equipped with a bilinear form $\langle \cdot, \cdot\rangle$, whose restrict onto U' is $\langle x, y\rangle = x \operatorname{Gr} y^T$. If there exists $w \in W$ such that $\langle w, k\rangle \neq 0$ for some $k \in K$ then the bilinear form on W is not semi-positive definite.

Proof. We will prove this by contradiction. Suppose that the bilinear form on W is semi-positive definite, then since $k \in \operatorname{Ker}\phi$ we have $\langle k, k\rangle = \langle k, k\rangle|_{U'} = 0$, therefore $k \in W^\perp$, but then $\langle w, k\rangle = 0$, a contradiction. □

Theorem 6.4.2 and Lemma 6.4.3 provide us the theory for future when we are adding vertices to the graph structure we already have.

6.5 How to find the projection vector

We start with the relation between the Gram-Schmidt process and the LDLT decomposition.

Theorem 6.5.1. Let G be a matrix supporting bilinear form on an n-dim vector space. Let R be the matrix formed by the vectors we get from the Gram-Schmidt process. Let $G = LDL^T$ be the LDLT decomposition of G. Then we have $R = L^{-1}$.

Proof. Consider the inner product $\langle x, y\rangle = x^T G y$, which will be a trivial dot product when G is the identity matrix I_n. In our computation, since we require the Gram matrix G be semi-positive-definite, we can apply the Gram-Schmidt process (Theorem 6.4.1) and change the usual basis $\{e_1, e_2, ..., e_n\}$ into $\{v_1, v_2, ..., v_n\}$ which is orthogonal with respect to the inner product $\langle \cdot, \cdot \rangle$.

The change-of-basis matrix with respect to this process will be a lower uni-triangular matrix L_1, the matrix corresponding to $\langle x, y\rangle = x^T G y$ under the basis $\{v_1, v_2, ..., v_n\}$ will be a diagonal matrix D_1. This will give us $G = L_1 D L_1^T$, which give us the same result as the LDLT decomposition in Section 4.4.1.

Therefore, we have $R = L^{-1}$. □

This theorem provides us a way to compute the projection matrix of a vector corresponding to a vertex over the space spanned by some other vectors, and we do not have to solve a linear system, which costs more time. Detailed method will be introduced in the following section.

One more thing to mention is that since our matrix is semi-positive definite, there are possibilities that for some index i, $D_{ii} = 0$. In which case, it means that this row G_i is linearly dependent to other rows, and hence we will get a zero vector R_i during the Gram-Schmidt process.

It remains a problem that how we find the projection vector of a vertex onto the space spanned by the vertices in the segment triple. In this section, we will show our computation.

Theorem 6.5.2. Let U be spanned by a segment triple $T = \{v_1, ..., v_{30}\}$. For any vertex d in the set `downs`, the projection of the vector d onto U is:

$$d_U = r \sum_{i=1}^{n} \frac{R_i^T R_i}{D_{ii}}, \tag{6.1}$$

where $r = (r_1, r_2, ..., r_{30})$ is the row vector defined as:

$$r_i = \begin{cases} \dfrac{2}{7}, & \text{if } v_i \sim d, \\ -\dfrac{1}{14}, & \text{if } v_i \not\sim d, \end{cases}$$

R_i are the row vectors we get from the Gram-Schmidt process, and D_{ii} are the entries of D that we get from the LDLT decomposition.

Proof. Let Gr be the Gram matrix generated by the vectors of $v_1, ..., v_{30}$. We will prove this by computing directly:

$$\begin{aligned} d_U &= \sum_{i=1}^{n} \frac{\langle d_U, R_i \rangle}{\langle R_i, R_i \rangle} R_i \\ &= \sum_{i=1}^{n} \frac{\langle d_U, R_i \rangle}{D_{ii}} R_i \\ &= \sum_{i=1}^{n} d_U \,\mathrm{Gr}\, R_i^T \frac{R_i}{D_{ii}} \\ &= d_U \,\mathrm{Gr} \left(\sum_{i=1}^{n} \frac{R_j^T R_j}{D_{ii}} \right) \\ &= r \sum_{i=1}^{n} \frac{R_i^T R_i}{D_{ii}}. \end{aligned}$$

□

Remark. In (6.1), we drop the terms that $D_{ii} = 0$, which will also lead R_i to be a zero vector. Therefore the formulae is well defined and will not cause any confusion.

6.6 Implementation of the Step 3 check

We first have a brief overview on what we are doing. As an overview, this Step 3 check is an iteration that will stop only when we find a contradiction, or a possible case.

We pick a vertex in the segment triple T and trying to find its neighbours. The searching will go along the set `verts`. Newly added vertices must be compatible with all existing vertices and not exceed the maximum number of common neighbours for each vertex pair found so far. Currently, we select the vertex number 13. Vertex number 13 has the advantage of being one of the vertices in the gluing handle between Σ_B and Σ_C. Since vertex number 13 and 14 are not adjacent, they each have two neighbours in Σ_B and Σ_C. Adding vertex B and C gives us a total of six neighbours. In addition, since vertex number 13 and 14 are not adjacent, they cannot have any more common neighbours. These analyses of the structure provide us with numerous conditions to eliminate certain cases. In the remainder of this chapter, we call this vertex v_S.

Meanwhile, we create an array named `comm` to record the number of required common neighbours for all vertex pairs within the segment triple T.

We start with the following theorem:

Theorem 6.6.1. Let v be a vertex in the set `verts`, if any two neighbours of v in T are already presented that require no further common neighbours in `comm`, then the vertex is automatically impossible.

Next, all possible vertices remaining in `verts` have neighbours in T. This implies that every possible vertex v in `verts` will contribute to some pairs of vertices (t_1, t_2) in T which still need more common neighbours. If any pair of vertices (t_1, t_2) still fails to satisfy the common neighbours condition after adding all their common neighbours in `verts`, then such segment triple T is impossible.

If there are any pair (t_1, t_2) that have just enough common neighbours after adding all their common neighbours in `verts`, we are forced to add these new vertices to the current structure. Otherwise, there would not be enough common neighbours for some pairs. Let

V_f be the set of all forced vertices. For vertices in V_f, two checks are waiting for them.

Firstly, the projection of the vector corresponding to a vertex $v_f \in V_f$ should have a length of less than 1 when projected onto the space U and $U_\perp$. Secondly, if $|V_f| > 1$, then vertices in V_f should be pairwise compatible, following by the condition in Theorem 6.2.1. Failing in either of these two checks will result in the conclusion that this segment triple T is impossible.

For all segment triples together with forced vertices that passes the check above, we will try to add some more vertices to them. But before we move forward, we have to look up some things:

Lemma 6.6.2. The vertices in V_f should not make any pair of vertices in T exceed the maximum number of common neighbours. Besides, any possible vertex in `verts` that are not compatible with any vertex in V_f are no longer possible in this case.

Lemma 6.6.3. If the number of vertices in V_f and vertices in $\hat{T}$ exceeds 85, or the number of remaining possible vertices together with vertices in $V_f \cup \hat{T}$ is less than 85, then this segment triple T is impossible.

Proof. Clearly, insufficient or excessive quantities of vertices will lead to contradictions. □

Now let us turn our attention to the vertices that are possible extra vertices but not the forced ones. To begin with, we do a search along the set `verts`.

Theorem 6.6.4. Let v_v be a vertex in `verts`. Let (x_a, y_a), (x_b, y_b), and (x_c, y_c) be the three pairs of neighbours of it in T corresponding to each segment, and denote the set formed by these vertices as $N_T(v_v)$. Then if any pair of vertices in $N_T(v_v)$ are already has enough common neighbours, i.e. have three common neighbours if they are adjacent or have two common neighbours if they are not adjacent, then v_v is no longer possible.

If no more vertices in `verts` are possible, we check that if we have got exactly 85 vertices in total. If so, then we found a possible case waiting for further checks, otherwise there is a contradiction.

In the case that there are still possible vertices in the set `verts`, we check that whether there are enough common neighbours between v_S and the other vertices in T. If all pairs have enough common neighbours, then adding all forced vertices must give us 85 vertices in total, or a contradiction. If not, among the vertices that still need common neighbours with the picked one, we select the one with the least requirement. Specifically, we select the one that also has the smallest ordinal, and call it v_J (since this vertex is denoted as j in our code).

We aim to add all the common neighbours of v_S and v_J. Now we again go back to search the vertices in the set `verts`. For those in the set `verts` that are also adjacent with v_J, we may be forced to add vertices if any of them are really forced, as what we have done above; or, we assume that a vertex is forced, and check if any contradiction arises.

As an iteration step, these checks will repeat until we find enough neighbours of v_S, or we find a contradiction.

The number of extra vertices that are still needed is computed as following:

Lemma 6.6.5. If v_S is the vertex number 13 then we need 44 extra vertices.

Proof. For the vertex number 13, it has 6 neighbours in the segment triple as we discussed before, so it needs $8(= 14 - 6)$ more neighbours. After we found all eight neighbours of it, there are still $44= (85 - 33 - 8)$ vertices needed. □

In order to check whether a pair of vertices can be adjacent or non-adjacent, we need to apply the conclusion of Theorem 6.2.1. The projection of vectors to the space U spanned by the 30 vertices can be computed by Theorem 6.5.2 and equation (6.1).

6.7 The last cases

For all the neighbours we found of v_S, they are recorded as `further` in our code. While we were checking these vertices, we only checked whether they are pairwise compatible. For

those cases where we find exactly eight vertices in `further` which pass all initial checks, we apply the final checks to eliminate them. In the programme it is called Step 4.

Theorem 6.7.1. The induced subgraph on all the neighbours of v_S, including those in T and those we just found, is a cubic graph that satisfies Theorem 3.2.1.

Proof. Clearly, this induced subgraph is the local graph of v_S, so it should have all the properties in Theorem 3.2.1. □

We denote this induced subgraph as $N(v_S)$. Let V_s be the set of vertices in $N(v_S)$ that are in the segment triple T, and V_n be the set of vertices not in T.

For $N(v_S)$, we know all the edges inside V_s and the edges between V_s and V_n. We don't know the edges between vertices in V_n exactly. However, we know some of the possibilities while checking the compatibility, as shown in Theorem 6.2.1.

There are many pairs of vertices that only satisfy one of the two conditions in Theorem 6.2.1, which means they have been determined to be adjacent or not. Also there are some pairs that satisfy both conditions, hence there can be an edge or an non-edge between them, which are the things we need to justify right now. Before going further, we first record all forced edges and non-edges, and for those that can be either an edge or an non-edge, we call them 'maybe'.

Together with the information of edges between V_s and V_n, for each vertex in V_n we can compute the number of neighbours it still needs, we call it demand. Besides, for each vertex, we call the number of possible edges it can have as supply. We first have some very simple way to eliminate cases:

Lemma 6.7.2. There are two cases that are obvious:

1. A vertex that has negative demand is impossible.

2. A vertex that has demand greater than supply is impossible.

After we remove these two obviously impossible cases, the structure of V_n is determined with reasonable demands and supplies. In order to justify whether each 'maybe' should

be an edge or a non-edge, we need to apply further checks. Firstly, we notice the following facts:

Lemma 6.7.3. If a vertex has demand equal to supply then all the 'maybe's should be edges and then all the incident edges are determined.

After that, for the graph $N(v_S)$ we check the properties mentioned in Theorem 3.2.1. The following lemmas show the criteria we are using:

Lemma 6.7.4. We have two easy cases: a vertex has all edges already, or it need all 'maybe's:

1. If any vertex v already has three neighbours, then all other 'maybe's incident with v should be non-edges. After such change, if any vertex has demand greater than supply, then this case is impossible.

2. If the demand of a vertex v is equal to the number of 'maybe's, then all these 'maybe's will be edges. After such change, if v or any vertex adjacent to v has a negative demand, then this case is impossible.

Lemma 6.7.5. After that, we need to check the number of common neighbours for each pair of vertices:

1. After applying the above changes, for a survived case, if any two vertices v_1, v_2 have more than two common neighbours in $N(v_S)$, then this case is impossible.

2. If any two vertices v_1, v_2 have two common neighbours in $N(v_S)$, then they must be adjacent. If there is a non-edge between v_1, v_2 then this case is impossible. Else if it is 'maybe' between v_1, v_2 then it should be an edge. If such change cause either v_1 or v_2 has a negative demand, then this case is impossible.

3. For any two adjacent vertices v_1, v_2 with two common neighbours, they should not have any other common neighbours. Suppose v_3 is adjacent to v_1, then v_2 should not be adjacent to v_3, and vice versa. If such change leads to the supply is less than the demand of a vertex, then this case is impossible.

4. For any two non-adjacent vertices v_1, v_2, if they already have one common neighbour in $N(v_S)$, they cannot have more. Therefore, suppose v_3 is adjacent to v_1, then v_2 should not be adjacent to v_3, and vice versa. If such change leads to the supply is less than the demand of a vertex, then this case is impossible.

Proof. The first case is using the fact that any two vertices in the local graph can have at most two common neighbours, according to Lemma 3.2.1.

The second and the third cases are also an application of Lemma 3.2.1, that any two vertices in a local graph that have two common neighbours must be adjacent, and vice versa.

The last case is that any two non-adjacent vertices in a local graph have exactly one common neighbour. □

Furthermore, if there are still some pairs of vertices in `further` that can be either adjacent or non-adjacent, then we just check both cases under the same criteria as above.

This iterative check does not take a lot time and kills all cases as we observed.

This check resolves 'maybe's. Now if a case passes the above check, the structure of V_n is fully determined. In other words, we now know exactly the edges in V_n.

After this fast combinatorial check on the adjacency condition, if there are any case still alive, we check whether the Gram matrix is semi-positive definite for all the remaining cases, and also the total dimension of such matrix which should not exceed 34. According to our current result, all the cases we computed are eliminated now.

CONCLUSION

Currently our programme is still running on the computer on multiple processors to see how many cases survived after these four steps. Up to now, all cases are eliminated after the last check.

We think that the methods we used in this thesis will also be useful in other undetermined cases of strongly regular graphs in [6], especially for those cases with small λ and μ. These cases will have much stronger common neighbours condition. Besides, the semi-positive definite check on the Gram matrices requires the dimension of the corresponding eigenspace not very large.

APPENDIX A

GAP CODE

A.1 The LDLT decomposition

```
LDLTDecomposition:=function(A)
 local B,n,L,D,i,j,k,sum;
 B:=ShallowCopy(A);
 n:=Size(B);
 L:=NullMat(n,n);
 D:=NullMat(n,n);

for i in [1..n] do
 sum:=[];
 for j in [1..i-1] do
  Add(sum,L[i][j]*L[i][j]*D[j][j]);
 od;

 D[i][i]:=B[i][i]-Sum(sum);

 if D[i][i]=0 then
  for j in [i+1..n] do
```

```
   sum:=[];
   for k in [1..i-1] do
    Add(sum,L[j][k]*L[i][k]*D[k][k]);
   od;
   if B[j][i]-Sum(sum)=0 then
    L[j][i]:=0;
   else
    return false;
   fi;
  od;
  L[i][i]:=1;
 else
  for j in [i+1..n] do
   sum:=[];
   for k in [1..i-1] do
    Add(sum,L[j][k]*L[i][k]*D[k][k]);
   od;
   L[j][i]:=(B[j][i]-Sum(sum))/D[i][i];
  od;
  L[i][i] := 1;
 fi;
od;

return [L,D];
end;;
```

A.2 The AddOne function

```
G:=List([1..30],i->List([1..30],j->0));
L:=List([1..40],i->List([1..40],j->0));
D:=[];

AddOne:=function(r)
  local n,i,sum,j;
  n:=Length(r);

  for i in [1..n] do
    sum:=0;
    for j in [1..i-1] do
      sum:=sum+L[n][j]*L[i][j]*D[j];
    od;
    if i<n then
      if D[i]=0 then
        if r[i]=sum then
          L[n][i]:=0;
        else
          return false;
        fi;
      else
        L[n][i]:=(r[i]-sum)/D[i];
      fi;
    else
      L[n][n]:=1;
      D[n]:=r[n]-sum;
      if D[n]<0 then
```

```
        return false;
      fi;
    fi;
  od;
  if n<=30 then
    for i in [1..n] do
      G[n][i]:=r[i];
      G[i][n]:=r[i];
    od;
  fi;
  return true;
end;
```

A.3 Select the segment from each goodgraph

```
segment:=[];
old:=100;
for n in [1..Length(goodgraphs)] do
   G:=goodgraphs[n];
   A:=AutomorphismGroup(G);
   edges:=[];
   for a in [1..14] do
     for b in Adjacency(G,a) do
       if Intersection(Adjacency(G,a),Adjacency(G,b))=[] then
         Add(edges,[a,b]);
       fi;
     od;
   od;
```

```
orbs:=Orbits(A,edges,OnPairs);
for o in orbs do
  replace:=false;
  e:=o[1];
  a:=e[1];
  b:=e[2];

  aa:=Set(Difference(Adjacency(G,a),[b]));
  bb:=Set(Difference(Adjacency(G,b),[a]));

  if aa[2] in Adjacency(G,aa[1]) then
    ta:=6;
  else
    ta:=4;
  fi;
  if bb[2] in Adjacency(G,bb[1]) then
    tb:=6;
  else
    tb:=4;
  fi;

  if 10*ta+tb<=old or Position(orbs,o)=1 then
    replace:=true;
    old:=10*ta+tb;
  fi;

  if replace then
    segment[n]:=rec(graph=n,edge:=e,type:=10*ta+tb);
```

```
    fi;
  od;
od;
```

A.4 Building the enumeration tree

```
Make_Tree:=function(n)
  local tree,newlevel;
  tree:=[];

  newlevel:=function(sg)
    local k,s,c,i,m;
    k:=Length(sg)+1;
    s:=Length(tree);
    c:=0;
    for i in [1..n] do
      if not (i in sg) then
        m:=Length(tree)+1;
        Add(tree,rec(k:=k,m:=i,bro:=0,son:=0));
        if c<>0 then
          tree[c].bro:=m;
        fi;
        if k<n then
          tree[m].son:=m+1;
          newlevel(Concatenation(sg,[i]));
        fi;
        c:=m;
      fi;
```

```
    od;
  end;

  newlevel([]);
  return tree;
end;
```

A.5 Building the big enumeration tree

```
Make_Big_Tree:=function(none,right,left,both)
  local n,L,R,tree,biglevel;
  n:=none+right+left+both;
  L:=left+both;
  R:=right+both;
  tree:=[];

  # biglevel

  biglevel:=function(k,sg,tau)
    local s,c,i,j,m;
    s:=Length(tree);
    c:=0;
    if k<=none then
      Add(tree,rec(k:=k,l:=0,r:=0,bro:=0,son:=s+2));
      biglevel(k+1,sg,tau);
    elif k<=none+right then
      for i in [1..R] do
        if not (i in tau) then
```

```
      m:=Length(tree)+1;
      Add(tree,rec(k:=k,l:=0,r:=i,bro:=0,son:=0));
      if c<>0 then
        tree[c].bro:=m;
      fi;
      if k<n then
        tree[m].son:=m+1;
        biglevel(k+1,sg,Concatenation(tau,[i]));
      fi;
      c:=m;
    fi;
  od;
elif k<=none+right+left then
  for j in [1..L] do
    if not (j in sg) then
      m:=Length(tree)+1;
      Add(tree,rec(k:=k,l:=j,r:=0,bro:=0,son:=0));
      if c<>0 then
        tree[c].bro:=m;
      fi;
      if k<n then
        tree[m].son:=m+1;
        biglevel(k+1,Concatenation(sg,[j]),tau);
      fi;
      c:=m;
    fi;
  od;
else
```

```
      for i in [1..L] do
        if not (i in sg) then
          for j in [1..R] do
            if not (j in tau) then
              m:=Length(tree)+1;
              Add(tree,rec(k:=k,l:=i,r:=j,bro:=0,son:=0));
              if c<>0 then
                tree[c].bro:=m;
              fi;
              if k<n then
                tree[m].son:=m+1;
                biglevel(k+1,Concatenation(sg,[i]),
                             Concatenation(tau,[j]));
              fi;
              c:=m;
            fi;
          od;
        fi;
      od;
    fi;
  end;

  biglevel(1,[],[]);
  return tree;
end;


shapes:=Cartesian([0..8],[0..8],[0..8],[0..8]);
```

```
shapes:=Filtered(shapes,s->Sum(s)=8);
shapes:=Filtered(shapes,s->s[2]<=s[3]);
shapes:=Filtered(shapes,s->s[2]+s[4] in [4,6]);
shapes:=Filtered(shapes,s->s[3]+s[4] in [4,6]);

code:=function(none,right,left,both)
  return 1000*none+100*right+10*left+both;
end;

big_trees:=[];
for s in shapes do
  none:=s[1];
  right:=s[2];
  left:=s[3];
  both:=s[4];
  big_trees[code(none,right,left,both)]:=
       Make_Big_Tree(none,right,left,both);
od;
```

A.6 The main enumeration function

```
#
# The main enumeration part
#

Reset(GlobalMersenneTwister,CurrentDateTimeString());; # randomize GAP

#
```

```
# Global variables
#

  # hard to eliminate cases counters
  # lev2alive: survivors of level 2 checks for the given pair
  # lev3alive: survivors of level 3 checks for the pair
  # exact: where we found exactly 85 vertex sets
    lev2alive:=0;
    lev3alive:=0;
    exact:=[];

  # begt, endt: time monitoring variables
    begt:=0;
    endt:=0;

  # print_string: array of what we print for each harder case
  # PrintCase: print function
    print_string:=[];
    PrintCase:=function()
      Print("\r
      Print("\r",print_string[1],print_string[2],print_string[3],print_string[4],"\
    end;

#
#

  # p: segment pair number
    p:=0;
```

```
# n: first segment number
# k: second segment number
# first: historic first segment
  n:=0;
  k:=0;
  first:=0;

# im: how the pair is glued together ([1,2] or [2,1])
  im:=[];

# s: first segment
# t: second segment
  s:=rec();
  t:=rec();

# match: size of matching between s and t
  match:=0;

# tree: tree of matchings between s and t
# u: node in tree
  tree:=[];
  u:=0;

# segment_triples: list of triples
# z: pair complement number
# w: third segment
  segment_triples:=[];
```

```
  z:=0;
  w:=rec();

# big_tree: tree of matchings between w and s plus t
# v: node in big_tree
  big_tree:=[];
  v:=0;

# Gram matrix and its decomposition
# G,L,D (such that G=LDL^t) in the list form, R=L^-1
#
# P the projection matrix
  G:=List([1..30],i->List([1..30],j->0));
  L:=List([1..30],i->List([1..30],j->0));
  D:=[];
  R:=List([1..30],i->List([1..30],j->0));

  P:=List([1..30],i->List([1..30],j->0));

# lists of vertices of the three components
  seg1:=[1..12];
  seg2:=[1,2,13,14,15,16,17,18,19,20,21,22];
  seg3:=[3,4,13,14,23,24,25,26,27,28,29,30];

# downs: precomputed sets of possible neighbours for further vertices
  downs:=[];
  for p1 in Combinations(seg1,2) do
    for p2 in Combinations(Difference(seg2,[13]),1) do
```

```
        if Intersection(p1,[1,2])=Intersection(p2,[1,2]) then
          for p3 in Combinations(Difference(seg3,[13]),1) do
            if Intersection(p1,[3,4])=Intersection(p3,[3,4]) and
               Intersection(p2,[14])=Intersection(p3,[14]) then
              Add(downs,rec(neibs:=Union(p1,p2,p3)));
            fi;
          od;
        fi;
      od;
    od;

  # forced: list of forced vertices
  # nfor: current number of forced vertices
    forced:=[];
    nfor:=0;

# loading segment data

Print("Loading segment data...\n");

if not IsBound(segment_data) then
  segments:=[];
  Read("data/segment_data.g");
fi;

Print("...done\n");

# loading doubles
```

```
Print("Loading segment pairs...\n");

if not IsBound(segment_pairs) then
  segment_pairs:=[];
  Read("data/segment_pairs.g");
fi;

Print("...done\n");

# loading the LDLT function AddOne

Print("Loading the AddOne function...\n");

AddOne:=function(r)
  return 1;
end;

Read("addone.g");

Print("...done\n");

#
# Vertex projection
#

ComputeProjMat:=function()
  local i;
```

```
  P:=List([1..30],i->List([1..30],j->0));
  for i in [1..30] do
    if D[i]<>0 then
      P:=P+TransposedMat([R[i]])*[R[i]]/D[i];
    fi;
  od;
end;

IsGoodVert:=function(d)
  local i,proj;
  for i in [1..30] do
    if D[i]=0 then
      if d.r*R[i]<>0 then
        return false;
      fi;
    fi;
  od;
  proj:=d.r*P;
  if d.r*proj>1 then
    return false;
  fi;
  d.proj:=proj;
  return true;
end;

#
# triple removed
#
```

```
NoTripleRemoved:=function(d)
  local s,i,j,n,h;
  s:=ShallowCopy(d.neibs);
  Add(s,13);
  h:=[];
  for i in [1..30] do
    if i in s then
      continue;
    fi;
    n:=0;
    for j in s do
      if G[i][j]=2/7 then
        n:=n+1;
      fi;
    od;
    if n>=3 then
      return false;
    elif n=2 then
      Add(h,i);
    fi;
  od;
  d.h:=h;
  return true;
end;

#
# Check vertices for compatibility
```

```
#

AreGoodNeighbours:=function(d,e)
  if Length(Intersection(d.neibs,e.neibs))>2 then
    return false;
  fi;
  if Length(Intersection(d.neibs,e.h))<>0 or
     Length(Intersection(d.h,e.neibs))<>0 then
    return false;
  fi;
  if (2/7-d.r*e.proj)^2>(1-d.r*d.proj)*(1-e.r*e.proj) then
    return false;
  fi;
  return true;
end;

AreGoodNonNeighbours:=function(d,e)
  if Length(Intersection(d.neibs,e.neibs))>1 then
    return false;
  fi;
  if (-1/14-d.r*e.proj)^2>(1-d.r*d.proj)*(1-e.r*e.proj) then
    return false;
  fi;
  return true;
end;

AreCompatibleVerts:=function(d,e)
  return AreGoodNeighbours(d,e) or AreGoodNonNeighbours(d,e);
```

```
end;

#
# The iterative part of Step 3
#

# number of missing neighbours of 13
tot:=8;

# vertices that are not 13
others:=Difference([1..30],[13]);

Step3Iterator:=function(verts,comm)
  local i,j,good,nverts,tverts,ncomm,tcomm,d,e,q,new,add;

  nverts:=StructuralCopy(verts);
  for d in nverts do
    if IsBound(d.used) then
      continue;
    fi;
    for i in d.neibs do
      if comm[i]=0 then
        d.used:=true;
      fi;
    od;
  od;

  ncomm:=StructuralCopy(comm);
```

```
  for i in others do
    if ncomm[i]=0 then
      ncomm[i]:=-1000000;
    fi;
  od;

  tcomm:=StructuralCopy(ncomm);

  for d in nverts do
    if IsBound(d.used) then
      continue;
    fi;
    for i in d.neibs do
      tcomm[i]:=tcomm[i]-1;
    od;
  od;

  new:=false;
  for i in others do
    if tcomm[i]>0 then
#      Print("Insufficient verts\n");
      return false;
    elif tcomm[i]=0 then
      new:=true;
    fi;
  od;
```

```
  if new then
    add:=[];
    for d in nverts do
      if IsBound(d.used) then
        continue;
      fi;
      for i in d.neibs do
        if tcomm[i]=0 then
          Add(add,d);
        fi;
      od;
    od;

    if nfor+Length(add)>tot then
#      Print("Too many forced\n");
      return false;
    fi;

#    Print("Number of additional forced vertices is ",Length(add),"\n");

    for i in [1..Length(add)-1] do
      for j in [i+1..Length(add)] do
        if not AreCompatibleVerts(add[i],add[j]) then
#          Print("Incompatible forced verts\n");
          return false;
        fi;
      od;
    od;
```

```
tverts:=[];
for d in nverts do
  if IsBound(d.used) then
    continue;
  fi;
  if d in add then
    continue;
  fi;

  good:=true;
  for e in add do
    if not AreCompatibleVerts(d,e) then
      good:=false;
      break;
    fi;
  od;
  if good then
    Add(tverts,d);
  fi;
od;

if Length(tverts)=0 then
  if Length(add)+nfor=tot then
    Add(exact,[n,k,u,z,v,StructuralCopy(forced)]);
  else
    return false;
  fi;
```

```
    fi;

    tcomm:=StructuralCopy(ncomm);
    for d in add do
      for i in d.neibs do
        tcomm[i]:=tcomm[i]-1;
      od;
    od;

    for i in others do
      if tcomm[i]<0 and tcomm[i]>-1000000 then
#        Print("Excess in forced\n");
        return false;
      fi;
    od;

    forced:=Concatenation(forced,add);
    nfor:=nfor+Length(add);

    good:=Step3Iterator(tverts,tcomm);
    nfor:=nfor-Length(add);
    forced:=forced{[1..nfor]};
    return good;
  else
    good:=false;
    for e in verts do
      if IsBound(e.used) then
        continue;
```

```
      else
        good:=true;
        break;
      fi;
    od;
    if not good then
      if nfor=tot then
        Add(exact,[n,k,u,z,v,StructuralCopy(forced)]);
        return true;
      else
#        Print("Non-exact\n");
        return false;
      fi;
    fi;

    j:=0;
    for i in others do
      if comm[i]<0 then
        continue;
      fi;
      if j=0 then
        j:=i;
      fi;
      if comm[i]<comm[j] then
        j:=i;
      fi;
      if comm[j]=1 then
        break;
```

```
      fi;
    od;
    if j=0 then
      if Length(forced)=tot then
        Add(exact,[n,k,u,z,v,StructuralCopy(forced)]);
        return true;
      else
#        Print("Inexact\n");
        return false;
      fi;
    fi;

    good:=false;
    for e in nverts do
      if IsBound(e.used) then
        continue;
      fi;
      if not (j in e.neibs) then
        continue;
      fi;

#     Print("Force a vert\n");

      tverts:=[];
      for d in nverts do
        if IsBound(d.used) then
          continue;
        fi;
```

```
  if d=e then
    continue;
  fi;

  if AreCompatibleVerts(d,e) then
    Add(tverts,d);
  fi;
od;

if Length(tverts)=0 then
  if nfor=tot-1 then
    Add(exact,[n,k,u,z,v,StructuralCopy(forced)]);
  else
    continue;
  fi;
fi;

tcomm:=StructuralCopy(ncomm);
for i in e.neibs do
  tcomm[i]:=tcomm[i]-1;
od;

e.used:=true;
Add(forced,e);
nfor:=nfor+1;

good:=good or Step3Iterator(tverts,tcomm);
```

```
      Unbind(forced[nfor]);
      nfor:=nfor-1;
    od;

    return good;
  fi;
end;

#
# Step 3 preparation
#

all:=[];

Step3:=function()
  local comm,i,j,ij,verts,d,e,good,p,r,caught;

  lev2alive:=lev2alive+1;

  ComputeProjMat();

  all:=List([1..29],i->[]);

  for i in [1..29] do
    for j in [i+1..30] do
      if G[i][j]=2/7 then
        all[i][j]:=3;
      else
```

```
      all[i][j]:=2;
    fi;
    for ij in [1..30] do
      if G[i][ij]=2/7 and G[j][ij]=2/7 then
        all[i][j]:=all[i][j]-1;
      fi;
    od;
    if IsSubset(seg1,[i,j]) then
      all[i][j]:=all[i][j]-1;
    fi;
    if IsSubset(seg2,[i,j]) then
      all[i][j]:=all[i][j]-1;
    fi;
    if IsSubset(seg3,[i,j]) then
      all[i][j]:=all[i][j]-1;
    fi;
    if all[i][j]<0 then
      Print("Too many neighbours\n");
      Sleep(5);
      return;
    fi;
  od;
od;

comm:=[];
for i in [1..12] do
  comm[i]:=all[i][13];
od;
```

```
for i in [14..30] do
  comm[i]:=all[13][i];
od;
verts:=[];

caught:=0;

for d in downs do
  e:=StructuralCopy(d);

  good:=true;
  for i in e.neibs do
    if comm[i]=0 then
      good:=false;
      break;
    fi;
  od;
  if not good then
    continue;
  fi;
  for p in Combinations(Union(e.neibs,[13]),2) do
    if all[p[1]][p[2]]=0 then
      good:=false;
      break;
    fi;
  od;
  if not good then
    continue;
```

```
    fi;

#    if not NoTripleRemoved(e) then
#      continue;
#    fi;

    e.r:=List([1..30],i->-1/14);
    e.r[13]:=2/7;
    for i in e.neibs do
      e.r[i]:=2/7;
    od;
    if IsGoodVert(e) then
     if not NoTripleRemoved(e) then
       caught:=caught+1;
       continue;
     fi;
     Add(verts,e);
    fi;
  od;

#  Print("--- ",Length(verts)," verts\n");
  if Length(verts)>=150 then
    print_string[4]:=Concatenation(" (",String(Length(verts))," verts)");
    PrintCase();
  fi;

  for i in others do
    if comm[i]=0 then
```

```
      comm[i]:=-1000000;
    fi;
  od;

  good:=Step3Iterator(verts,comm);
  if not good then
#    Print("Killed at Level 3\n");
  else
    lev3alive:=lev3alive+1;
    Print("\n----------------------------Survived at Level 3\n");
  fi;

  return;
end;

#
# Matchings enumeration to be used at Step 2
#

gram2:=[[]];
im1:=[];
im2:=[];
lset:=[];
rset:=[];

Step2_node:=function(v)
  local k,r,ll,rr,good,son;
```

```
  k:=4+big_tree[v].k;

  r:=List([1..22],j->-1/14);

  r[2+1]:=gram2[k][im1[1]];
  r[2+2]:=gram2[k][im1[2]];
  r[12+1]:=gram2[k][im2[1]];
  r[12+2]:=gram2[k][im2[2]];

  ll:=big_tree[v].l;
  if ll<>0 then
    r[lset[ll]]:=2/7;
  fi;

  rr:=big_tree[v].r;
  if rr<>0 then
    r[rset[rr]]:=2/7;
  fi;

  Append(r,gram2[k]{[5..k]});
  good:=AddOne(r);
  if good then
    son:=big_tree[v].son;
    if son=0 then
#      Print("\n",v," out of ",Length(big_tree),"\n");
      print_string[3]:=Concatenation(", big_tree ",String(v),"/",String(Length(big_
      print_string[4]:="";
      PrintCase();
```

```
      Step3();
    else
      while son<>0 do
        son:=Step2_node(son);
      od;
    fi;
#  else
#    Print("Step2 fails.\n");
  fi;
  return big_tree[v].bro;

end;

#
# Preparing Step 2
#

Step2:=function()
  local c,type,none,right,left,both;

  for z in [1..Length(segment_triples)] do

#    Print("\rDoing pair ",print_string[1],", matching ",print_string[2],", triple
    print_string[2]:=Concatenation(", triple ",String(z),"/",String(Length(segment_
    c:=segment_triples[z];

    w:=segments[c.m];
    gram2:=w.gram;
```

```
type:=w.type;
none:=type[1];
right:=type[2];
left:=type[3];
both:=type[4];

Read(Concatenation("data/trees/big",String(none),String(right),String(left),Str

im1:=c.im1;
im2:=c.im2;



type:=s.type;
none:=type[1];
right:=type[2];
left:=type[3];
both:=type[4];
lset:=4+Concatenation([none+1..none+right],[9-both..8]);

type:=t.type;
none:=type[1];
right:=type[2];
left:=type[3];
both:=type[4];
rset:=14+Concatenation([none+1..none+right],[9-both..8]);

v:=1;
while v<>0 do
```

```
      v:=Step2_node(v);
    od;
  od;
end;

#
# Matchings enumeration to be used at Step 1
#

match:=0;
tree:=[];

Step1_node:=function(v)
  local k,r,son,good;

  k:=12-match+tree[v].k;
  r:=List([1..12],j->-1/14);
  r[1]:=t.gram[k][im[1]];
  r[2]:=t.gram[k][im[2]];
  r[12-match+tree[v].m]:=2/7;
  Append(r,t.gram[k]{[3..k]});
  good:=AddOne(r);
  if good then
    son:=tree[v].son;
    if son=0 then
      print_string[1]:=Concatenation("matching ",String(v),"/",String(Length(tree))
      Step2();
    else
```

```
      while son<>0 do
        son:=Step1_node(son);
      od;
    fi;
  fi;
  return tree[v].bro;
end;


#
# main enumeration: Step 1
#

first:=0;
allalive:=0;

print_string:=[];

forced:=[];

for p in [63001..64000] do

if IsExistingFile(Concatenation("data/record/rec",String(p),".g")) then
  continue;
fi;

if IsExistingFile(Concatenation("data/working/",String(p),".g")) then
  continue;
```

```
fi;

PrintTo(Concatenation("data/working/",String(p),".g"),"Working\n");

Print("\n\n\nStarting segment pair ",p,"\n\n");

begt:=Runtime();

# reading in the triples

Read(Concatenation("data/abrev/",String(p),".g"));

# setting up the run

stillalive:=0;

n:=segment_pairs[p].n;
k:=segment_pairs[p].k;
im:=segment_pairs[p].im;

# update the first segment if necessary

if n<>first then
  first:=n;
  s:=segments[n];
  for i in [1..12] do
    r:=s.gram[i]{[1..i]};
    zero:=AddOne(r);
```

```
  od;
fi;

# add the unmatched part of second segment

t:=segments[k];
match:=Sum(t.type{[3,4]});

for i in [3..12-match] do
  r:=[];
  r[1]:=t.gram[i][im[1]];
  r[2]:=t.gram[i][im[2]];
  Append(r,List([3..12],j->-1/14));
  Append(r,t.gram[i]{[3..i]});
  zero:=AddOne(r);
od;

# reading in the small tree

Read(Concatenation("data/trees/small",String(match),".g"));

# the matched part of second segment is added iteratively
# in each case

u:=1;
while u<>0 do
  u:=Step1_node(u);
od;
```

```
endt:=Runtime();

Print("\n");
Print("Time used: ",StringTime(endt-begt),"\n");
Print("Total cases still alive: ", lev3alive, "\n");

Print("Saving the results...");
PrintTo(Concatenation("data/record/rec",String(p),".g"),"exact:=",exact,";\n");
Print("done\n");

RemoveFile(Concatenation("data/working/",String(p),".g"));

od;
```

LIST OF REFERENCES

[1] Monther R. Alfuraidan, Ibrahim O. Sarumi, and Sergey Shpectorov. On the non-existence of srg(76, 21, 2, 7). *Graphs and Combinatorics*, 35(4):847–854, July 2019. © Springer Japan KK, part of Springer Nature 2019 Cite this article as: Alfuraidan, M.R., Sarumi, I.O. & Shpectorov, S. Graphs and Combinatorics (2019). https://doi.org/10.1007/s00373-019-02039-w.

[2] E. Bannai and T. Ito. *Algebraic Combinatorics I: Association Schemes*. Basic Books, 1984.

[3] Andriy V. Bondarenko and Danylo V. Radchenko. On a family of strongly regular graphs with λ=1. *Journal of Combinatorial Theory, Series B*, 103(4):521–531, 2013.

[4] R. C. Bose. Strongly regular graphs, partial geometries and partially balanced designs. *Pacific Journal of Mathematics*, 13(2):389 – 419, 1963.

[5] R. C. Bose and Dale M. Mesner. On Linear Associative Algebras Corresponding to Association Schemes of Partially Balanced Designs. *The Annals of Mathematical Statistics*, 30(1):21 – 38, 1959.

[6] A.E. Brouwer. Parameters of strongly regular graphs. https://www.win.tue.nl/aeb/graphs/srg/srgtab.html.

[7] Andries E. Brouwer, Arjeh M. Cohen, and Arnold Neumaier. *Distance-Regular Graphs*. Ergebnisse der Mathematik und ihrer Grenzgebiete. 3. Folge / A Series of Modern Surveys in Mathematics. Springer Berlin, 1 edition, 1989.

[8] F.C Bussemake, S Čobeljić, D.M Cvetković, and J.J Seidel. Cubic graphs on $\leq$14 vertices. *Journal of Combinatorial Theory, Series B*, 23(2):234–235, 1977.

[9] F.C. Bussemaker, W.H. Haemers, R. Matron, and H.A. Wilbrink. A $(49, 16, 3, 6)$ strongly regular graph does not exist. *European Journal of Combinatorics*, 10(5):413–418, 1989.

[10] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.12.2*, 2022. https://www.gap-system.org.

[11] C. D. Godsil. *Algebraic combinatorics*. Chapman and Hall Mathematics Series. Chapman & Hall, 1 edition, 1993.

[12] David M. Goldschmidt. Automorphisms of trivalent graphs. *Annals of Mathematics*, 111(2):377–406, 1980.

[13] Willem H. Haemers and Edward Spence. The pseudo-geometric graphs for generalized quadrangles of order (3, t). *European Journal of Combinatorics*, 22(6):839–845, 2001.

[14] A. J. Hoffman and R. R. Singleton. On moore graphs with diameters 2 and 3. *IBM J. Res. Dev.*, 4(5):497–504, nov 1960.

[15] Ulrich Knauer and Kolja Knauer. *Algebraic Graph Theory*. De Gruyter, Berlin, Boston, 2019.

[16] Douglas A Leonard. Parameters of association schemes that are both p- and q-polynomial. *Journal of Combinatorial Theory, Series A*, 36(3):355–363, 1984.

[17] A. A. Makhnev. Moore graph with parameters (3250,57,0,1) does not exist. *arXiv: Combinatorics*, 2020.

[18] The House of Graphs. Connected cubic graphs. https://hog.grinvin.org/Cubic.

[19] J.J. Seidel. Strongly regular graphs with $(-1, 1, 0)$ adjacency matrix having eigenvalue 3. *Linear Algebra and its Applications*, 1(2):281–298, 1968.

[20] Edward Spence. Classication of regular two-graphs on 36 and 38 vertices. *The Australasian Journal of Combinatorics [electronic only]*, 24, 01 2001.

[21] G. Szegő. *Orthogonal Polynomials*. American Math. Soc: Colloquium publ. American Mathematical Society, 1975.

[22] J. H. van Lint and R. M. Wilson. *A Course in Combinatorics*. Cambridge University Press, 2 edition, 2001.

[23] JH van Lint and AE Brouwer. Strongly regular graphs and partial geometries. In *Enumeration and design*, pages 85–122. Academic Press Inc., 1984.

[24] Wikipedia. Cholesky decomposition. wikipedia.org/wiki/Cholesky_decomposition.

[25] Wikipedia. Table of simple cubic graphs. wikipedia.org/wiki/Table_of_simple_cubic_graphs.

[26] H.A. Wilbrink and A.E. Brouwer. A (57, 14, 1) strongly regular graph does not exist. *Indagationes Mathematicae (Proceedings)*, 86(1):117–121, 1983.