Icosahedral Virus Transitions: Computational Techniques

Kalamazoo College Senior Integrated Project

Xavier Silva

Advisors: Dr. Stephen Oloo and Dr. Dave Wilson

March 4, 2024

Contents

1 Abstract

Icosahedral viruses have the symmetries of an icosahedron, which involves 2-fold, 3-fold, and 5-fold rotational symmetries. We can characterize these virus capsids with finite sets of points (called a point array) which we realize in 6D (not just 3D) for the purpose of crystallography: our 6D point arrays naturally fit inside 6D icosahedral lattices. There are 55 standard point arrays (called *one-base*) from which we build all the others. We model virus maturation by 6D linear transformations (transitions) of point arrays that preserve some or all of icosahedral symmetry. To find these transitions (preserving either the full icosahedral group symmetry or one of its maximal subgroups $A_4, D_{10},$ or D_6) we solve matrix equations of the form $TB_0 = B_1$ for T, where T, the transition matrix, is a 6×6 matrix that depends on either 2, 4, 6, or 8 real variables and the matrices B_0 and B_1 are representations of the point arrays. We employ parallel computation techniques to efficiently find transitions between these point arrays. We are able to reproduce previously discovered transitions for the Cowpea Chlorotic Mottle Virus that preserve D_6 symmetry, and we create a comprehensive list of what symmetries can be preserved between any possible combination of the 55 standard point arrays.

2 Introduction

Icosahedral viruses have the symmetries of an icosahedron, which involves 2-fold, 3-fold, and 5-fold rotational symmetries. We can approximate these virus capsids with finite sets of points (called a point array) which we realize in 6D (not just 3D) for the purpose of crystallography: our 6D point arrays naturally fit inside 6D icosahedral Bravais lattices. There are 55 standard point arrays (called *one-base*) from which we build all the others [KT09].[1] We model virus maturation by 6D linear transformations (transitions) of point arrays that preserve some or all of icosahedral symmetry.

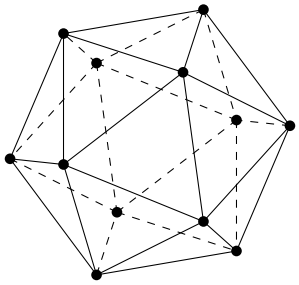


Figure 1: The icosahedron.

The symmetry group of the icosahedron is defined abstractly as follows.

Definition 1 (Icosahedral Group)**.** The icosahedral group, which we denote by $\mathcal{I}$, is the 60-element group given by the presentation

$$\mathcal{I} := \langle a, b | a^2 = b^3 = (ab)^5 = 1 \rangle .$$

We will work with the explicit realization of the icosahedral group given by the following 6D representation.

$$a = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad b = \begin{bmatrix} 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} .$$

[1] See section A.2 for the list of all 55 one-base point arrays.

The matrix a is rotation along a 2-fold symmetry axis, b is rotation along a 3-fold symmetry axis, and the product ab represents a 5-fold symmetry.

The icosahedral group has 3 maxmial subgroups, which we call A_4, D_{10}, and D_6.[2] These are subsets of the symmetries of an icosahedron, corresponding to the symmetries of the tetrahedron, pentagonal prism, and triangular prism respectively as shown in figure 2.

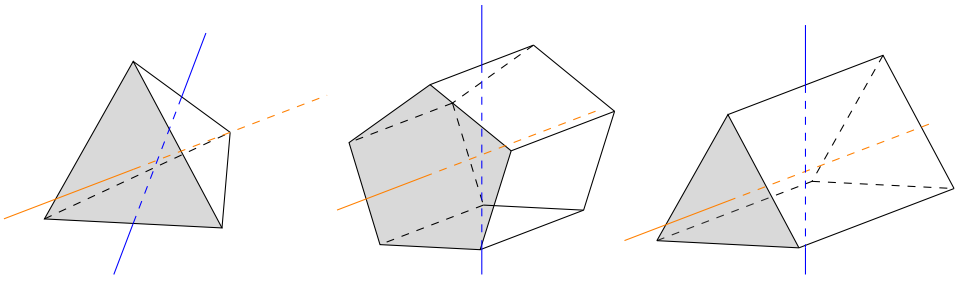


Figure 2: A visual representation of the maximal subgroups. From left to right we have A_4, D_{10}, and D_6. The 2-fold axes are labeled in blue while the 3-fold and 5-fold axes are labeled in orange. Note that A_4 has multiple 2-fold and 3-fold axes (by rotating the labeled axes such that the 3-fold is in the center of a different face). D_6 has multiple 2-fold axes but only one 3-fold axis.

3 Point Arrays

Icosahedral viruses are characterized by point arrays, which are a set of points in 3 dimensions with icosahedral symmetry at various radial levels. These sets of points approximate the structural features of the virus. When using point arrays to represent viruses, we investigate not just the 3-dimensional ones but also their *lifted* versions.

Definition 2 (Lifted Point Array)**.** Let $\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ be 6 dimensional vectors. Then the lifted point array generated by these vectors is

$$\Sigma(\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n) := \mathcal{I}\mathbf{v}_1 \cup \mathcal{I}\mathbf{v}_2 \cup \cdots \cup \mathcal{I}\mathbf{v}_n \cup (\mathcal{I}\mathbf{v}_1 + \mathcal{I}\mathbf{t}) \cup (\mathcal{I}\mathbf{v}_2 + \mathcal{I}\mathbf{t}) \cup \cdots \cup (\mathcal{I}\mathbf{v}_n + \mathcal{I}\mathbf{t}).$$

The vector $\mathbf{t}$ is referred to as the translation vector and the $\mathbf{v}_i$ are called the base vectors.

Notice that while icosahedral viruses are characterized by 3 dimensional point arrays, *lifted point arrays* are 6 dimensional. This is because we want lifted point arrays to fit inside icosahedral lattices, and no such lattices exist in dimensions lower than six.[3] Note that for this paper, we will be treating Bravais lattices and "lattices" as the same thing, despite "lattice" having many possible definitions in mathematics.[4]

While the lifted point arrays are 6 dimensional, we can project them into 3 dimensions using the projection matrix

$$\begin{bmatrix} \phi & 0 & -1 & 0 & \phi & 1 \\ 1 & \phi & 0 & -\phi & -1 & 0 \\ 0 & 1 & \phi & 1 & 0 & \phi \end{bmatrix}$$

where ϕ is the golden ratio given by $\phi = \frac{1}{2}(1+\sqrt{5})$ [ICS+12]. Our representation of $\mathcal{I}$ decomposes $\mathbb{R}^6$ into two inequivalent 3-dimensional irreducible decompositions, $\mathbb{R}^6 \simeq E^{\parallel} \oplus E^{\perp}$, one of which, $E^{\parallel}$, is the standard representation. We refer to $E^{\parallel}$ as the physical (parallel) space and $E^{\perp}$ as the perpendicular space. The projection matrix maps $\mathbb{R}^6$ to the physical space $E^{\parallel}$ and maps our lifted

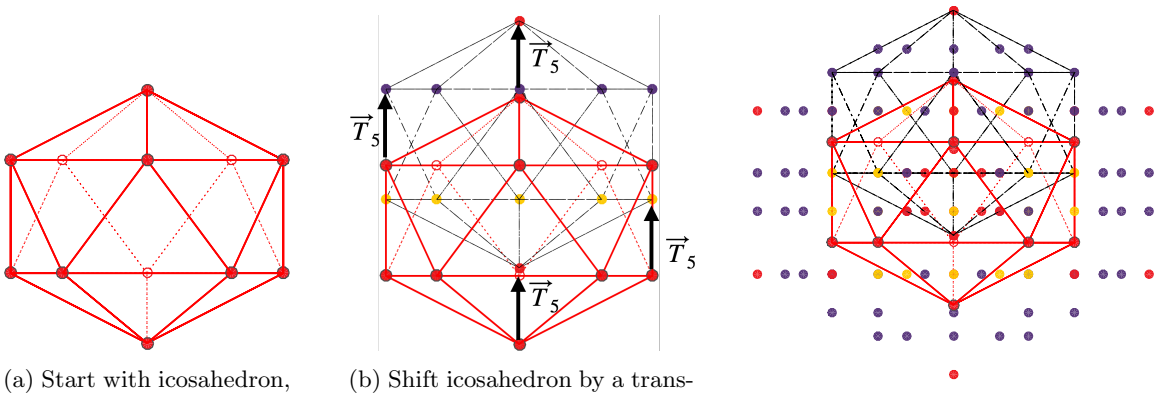
[2] Like the icosahedral group, these subgroups are also each generated by two 6×6 matrices.
[3] All lattices of dimension 5 and lower do not have 5-fold rotational symmetry.
[4] See definition 6 for the definition of a Bravais lattice.

point arrays into point arrays that approximate the virus capsids. Using the projection matrix, if we define

$$\mathbf{s} := \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

then the set $\mathcal{I}\mathbf{s}$ projects down to an icosahedron (see figure 1). While our point arrays are fundamentally a set of 3 dimensional points, for the rest of the paper we will focus on the 6 dimensional lifted point arrays.

There are 55 standard point arrays from which we build all others [KT09]. These are called the *one-base* point arrays and take the form $\mathcal{I}\mathbf{v} \cup (\mathcal{I}\mathbf{v} + \mathcal{I}\mathbf{t})$. This form tells us that only one base vector is used to create the point array. Figure 3 gives a visualization of the construction of a one base point array in 3D.



(a) Start with icosahedron, created by applying icosahedral symmtry on one vector. Mathematically this is $\mathcal{I}\mathbf{v}$.

(b) Shift icosahedron by a translation vector. In this example we are translating along the 5-fold axis of the icosahedron. Mathematically this is $\mathcal{I}\mathbf{v} \cup (\mathcal{I}\mathbf{v} + \mathbf{t})$.

(c) Reapply icosahedral symmetry, creating our one-base point array and getting the final expression $\mathcal{I}\mathbf{v} \cup (\mathcal{I}\mathbf{v} + \mathcal{I}\mathbf{t})$.

Figure 3: The process of creating a one-base point array. Graphics from Dr. Dave Wilson.

Here we define these one-base point arrays with two vectors, a translation and base vector. However, we could define the same point array with more than two vectors, if we added on a vector from the base vector's icosahedral orbit. Thus if $\mathbf{u} \in \mathcal{I}\mathbf{v}$ we would have that

$$\mathcal{I}\mathbf{v} \cup (\mathcal{I}\mathbf{v} + \mathcal{I}\mathbf{t}) = \mathcal{I}\mathbf{v} \cup \mathcal{I}\mathbf{u} \cup (\mathcal{I}\mathbf{v} + \mathcal{I}\mathbf{t}) \cup (\mathcal{I}\mathbf{u} + \mathcal{I}\mathbf{t}). \tag{1}$$

Therefore we have many ways of talking about the same point array.[5]

Definition 3 (Minimal Generating List)**.** A list of six dimensional vectors $(\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n)$ is a *minimal generating list* and the union

$$\Sigma(\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n) := \left(\bigcup_{i=1}^{n} \mathcal{I}\, \mathbf{v}_i \right) \cup \left(\bigcup_{i=1}^{n} \mathcal{I}\, \mathbf{v}_i + \mathcal{I}\, \mathbf{t} \right)$$

is called the *(n-base) lifted point array* generated by this list so long as

[5] A quick proof for equation 1: Let LHS $= \mathcal{I}\mathbf{v} \cup (\mathcal{I}\mathbf{v} + \mathcal{I}\mathbf{t})$ and RHS $= \mathcal{I}\mathbf{v} \cup \mathcal{I}\mathbf{u} \cup (\mathcal{I}\mathbf{v} + \mathcal{I}\mathbf{t}) \cup (\mathcal{I}\mathbf{u} + \mathcal{I}\mathbf{t})$ It is obvious that LHS $\subseteq$ RHS. Now take an element of the right hand side. Let $x \in$ RHS. If $x \in \mathcal{I}\mathbf{v} \cup (\mathcal{I}\mathbf{v} + \mathcal{I}\mathbf{t})$ we trivially have $x \in$ LHS. So first let $x \in \mathcal{I}\mathbf{u}$. Then there exists $G \in \mathcal{I}$ such that $x = G\mathbf{u}$. Since $\mathbf{u} \in \mathcal{I}\mathbf{v}$ then there exists $H \in \mathcal{I}$ such that $\mathbf{u} = H\mathbf{v}$. Thus $x = GH\mathbf{v} \implies x \in \mathcal{I}\mathbf{v}$. Using the same idea, we get that $x \in (\mathcal{I}\mathbf{u} + \mathcal{I}\mathbf{t}) \implies x \in (\mathcal{I}\mathbf{v} + \mathcal{I}\mathbf{t})$. Therefore LHS $\supseteq$ RHS. We get equality by double inclusion.

1. Each of the vectors $\mathbf{t}$, $\mathbf{v}_i$ in the list points along a 2-fold, 3-fold or 5-fold rotational axis of the action of $\mathcal{I}$.

2. The orbits $\mathcal{I}\ \mathbf{v}_i$, $1 \leq i \leq n$, are all distinct.

3. We have the *strict* inequality $|\Sigma(\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n)| < \sum_{i=1}^{n} |\mathcal{I}\ \mathbf{v}_i| + |\mathcal{I}\ \mathbf{v}_i||\mathcal{I}\ \mathbf{t}|$.

We say that a lifted point array is *minimally generated* by $(\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n)$ if this list is a minimal generating list.[6]

Definition 4 (Generating List)**.** More generally, a list $(\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n)$ is a *generating list* if it has a sub-list $(\mathbf{t}, \mathbf{v}_{i_1}, \mathbf{v}_{i_2}, ..., \mathbf{v}_{i_m})$, $1 \leq i_1 < i_2 < ... < i_m \leq n$, that is a minimal generating list and the unions

$$\Sigma(\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n) := \left(\bigcup_{i=1}^{n} \mathcal{I}\ \mathbf{v}_i\right) \cup \left(\bigcup_{i=1}^{n} \mathcal{I}\ \mathbf{v}_i + \mathcal{I}\ \mathbf{t}\right)$$

and $\Sigma(\mathbf{t}, \mathbf{v}_{i_1}, \mathbf{v}_{i_2}, ..., \mathbf{v}_{i_m})$ are equal.

These definitions allow us to distinguish between the two cases given in equation 1 on the preceding page. In that example the list $(\mathbf{t}, \mathbf{v})$ is the minimal generating list and the list $(\mathbf{t}, \mathbf{v}, \mathbf{u})$ is simply a generating list (assuming $\mathbf{u} \in \mathcal{I}\mathbf{v}$).

3.1 Transitions Between Point Arrays

Let us take an overall look at how we wish to use point arrays and their lifted counterparts. Viruses are characterized by 3-dimensional point arrays. We want our point arrays to represent the symmetry and structure of the virus. We then lift the point arrays into 6 dimensions, because we want the point array to fit inside an icosahedral lattice. Lattices on a general level encode symmetry (in particular they have both rotational and translational symmetry). We will go more into depth about lattices in section 4.1. By understanding the lattice structure, we can use the symmetries of a lattice in order to describe symmetry of the point array. Point arrays let us look at the most important structural points of a virus and can determine what potential modifications could be done to a particular virus [Wil20].

With point arrays, we can try to find linear transformations that map one point array onto another. These transformations represent how a virus's structure changes over time. In general, finding such a linear map is relatively easy. A more difficult challenge is to find such a linear map that preserves symmetry. Our point arrays have icosahedral symmetry. Therefore our desire is to find a linear map from one point array to another where the intermediate point arrays also have icosahedral symmetry. It may also be desirable to find a map that preserves the symmetries of one of the maximal subgroups of $\mathcal{I}$ in cases where we cannot find a transition that preserves all of icosahedral symmetry.

Additionally, instead of mapping the point arrays directly onto each other, our linear transformations will transform one lattice into another. The consequence of doing this is that when we restrict our map to the point array, it becomes a map from one point array to another. The process of mathematically describing these symmetry preserving maps and how we find them will be the focus of the rest of this paper. However, before we delve into the mathematics, we take a quick look at an example of virus that will be relevant to us.

[6] When looking at the 3rd condition for an n-base lifted point array, it may seem like we trivially have this condition. However, it is not trivial because we require a strict inequality. What this condition tells us is that the translated icosahedral orbits must intersect in some way. That is, there must exist $G, H \in \mathcal{I}$ such that the sets $\mathcal{I}\mathbf{v} + G\mathbf{t}$ and $\mathcal{I}\mathbf{v} + H\mathbf{t}$ have a nonempty intersection.

3.2 Viral Maturation

Icosahedral viruses mature over time, and maturation is how they become infectous. For this paper we will focus on the Cowpea Chlorotic Mottle Virus (CCMV). This virus is characterized by one of the standard point arrays.[7] How CCMV matures is shown in figure 4. Notice the points in orange, these points collectively are the point arrays that characterize the CCMV virus. We wish to see if there exists a linear transformation that maps the native point array to the mature point array while preserving icosahedral symmetry (or one of its maximal subgroups).

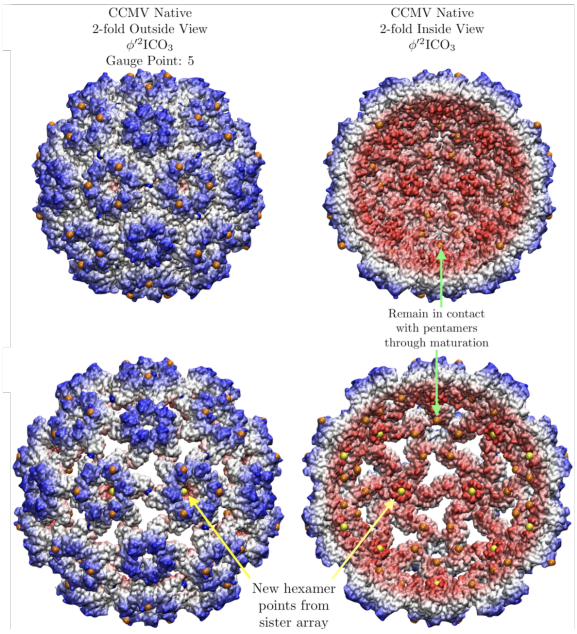


Figure 4: The above diagram shows CCMV in its native state (first row) and mature state (second row). Notice the points in orange are the points that characterize CCMV. Graphics from Dr. Dave Wilson.

4 Mathematical View of the Problem

Our desire is to find transitions between point arrays that preserves some form of icosahedral symmetry. In order to find such transitions, we need to understand this problem mathematically. On the highest level of abstraction, we are simply solving matrix equations of the form

$$TB_0 = B_1.$$

In this equation, B_0 and B_1 represent the native and mature point arrays respectively, and T is our desired linear transformation that preserves icosahedral symmetry.[8] Because we want to represent the point arrays, there is a specific structure to the B_0 and B_1 matrices. Let us call us these matrices *point array matrices*.

[7] The native state is characterized by the one-base point array 10 and the mature state is characterized by the one-base point array 27. See section A.2 for the definitions of these point arrays.

[8] Recall that we realize icosahedral symmetry as a matrix group and lifted point arrays are sets of vectors, so it makes sense that we are solving a matrix equation.

Definition 5 (Point Array Matrix)**.** If $(\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n)$ is a generating list of a point array P, then the $6 \times (n+1)$ matrix

$$\begin{bmatrix} | & | & | & \cdots & | \\ \mathbf{t} & \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ | & | & | & \cdots & | \end{bmatrix}$$

is the corresponding point array matrix for P. We denote the set of all point array matrices for P as $\text{pam}(P)$. The set $\text{pam}(P)$ is an infinite set since P has an infinite number of generating lists.[9]

If $(\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n)$ is a *minimal* generating list for P, then the corresponding matrix is called a minimal point array matrix. We denote the set of all minimal point arrays matrices for P as $\text{minpam}(P)$. Note that $\text{minpam}(P)$ is a finite set and contains all permutations of the minimal generating lists for P.

The B_0 and B_1 matrices are both point array matrices and so both have this general form for their respective point arrays.

One fact that will be useful to know is how many minimal point array matrices there are for a point array. Given a point array P with minimal generating list $(\mathbf{t}, \mathbf{v}_1, ..., \mathbf{v}_n)$ then

$$|\text{minpam}(P)| = (|\mathcal{I}\mathbf{t}| \cdot |\mathcal{I}\mathbf{v}_1| \cdot |\mathcal{I}\mathbf{v}_2| \cdot \cdots \cdot |\mathcal{I}\mathbf{v}_n|) \cdot n!$$

Note that we multiply by $n!$ because $\text{minpam}(P)$ contains all permutations of our minimal generating list.

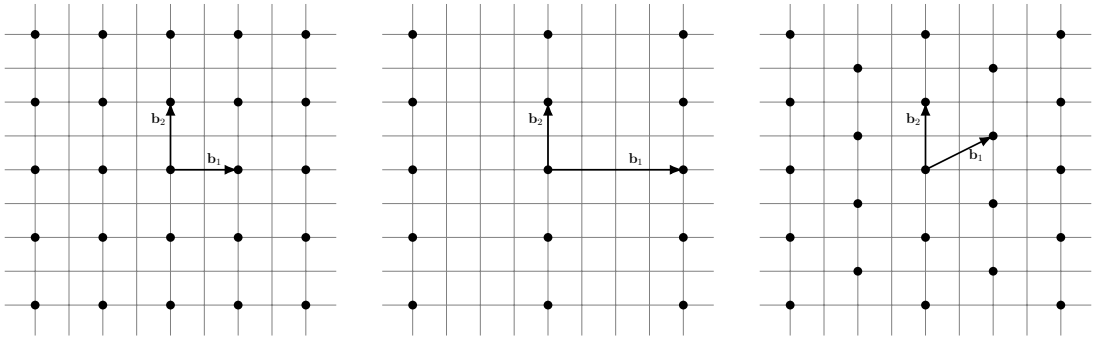
4.1 Lattices

In order to understand the nature of the transition matrix, we must understand lattices and how point arrays work with them. For our point arrays, we use 6-dimensional icosahedral lattices. First a definition:

Definition 6 (Bravais Lattice)**.** A *Bravais lattice* in $\mathbb{R}^n$ is the integer span of n linearly independent vectors; the Bravais lattice generated by linearly independent vectors $\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n$ is

$$\mathcal{L} := \{k_1\mathbf{a}_1 + k_2\mathbf{a}_2 + \cdots + k_n\mathbf{a}_n \mid k_i \in \mathbb{Z}\}$$

Our point arrays fit inside this kind of lattice. Lattices are interesting because of their discrete structure. Let us consider three 2-dimensional examples, shown in figure 5, that help explain two important invariants of lattices: the point group and the lattice group.



(a) Square Lattice (b) Rectangular Lattice (c) Centered Rectangular Lattice

Figure 5: Examples of 2-dimensional lattices. The grid represents standard integer coordinates for all three lattices. The basis vectors for each are labeled as $\mathbf{b}_1$ and $\mathbf{b}_2$.

[9] If we have a generating list $(\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n)$, then $(\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n, \mathbf{v}_n)$, $(\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n, \mathbf{v}_n, \mathbf{v}_n)$, etc. are all generating lists for P.

Definition 7 (Point Group)**.** The point group of a lattice $\mathcal{L}$, denoted $\mathcal{P}(\mathcal{L})$, is the set of all orthogonal transformations that leave $\mathcal{L}$ invariant.[10]

Let us determine the point group of our three example lattices. For all three example lattices, we have two types of orthogonal transformations that leave the lattice invariant: rotations and reflections. In the case of the square lattice, we have 4 reflections (reflections over the x-axis, y-axis, and the lines $y = x$ and $y = -x$) and 4 rotations (rotate by either $0°, 90°, 180°,$ or $270°$). These 8 transformations are the point group for the square lattice. For both the rectangular and centered rectangular, we have 2 reflections (reflections over the x and y axes) and 2 rotations (rotate by either $0°$ or $180°$). These four transformations are the point group for both the rectangular and centered rectangular lattices. The point group gives us a coarse classification of lattices, in this case enabling us to distinguish the square lattice from the rectangular ones due to the non-isomorphic point groups.

Despite the rectangular and centered rectangular lattices having the same point group we notice that the fundamental structure of the two lattices are different. Therefore there must be some way to distinguish them from each other by some other mathematical structure different from the point group. This finer classification is given by the lattice group, which combines the point group and basis vectors together.

Definition 8 (Lattice Group)**.** Let B be a basis matrix[11] for a lattice $\mathcal{L}$. Then the lattice group, $\Lambda(B)$, consists of the representations of the elements of $\mathcal{P}(\mathcal{L})$ in this chosen basis. That is,

$$\Lambda(B) := \{N \in \mathrm{GL}(n, \mathbb{Z}) \mid BN = MB \quad \text{for some} \quad M \in \mathcal{P}(\mathcal{L})\}$$

where $\mathrm{GL}(n, \mathbb{Z})$ is the set of invertible $n \times n$ matrices with integer entries, and in fact

$$\Lambda(B) = B^{-1}\, \mathcal{P}(\mathcal{L})\, B.$$

In order to understand this definition, we will compute the lattice groups for the rectangular and centered rectangular lattices. First, since transformations of 2-dimensional space can be thought of as 2×2 matrices, we will express the point groups in this form. Let R be the rectangular lattice and CR be the centered rectangular lattice. Then

$$\mathcal{P}(R) = \mathcal{P}(CR) = \left\{ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \right\}$$

This set gives the matrix representations of rotation by $0°$, rotation by $180°$, reflection over the y-axis and reflection over the x-axis respectively. In order to compute the lattice group, we first need the basis matrix for both lattices:

$$B_R = \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix} \quad B_{CR} = \begin{bmatrix} 2 & 0 \\ 1 & 2 \end{bmatrix}$$

Now in order to compute the lattice groups $\Lambda(R)$ and $\Lambda(CR)$, we will solve the equations $B_R N = MB_R$ and $B_{CR} N = MB_{CR}$ for N for each M in the point group. The computations shown in table 1 tell us what the lattice groups are for our two rectangular lattices:

$$\Lambda(R) = \left\{ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \right\}$$

$$\Lambda(CR) = \left\{ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ -1 & -1 \end{bmatrix} \right\}$$

Notice that the lattice groups for the rectangular and centered rectangular lattices are not $\mathrm{GL}(2, \mathbb{Z})$ conjugate. Because these lattice groups are not $\mathrm{GL}(2, \mathbb{Z})$ conjugate, the structure of their corresponding lattices is fundamentally different. So by using lattice groups, we can clearly distinguish between different lattices.

[10] Orthogonal transformations can be thought as distance preserving linear transformations that keep the origin fixed.

[11] That is, $B = \begin{bmatrix} | & | & \dots & | \\ \mathbf{b}_1 & \mathbf{b}_2 & \dots & \mathbf{b}_n \\ | & | & \dots & | \end{bmatrix}$ where $\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n$ are basis vectors.

$\Lambda(R)$	$\Lambda(CR)$
$\begin{bmatrix}4 & 0\\ 0 & 2\end{bmatrix}\cdot N=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}4 & 0\\ 0 & 2\end{bmatrix}\implies N=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$	$\begin{bmatrix}2 & 0\\ 1 & 2\end{bmatrix}\cdot N=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}2 & 0\\ 1 & 2\end{bmatrix}\implies N=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$
$\begin{bmatrix}4 & 0\\ 0 & 2\end{bmatrix}\cdot N=\begin{bmatrix}-1 & 0\\ 0 & -1\end{bmatrix}\begin{bmatrix}4 & 0\\ 0 & 2\end{bmatrix}\implies N=\begin{bmatrix}-1 & 0\\ 0 & -1\end{bmatrix}$	$\begin{bmatrix}2 & 0\\ 1 & 2\end{bmatrix}\cdot N=\begin{bmatrix}-1 & 0\\ 0 & -1\end{bmatrix}\begin{bmatrix}2 & 0\\ 1 & 2\end{bmatrix}\implies N=\begin{bmatrix}-1 & 0\\ 0 & -1\end{bmatrix}$
$\begin{bmatrix}4 & 0\\ 0 & 2\end{bmatrix}\cdot N=\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}4 & 0\\ 0 & 2\end{bmatrix}\implies N=\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}$	$\begin{bmatrix}2 & 0\\ 1 & 2\end{bmatrix}\cdot N=\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}2 & 0\\ 1 & 2\end{bmatrix}\implies N=\begin{bmatrix}-1 & 0\\ 1 & 1\end{bmatrix}$
$\begin{bmatrix}4 & 0\\ 0 & 2\end{bmatrix}\cdot N=\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}\begin{bmatrix}4 & 0\\ 0 & 2\end{bmatrix}\implies N=\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$	$\begin{bmatrix}2 & 0\\ 1 & 2\end{bmatrix}\cdot N=\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}\begin{bmatrix}2 & 0\\ 1 & 2\end{bmatrix}\implies N=\begin{bmatrix}1 & 0\\ -1 & -1\end{bmatrix}$

Table 1: The series of computations used to compute $\Lambda(R)$ and $\Lambda(CR)$.

Remark. The lattice groups associated to bases of the same lattice may be different but they are always $\mathrm{GL}(n,\mathbb{Z})$ conjugate. This is because if B, B' are basis matrices of the same lattice then there is an invertible integer matrix N such that

$$BN = B'.$$

Now that we have explained the importance of the point and lattice groups using these 2D examples, we can define the 6-dimensional icosahedral lattices relevant to our point arrays. We want the lattices in which our point arrays sit to be invariant under the icosahedral group $\mathcal{I}$. There are three 6-dimensional icosahedral lattices that are relevant to us:

$$\begin{aligned}
\mathcal{L}_{SC} &:= \{(x_1, x_2, ..., x_n) \mid x_i \in \mathbb{Z}\}\\
\mathcal{L}_{BCC} &:= \{\tfrac{1}{2}(x_1, x_2, ..., x_n) \mid x_i \in \mathbb{Z},\ x_i \equiv x_j \bmod 2\ \ \forall i, j\}\\
\mathcal{L}_{FCC} &:= \{\tfrac{1}{2}(x_1, x_2, ..., x_n) \mid x_i \in \mathbb{Z},\ x_1 + x_2 + ... + x_6 \equiv 0 \bmod 2\ \ \forall i, j\},
\end{aligned}$$

which are referred to the standard simple cubic, body centered cubic, and face centered cubic lattices respectively. Notice that $\mathcal{L}_{SC} \subset \mathcal{L}_{BCC} \subset \mathcal{L}_{FCC}$. All icosahedral lattices are equivalent to one of these three [LR88].[12] When considering our point arrays, we want to know which of these lattices they fit into best.

Definition 9 (Minimal Lattice)**.** Let P be a lifted point array. A *minimal lattice*, denoted $\mathcal{L}_{\mathcal{I}}(P)$ is an icosahedral lattice containing P that is contained in every other icosahedral lattice containing P.

A minimal lattice can also be thought of the smallest lattice that contains P. The following propositions (which we will state without proof) tells what the minimal lattice is for a set of points in 6-dimensions.

Proposition 10 ([ICS+12, Proposition 1])**.** *Given vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n \in \mathbb{R}^6$ all contained in some 6-dimensional icosahedral lattice $\mathcal{L}$, the $\mathbb{Z}$-span of the union $\mathcal{O} := \mathcal{I}\mathbf{v}_1 \cup \mathcal{I}\mathbf{v}_2 \cup \cdots \cup \mathcal{I}\mathbf{v}_n$, which is defined as*

$$\mathcal{L}_{\mathcal{I}}(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n) := \left\{ \sum_{\mathbf{w}\in\mathcal{O}} c_{\mathbf{w}}\mathbf{w} \;\middle|\; c_{\mathbf{w}} \in \mathbb{Z} \right\},$$

is the minimal icosahedral lattice containing the vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$.

Proposition 11 ([ICS+12, Proposition 1])**.** *If a lifted point array is generated by the vectors $\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$, then the minimal lattice containing that point array is $\mathcal{L}_{\mathcal{I}}(\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n)$.*

[12] Roughly speaking, two lattices are equivalent if their lattice groups are conjugate in $\mathrm{GL}(n,\mathbb{Z})$.

4.2 Transitions Between Lifted Point Arrays

Now that we have an understanding of lattices, we have the tools necessary to mathematically describe our symmetry preserving transformations between point arrays. The symmetries of our point arrays are encoded within its minimal lattice and its corresponding lattice group. Thus we want our linear transformations to not only be a map from one point array to another, but also a map of their minimal lattices that will preserve some form of symmetry (in our case it will be a subgroup of $\mathcal{I}$). The following definition gives us a description of a transformation that does this.

Definition 12 (Definitions 4 and 5 from [ICS+12])**.** Let $\mathcal{L}_0$ and $\mathcal{L}_1$ be icosahedral lattices and $\mathcal{G}_0$ be a subgroup of $\mathcal{I}$.

1. A *transition* from $\mathcal{L}_0$ and $\mathcal{L}_1$ with intermediate symmetry $\mathcal{G}_0$ is a continuous function

$$\begin{aligned} \mathcal{B}\colon [0,1] &\to \mathrm{GL}(n,\mathbb{R}) \\ \sigma &\mapsto \mathcal{B}_\sigma \end{aligned}$$

   such that $\mathcal{B}_0$ is a basis matrix of $\mathcal{L}_0$, $\mathcal{B}_1$ is a basis matrix of $\mathcal{L}_1$, and letting

$$\mathcal{G} := B_0^{-1}\mathcal{G}_0\mathcal{B}_0 \subset \Lambda(\mathcal{B}_0)$$

   we have

$$\mathcal{G} \subset \Lambda(\mathcal{B}_\sigma) \text{ for all } \sigma \in [0,1].$$

2. Additionally a transition $\mathcal{B}$ from $\mathcal{L}_0$ to $\mathcal{L}_1$ with intermediate symmetry $\mathcal{G}_0$ is compatible with projection if

$$\mathcal{B}_\sigma \mathcal{G} B_\sigma^{-1} \subset \mathcal{P}(\mathcal{L}(\mathcal{B}_\sigma)) \text{ for all } \sigma \in [0,1].$$

We refer to

$$T(\sigma) := \mathcal{B}_\sigma \mathcal{B}_0^{-1} \quad \text{and} \quad T := T(1) = \mathcal{B}_1 \mathcal{B}_0^{-1}$$

as the *transition path* and the *transition* respectively.

This definition tells us that our transitions from one lattice to another will have the following properties:

1. All intermediate lattices have at least $\mathcal{G}_0$ symmetry;
2. The intermediate lattices are actually $\mathcal{G}_0$ invariant.

Additionally, because we want our transitions to both describe a continuous path and preserve $\mathcal{G}_0$ symmetry, it turns out that our transition must be in the centralizer of $\mathcal{G}_0$ [ICS+12].

Definition 13 (Centralizer)**.** The centralizer of a group G, denoted by $Z(G)$ is the set of elements that commute with all elements of G. That is,

$$Z(G) = \{z \mid gz = zg \ \forall g \in G\}.$$

Since these transitions belong to the centralizer of our symmetry groups, the matrices that describe our transitions will also commute with the lattice's point group and lattice group.

4.3 The Form of Transitions

In the previous section, we built up the necessary background in order to arrive at the conclusion that our transitions must be in the centralizer of $\mathcal{I}$ or in the centralizer of one its maximal subgroups. Luckily for us, Indelicato et al. have explicitly calculated what the centralizers are for our symmetry

groups [ICS+12]. Let's first start with the icosahedral group. A transition T that preserves all of icosahedral symmetry must have the form:

$$T = \begin{bmatrix} z & x & -x & -x & x & x \\ x & z & x & -x & -x & x \\ -x & x & z & x & -x & x \\ -x & -x & x & z & x & x \\ x & -x & -x & x & z & x \\ x & x & x & x & x & z \end{bmatrix}$$

where $x, z \in \mathbb{R}$. Any matrix of the form given in the above equation is in $Z(\mathcal{I})$, the centralizer of $\mathcal{I}$, and thus mathematically describes a transition that preserves icosahedral symmetry. Now we can look at the maximal subgroups of $\mathcal{I}$. Below are the general matrix forms for $Z(A_4)$, $Z(D_{10})$, and $Z(D_6)$. [ICS+12]

$$\begin{bmatrix} z & -x & -y & -t & t & -x \\ t & z & t & x & x & y \\ -y & -x & z & t & -t & -x \\ x & -t & -x & z & y & t \\ -x & -t & x & y & z & t \\ t & y & t & -x & -x & z \end{bmatrix} \in Z(A_4) \qquad \begin{bmatrix} z & x & y & y & x & t \\ x & z & x & y & y & t \\ y & x & z & x & y & t \\ y & y & x & z & x & t \\ x & y & y & x & z & t \\ u & u & u & u & u & w \end{bmatrix} \in Z(D_{10})$$

$$\begin{bmatrix} u & w & -w & x & s & s \\ -t & y & v & -v & z & -t \\ t & v & y & v & t & -z \\ z & -v & v & y & -t & -t \\ s & x & -w & w & u & s \\ s & w & -x & w & s & u \end{bmatrix} \in Z(D_6)$$

Similarly to $Z(\mathcal{I})$, we have that all of these variables represent real numbers. These general matrix forms give us linear transformations that preserve the symmetry of one of the maximal subgroups. Notice that with particular choices of variables, all three of these general forms can be equal to general form for icosahedral symmetry.[13] This tells us that $Z(\mathcal{I})$ is a subset of the sets $Z(A_4), Z(D_{10}), Z(D_6)$.

Definition 12 gives us a rigorous definition of a transition between point arrays. However, for the purposes of computing transitions between point arrays, we can vastly simplify our definitions. For transition matrices, we will only need to work with the following definition, which we will be using for the rest of this paper.

Definition 14 (Transition Matrix)**.** Let $\mathcal{G} = \mathcal{I}, A_4, D_{10}, \text{or } D_6$. A *transition matrix* that preserves intermediate symmetry $\mathcal{G}$ is an invertible matrix[14] that is also an element of $Z(\mathcal{G})$.

Additionally while definition 12 requires us to find a continuous map from 0 to 1, we actually only need to find the final transition T in the transition path.[15] By simplifying our definitions, it will give us a strong start in finding and computing transitions between point arrays.

4.4 How To Find Transitions

We now understand the mathematical structures of the problem. The question now is what procedure should we use in order to find transitions that preserve symmetry.

Suppose we want to a symmetry preserving transition from point array P to point array Q. Then to find this transition, we need to find a point array matrix $B_0 \in \text{minpam}(P)$ and a point array

[13]For $Z(A_4)$, choosing $t = y = -x$ gives us the general form for $Z(\mathcal{I})$. For $Z(D_{10})$, choosing $y = u = t = x$ and $w = z$ gives us the general form for $Z(\mathcal{I})$. And for $Z(D_6)$, choosing $u = y$, $x = t = z = -w$, and $s = v = w$ gives us the general form for $Z(\mathcal{I})$.

[14]Note that we require our transitions to be invertible matrices because if we find a transition matrix T for $P \to Q$, we want to be guarenteed that T^{-1} is a transition for $Q \to P$.

[15]The transition path describes a straight line, so if we have the final transition matrix we can simply create a continuous linear map from the identity matrix to T.

matrix $B_1 \in \text{minpam}(Q)$ such that the equation $TB_0 = B_1$ can be solved for T. Thus we if wish to find a transition preserving icosahedral symmetry, we solve all equations of the form

$$\begin{bmatrix} z & x & -x & -x & x & x \\ x & z & x & -x & -x & x \\ -x & x & z & x & -x & x \\ -x & -x & x & z & x & x \\ x & -x & -x & x & z & x \\ x & x & x & x & x & z \end{bmatrix} \cdot \begin{bmatrix} | & | & | & \cdots & | \\ \mathbf{t} & \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ | & | & | & \cdots & | \end{bmatrix} = \begin{bmatrix} | & | & | & \cdots & | \\ \mathbf{t}' & \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n \\ | & | & | & \cdots & | \end{bmatrix} \tag{2}$$

where $(\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n)$ and $(\mathbf{t}', \mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n)$ are *minimal* generating lists for P and Q.[16] Because $\text{minpam}(P)$ and $\text{minpam}(Q)$ are sets of matrices, equation 2 describes a set of equations, whose cardinality is equal to $|\text{minpam}(P)| \cdot |\text{minpam}(Q)|$. Therefore, to find an icosahedral symmetry preserving transition, we must find an equation within this set that is solvable.[17]

4.5 CCMV D_6 Transition Example

The following is an example of an equation for the CCMV virus the preserves D_6 symmetry:

$$\begin{bmatrix} u & w & -w & x & s & s \\ -t & y & v & -v & z & -t \\ t & v & y & v & t & -z \\ z & -v & v & y & -t & -t \\ s & x & -w & w & u & s \\ s & w & -x & w & s & u \end{bmatrix} \cdot \begin{bmatrix} -\frac{1}{2} & -\frac{3}{2} \\ -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ -1 & 0 \\ 0 & -1 \\ 0 & -1 \\ 0 & -1 \end{bmatrix}$$

and simplifies to:

$$\begin{bmatrix} -\frac{u}{2} - w - \frac{x}{2} & s - \frac{3u}{2} + w - \frac{x}{2} \\ v - \frac{y}{2} - \frac{z}{2} & t + \frac{y}{2} + \frac{z}{2} \\ -t - v + \frac{y}{2} - \frac{z}{2} & -t - \frac{y}{2} - \frac{z}{2} \\ v - \frac{y}{2} - \frac{z}{2} & -t - v - \frac{y}{2} - \frac{3z}{2} \\ -\frac{u}{2} - w - \frac{x}{2} & -s + \frac{u}{2} + \frac{x}{2} \\ -s + \frac{u}{2} - w - \frac{x}{2} & -s + \frac{u}{2} + \frac{x}{2} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ -1 & 0 \\ 0 & -1 \\ 0 & -1 \\ 0 & -1 \end{bmatrix}$$

This equation can be solved and gives us the transition matrix:

$$T = \begin{bmatrix} 1 & 0 & 0 & -1 & 1 & 1 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

In this case, these particular B_0 and B_1 matrices make the equation such that it can be solved for T. However, there are many other choices for B_0 and B_1 for which the equation cannot be solved for T.

[16] It is possible that P and Q are minimally generated by an unequal number vectors, which would cause this equation to be invalid. There is a way to make the equation work when this is case and is described in section A.1.

[17] One may notice that by how this equation is setup, the order of the generators matters. This is in constrast to the definition of an lifted point array, where the order doesn't matter. However, because of how we have carefully defined minimal generating lists and minimal point array matrices, we have taken care of this issue since the set minpam takes into account permutations. Since we account for all permutations, the number of equations that must be checked for the transition $P \to Q$ is simply $|\text{minpam}(P)| \cdot |\text{minpam}(P)|$.

5 Computational Techniques

In order to compute transition matrices, we need to generate all of the equations that could describe a transition from point array P to point array Q, and then check if any of them can be solved. Generating all of the possible equations is simple enough, as we can take the cartesian product of the icosahedral orbits of the generators. That is, if P is minimally generated by $(\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n)$ and Q is minimally generated by vectors $(\mathbf{t}', \mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n)$, then the each equation we need to solve corresponds to an element of the product

$$\mathcal{I}\mathbf{t} \times \mathcal{I}\mathbf{v}_1 \times \mathcal{I}\mathbf{v}_2 \times \cdots \times \mathcal{I}\mathbf{v}_n \times \mathcal{I}\mathbf{t}' \times \mathcal{I}\mathbf{u}_1 \times \mathcal{I}\mathbf{u}_2 \times \cdots \times \mathcal{I}\mathbf{u}_n.$$

So we just need to generate and loop over all elements of this cartesian product and check if the equation is solvable. This description of how to solve the problem is the brute force approach. While the approach will give us the answer we desire, it may be very slow in finding transitions, especially if $n \geq 3$ since the size of the cartesian product grows exponentially with n.[18]

Despite this, we can speed up this computation even with this brute force approach through parallelization. We can parallelize this problem because each matrix equation is independent of every other equation. Thus each element of our cartesian product is an independent task that needs to be completed. Because this problem is easily parallelizable, it known as *embarassingly parallel*.

Definition 15 (Embarassingly Parallel [Pac11])**.** An *embarrassingly parallel* problem is one where little or no effort is needed to separate the problem into a number of parallel tasks.

Parallelization speeds up computation for the problem, but even for $n = 3$, the brute force approach is still too slow.

Another way to speed up this computation would be to reduce the number of equations we need to solve in order to decide whether a symmetry preserving transition exists. In order to do this we need to be able to determine whether an equation can or cannot be solved without actually fully solving the equation. The following proposition will help us determine when a matrix equation can and cannot be solved.

Proposition 16 (Matrix Equation Solving)**.** *Let M be an $n \times n$ matrix with variables. Let $\mathbf{a}_1, \mathbf{a}_2, \ldots \mathbf{a}_k$, $\mathbf{b}_1, \mathbf{b}_2, \ldots \mathbf{b}_k$ be n-dimensional vectors. If the equation*

$$M \cdot \begin{bmatrix} | & | & \cdots & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_k \\ | & | & \cdots & | \end{bmatrix} = \begin{bmatrix} | & | & \cdots & | \\ \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_k \\ | & | & \cdots & | \end{bmatrix}$$

is solvable then the equation

$$M\mathbf{a}_i = \mathbf{b}_i$$

is solvable for all $i \in \{1, 2, \ldots k\}$.[19]

Proof. Rewrite the given matrix equation (the one which can be solved) as

$$\begin{bmatrix} m_{11} & \ldots & m_{1n} \\ \vdots & \ddots & \vdots \\ m_{n1} & \ldots & m_{nn} \end{bmatrix} \cdot \begin{bmatrix} a_{11} & a_{21} & \ldots & a_{k1} \\ \vdots & \vdots & \vdots & \vdots \\ a_{1n} & a_{2n} & \ldots & a_{kn} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{21} & \ldots & b_{k1} \\ \vdots & \vdots & \vdots & \vdots \\ b_{1n} & b_{2n} & \ldots & b_{kn} \end{bmatrix}.$$

Doing the matrix multiplication gives us the following equation:

$$\begin{bmatrix} m_{11}a_{11} + \cdots + m_{1n}a_{1n} & m_{11}a_{21} + \cdots + m_{1n}a_{2n} & \ldots & m_{11}a_{k1} + \ldots m_{1n}a_{kn} \\ \vdots & \vdots & \vdots & \vdots \\ m_{n1}a_{11} + \cdots + m_{nn}a_{1n} & m_{n1}a_{21} + \cdots + m_{nn}a_{2n} & \ldots & m_{n1}a_{k1} + \cdots + m_{nn}a_{kn} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{21} & \ldots & b_{k1} \\ \vdots & \vdots & \vdots & \vdots \\ b_{1n} & b_{2n} & \ldots & b_{kn} \end{bmatrix}.$$

[18] The size of the icosahedral orbit of a 6-dimensional vector could in theory be any factor of 60, however in application they have sizes 12, 20, and 30. If we assume they have all of size 20, then when $n = 3$, the size of the cartesian product is $20^{2(n+1)} = 25,600,000,000$. This is infeasible for any computer to fully check in a reasonable amount of time.

[19] The converse of this proposition is not true. Consider the example where $M = \begin{bmatrix} x \end{bmatrix}$, $\mathbf{a}_1 = \mathbf{a}_2 = \begin{bmatrix} 1 \end{bmatrix}$, $\mathbf{b}_1 = \begin{bmatrix} 1 \end{bmatrix}$, and $\mathbf{b}_2 = \begin{bmatrix} 2 \end{bmatrix}$. Then $M \cdot \mathbf{a}_1 = \mathbf{b}_1$ and $M \cdot \mathbf{a}_2 = \mathbf{b}_2$ can be solved, but $M \cdot \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 \end{bmatrix}$ cannot be solved.

Since we know that this equation has a solution, then for any $i \in \{1, 2, \ldots, k\}$ the equation

$$\begin{bmatrix} m_{11}a_{i1} + \cdots + m_{1n}a_{in} \\ \vdots \\ m_{n1}a_{i1} + \cdots + m_{nn}a_{in} \end{bmatrix} = \begin{bmatrix} b_{i1} \\ \vdots \\ b_{in} \end{bmatrix}$$

can be solved with the same solution (i.e. same choices for each m_{ij}). All equations of this form are equivalent to $M\mathbf{a}_i = \mathbf{b}_i$. ■

Proposition 17 (Contrapositive of Matrix Equation Solving)**.** *Let M be an $n \times n$ matrix with variables. Let $\mathbf{a}_1, \mathbf{a}_2, \ldots \mathbf{a}_k, \mathbf{b}_1, \mathbf{b}_2, \ldots \mathbf{b}_k$ be n-dimensional vectors. If there exists $i \in \{1, 2, \ldots k\}$ such that the equation*

$$M\mathbf{a}_i = \mathbf{b}_i$$

cannot be solved, then the equation

$$M \cdot \begin{bmatrix} | & | & \cdots & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_k \\ | & | & \cdots & | \end{bmatrix} = \begin{bmatrix} | & | & \cdots & | \\ \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_k \\ | & | & \cdots & | \end{bmatrix}$$

cannot be solved.

What this proposition tells us is that if we find two vectors $\mathbf{v}$ and $\mathbf{u}$ such that $T\mathbf{v} = \mathbf{u}$ cannot be solved, then any matrix equation of the form

$$T \cdot \begin{bmatrix} | & | & \cdots & | & | & | & \cdots & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_{i-1} & \mathbf{v} & \mathbf{a}_{i+1} & \cdots & \mathbf{a}_k \\ | & | & \cdots & | & | & | & \cdots & | \end{bmatrix} = \begin{bmatrix} | & | & \cdots & | & | & | & \cdots & | \\ \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_{i-1} & \mathbf{u} & \mathbf{b}_{i+1} & \cdots & \mathbf{b}_k \\ | & | & \cdots & | & | & | & \cdots & | \end{bmatrix}$$

cannot be solved. What we have done is by finding a simple matrix equation that cannot be solved, we can find a whole class of matrix equations that cannot be solved. This proposition will allow us to build up the B_0 and B_1 matrices one vector at time.[20]

5.1 A Non-Exhaustive Approach Using C++

The first programming approach is to employ guess and check strategy. We used C++ along with the `Eigen` library for fast linear algebra.[21] We use a guess and check approach when using C++ for a few reasons:

1. When first approaching the problem, it seemed intractable and difficult to solve with anything except a guess and check approach. The approach that [ICS+12] describe using is a brute force approach, and so this is where we started for computing transitions.
2. C++ is fast, especially when using code that can run in parallel.
3. There are few well-documented C++ libraries that can do symbolic mathematics.[22]

The idea with this approach is that instead of solving the equation $TB_0 = B_1$, we create a set of candidate transition matrices, lets call this set $\mathcal{T}$. Thus if we want to find a transition from point arrays $P \to Q$ instead of solving equations that are represented by $\text{pam}(P) \times \text{pam}(Q)$, we look at the set $\mathcal{T} \times \text{pam}(P)$ and see if the product $TB_0 \in \text{pam}(Q)$. The question is how do we decide what matrices to put in the set $\mathcal{T}$. The answer is through entry sampling.[23]

[20] See sections 5.2.1 on page 19 and 5.2.2 on page 20.

[21] See [GJ+10] for more on the `Eigen` library.

[22] At the time of coding we did not know about the GiNaC framework for C++ and other similar libraries (see `https://en.cppreference.com/w/cpp/links/libs`). Many of the C++ symbolic libraries do not have the greatest documentation in comparison to the `Eigen` library, which made it easier to quickly learn and implement and get results. If we had known about it at the time, instead of two programs, there would've been likely only one program in C++ using GiNaC.

[23] Note that while this method allows us to find transitions, it cannot tell us definitively whether a transition does not exist. In order to do so with this method we would have to check if $TB_0 \in \text{pam}(Q)$ for all possible transitions T. But since there are an infinite number of transitions T we cannot check this computationally.

5.1.1 Entry Sampling

The idea for entry sampling is as follows: since we are looking at the matrix equation $TB_0 = B_1$, if we had that B_0 and B_1 were invertible, then we could reframe the equation as $T = B_1 B_0^{-1}$. Thus instead of solving for T we could consider all products of the form $B_1 B_0^{-1}$ and see if they fit any of the general matrix forms given by $Z(\mathcal{I}), Z(A_4), Z(D_{10}),$ or $Z(D_6)$. This way of finding transitions by itself is completely ineffective. However, because we have $T = B_1 B_0^{-1}$, we notice that the entries of the T matrix must be a equal to the entries of the product $B_1 B_0^{-1}$. If we take a random sample of matrices $\mathcal{B} \subseteq \text{pam}(P) \times \text{pam}(Q)$, then we can look at the union of all entries that occur in the products $B_1 B_0^{-1}$. This would be formulated as

$$\text{Ent}(T) \subseteq \bigcup_{(B_0, B_1) \in \mathcal{B}} \text{Ent}(B_1 B_0^{-1})$$

where $\text{Ent}(M)$ is the set of entries of a matrix M. This union would be the set of entries that could make up a transition matrix.

Now that we have an idea of what possible values the T matrix could have, we can now generate the $\mathcal{T}$ set by taking a cartesian product of the entries. As an example, we use the point arrays for CCMV. While the point arrays for CCMV are one-base point arrays (and so the B_0 and B_1 matrices are 6×2), we can make B_0 and B_1 matrices that are invertible by repeating the base vector multiple times. That is, we transform the generating list $(\mathbf{t}, \mathbf{v})$ into $(\mathbf{t}, \mathbf{v}, \mathbf{v}, \mathbf{v}, \mathbf{v}, \mathbf{v})$. If we take a random B_0 and B_1 for CCMV we get the following example.

$$B_1 B_0^{-1} = \begin{bmatrix} 0 & 0 & 1 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & -1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} -0.5 & -0.5 & 0.5 & 0.5 & 0.5 & -0.5 \\ -0.5 & 0.5 & -0.5 & 0.5 & -0.5 & 0.5 \\ -0.5 & 0.5 & 0.5 & -0.5 & 0.5 & -0.5 \\ -0.5 & -0.5 & 0.5 & 0.5 & -0.5 & -0.5 \\ 0.5 & 1.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & -0.5 & -0.5 & -0.5 & 0.5 & -0.5 \end{bmatrix}^{-1} = \begin{bmatrix} -0.5 & -1.5 & 0.5 & -0.5 & -0.5 & -1.5 \\ -0.5 & -0.5 & -0.5 & 0.5 & 0.5 & 0.5 \\ -0.5 & 1.5 & -0.5 & 1.5 & 0.5 & 1.5 \\ 0 & 1.0 & 0 & 1.0 & 1.0 & 1.0 \\ -1.0 & 0 & 1.0 & 2.0 & 1.0 & 1.0 \\ -0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{bmatrix}$$

Notice $B_1 B_0^{-1}$ does not have the correct form of any symmetry preserving transition. However, we notice that the entries of a transition matrix will likely have a nonempty intersection with the entries of $B_1 B_0^{-1}$. From this example we see that the set of entries of $B_1 B_0^{-1}$ is the set $\{2, \pm 1.5, \pm 1, \pm 0.5, 0\}$.

Let's say we wanted to generate a list of candidate matrices that preserve icosahedral symmetry with the entries $\{-1, -0.5, 0, 0.5, 1\}$. Then we take the cartesian product $\{-1, -0.5, 0, 0.5, 1\} \times \{-1, -0.5, 0, 0.5, 1\}$ and generate transition matrices of the correct form (since the icosahedral general form has two variables, we only multiply the set twice). This would give us the following set for $\mathcal{T}$:

$$\mathcal{T} = \left\{ \begin{bmatrix} -1 & -1 & 1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 \\ -1 & 1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0.5 & -0.5 & -0.5 & 0.5 & 0.5 \\ 0.5 & -1 & 0.5 & -0.5 & -0.5 & 0.5 \\ -0.5 & 0.5 & -1 & 0.5 & -0.5 & 0.5 \\ -0.5 & -0.5 & 0.5 & -1 & 0.5 & 0.5 \\ 0.5 & -0.5 & -0.5 & 0.5 & -1 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & -1 \end{bmatrix}, \right.$$
$$\left. \ldots, \begin{bmatrix} 1 & 0.5 & -0.5 & -0.5 & 0.5 & 0.5 \\ 0.5 & 1 & 0.5 & -0.5 & -0.5 & 0.5 \\ -0.5 & 0.5 & 1 & 0.5 & -0.5 & 0.5 \\ -0.5 & -0.5 & 0.5 & 1 & 0.5 & 0.5 \\ 0.5 & -0.5 & -0.5 & 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 & -1 & -1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & 1 \\ -1 & 1 & 1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \right\}$$

This set has 25 matrices, but it will have less than 25 transitions since some matrices will have determinant zero (since $\mathcal{T}$ will at least contain the zero matrix).

Now that we have our set $\mathcal{T}$, we perform the multiplication TB_0 for each $(T, B_0) \in \mathcal{T} \times \text{pam}(P)$ and check whether $TB_0 \in \text{pam}(Q)$. In this case the set $\mathcal{T} \times \text{pam}(P)$ is significantly smaller than $\text{pam}(P) \times \text{pam}(Q)$, which means that we have significantly less equations to check.

While this is nice in the case of preserving icosahedral symmetry, we run into problems when we wish to preserve symmetries of the maximal subgroups. Since the general form for $Z(A_4), Z(D_{10}),$ and $Z(D_6)$ have more variables, then the size of $\mathcal{T}$ grows exponentially. However, in the cases of $Z(D_{10}),$ and $Z(D_6)$, we can reduce the size of $\mathcal{T}$ by using partial transitions.

5.1.2 Partial Transitions

Definition 18 (Partial Transition)**.** Let $\mathcal{G} = \mathcal{I}, A_4, D_{10}, \text{or } D_6$. A *partial transition* is a nonzero element of $Z(\mathcal{G})$ where at least one row of the matrix is all zero.

Another way of thinking about this definition is that we are allowing ourselves to set some of the variables to zero. Notice that there do not exist any partial transitions for $\mathcal{I}$ or A_4, since forcing any row to be zero forces the matrix to be the zero matrix.[24] However, in the cases of D_{10} and D_6, there do exist partial transitions, and they turn out to be useful in finding which transitions to put in our set $\mathcal{T}$.
For D_{10} we notice the following:

$$T = \begin{bmatrix} z & x & y & y & x & t \\ x & z & x & y & y & t \\ y & x & z & x & y & t \\ y & y & x & z & x & t \\ x & y & y & x & z & t \\ u & u & u & u & u & w \end{bmatrix} = \begin{bmatrix} z & x & y & y & x & t \\ x & z & x & y & y & t \\ y & x & z & x & y & t \\ y & y & x & z & x & t \\ x & y & y & x & z & t \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ u & u & u & u & u & w \end{bmatrix}.$$

And similarly for D_6 we notice the following:

$$T = \begin{bmatrix} u & w & -w & x & s & s \\ -t & y & v & -v & z & -t \\ t & v & y & v & t & -z \\ z & -v & v & y & -t & -t \\ s & x & -w & w & u & s \\ s & w & -x & w & s & u \end{bmatrix} = \begin{bmatrix} u & w & -w & x & s & s \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ s & x & -w & w & u & s \\ s & w & -x & w & s & u \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ -t & y & v & -v & z & -t \\ t & v & y & v & t & -z \\ z & -v & v & y & -t & -t \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

With this idea in mind, we can create a couple of useful propositions that use partial transitions.

Lemma 19 (Norm Condition for Transitions)**.** *If T is a transition from point array $P \to Q$, then for all $\mathbf{v} \in P$ we have that*

$$\|T\mathbf{v}\| \leq \max_{\mathbf{u} \in Q} \|\mathbf{u}\|$$

where $\|\mathbf{u}\|$ is the standard Euclidean norm.

Proof. Since T is a transition from P to Q, therefore for each $\mathbf{v} \in P$ we have that $T\mathbf{v} = \mathbf{u}$ for some $\mathbf{u} \in Q$. Thus $\|T\mathbf{v}\| = \|\mathbf{u}\|$ and so $\|T\mathbf{v}\| \leq \max_{\mathbf{u} \in Q} \|\mathbf{u}\|$. ■

Proposition 20 (Norm Condition for Partial Transitions)**.** *Suppose a transition matrix T decomposes into two partial transitions such that $T = A + B$. If T is a transition from point array $P \to Q$, then for every $\mathbf{v} \in P$ we have that*

$$\|A\mathbf{v}\| \leq \max_{\mathbf{u} \in Q} \|\mathbf{u}\| \text{ and } \|B\mathbf{v}\| \leq \max_{\mathbf{u} \in Q} \|\mathbf{u}\|$$

where $\|\mathbf{u}\|$ is the standard Euclidean norm.

Proof. Since partial transitions have at least one row of zeroes, we have that $\|A\mathbf{v}\| \leq \|T\mathbf{v}\|$ and $\|B\mathbf{v}\| \leq \|T\mathbf{v}\|$. The result then follows from Lemma 19. ■

Lemma 21. *If T is a transition from point array $P \to Q$, then for every $\mathbf{v} \in P$ there exists $\mathbf{u} \in Q$ such that $T\mathbf{v} = \mathbf{u}$.*

[24] Recall we require our transition matrices to be invertible.

Proof. Since transitions map one point array onto another, it must be the case that for every $\mathbf{v} \in P$ there exists $\mathbf{u} \in Q$ such that $T\mathbf{v} = \mathbf{u}$. ■

In order to state the next proposition, we need to understand how the partial transitions work with vector multiplication. Let us use the D_6 partial transition as an example. Suppose $T \in Z(D_6)$ such that $T\mathbf{v} = \mathbf{u}$. Then we have that

$$\begin{bmatrix} u & w & -w & x & s & s \\ -t & y & v & -v & z & -t \\ t & v & y & v & t & -z \\ z & -v & v & y & -t & -t \\ s & x & -w & w & u & s \\ s & w & -x & w & s & u \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix}$$

If we use the partial transition decomposition then the previous equation turns into

$$\left(\begin{bmatrix} u & w & -w & x & s & s \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ s & x & -w & w & u & s \\ s & w & -x & w & s & u \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ -t & y & v & -v & z & -t \\ t & v & y & v & t & -z \\ z & -v & v & y & -t & -t \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \right) \cdot \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix}$$

which simplifies to

$$\begin{bmatrix} a_1 \\ 0 \\ 0 \\ 0 \\ a_5 \\ a_6 \end{bmatrix} + \begin{bmatrix} 0 \\ a_2 \\ a_3 \\ a_4 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix}$$

In addition to decomposing the transition matrix, they also decompose the result of any vector multiplied by the transition matrix! So we can check both decompositions separately. In order to state this mathematically, we need to create partial identity matrices, which are identity matrices which have zero rows in same places as the partial transition matrix. For D_6 we decompose the identity matrix as

$$I_6 = I_A + I_B \iff \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Therefore instead of checking whether $T\mathbf{v} = \mathbf{u}$ can be solved, we check whether both $A\mathbf{v} = I_A\mathbf{u}$ and $B\mathbf{v} = I_B\mathbf{u}$ can be solved.

Proposition 22 (Partial Transition Mapping Condition)**.** *Suppose a transition matrix T decomposes into two partial transitions such that $T = A + B$. Let I_A and I_B be the corresponding identity decomposition such that $I_AT = A$ and $I_BT = B$. If T is a transition from point array $P \to Q$, then for every $\mathbf{v} \in P$ there exists $\mathbf{u} \in Q$ such that*

$$A\mathbf{v} = I_A\mathbf{u} \text{ and } B\mathbf{v} = I_B\mathbf{u}.$$

Proof. Since T is a transition from P to Q, then for each $\mathbf{v} \in P$ there exists $\mathbf{u} \in Q$ such that $T\mathbf{v} = \mathbf{u}$. Because our transition T decomposes into two partial transitions A and B, we have that $(A + B)\mathbf{v} = \mathbf{u}$. Taking the corresponding partial identity matrices I_A and I_B, we have that $(A+B)\mathbf{v} = A\mathbf{v}+B\mathbf{v} = I_A\mathbf{u}+I_B\mathbf{u}$. We then must have that $A\mathbf{v} = I_A\mathbf{u}$ and $B\mathbf{v} = I_B\mathbf{u}$ because the I_A and I_B matrices have the same zero rows as A and B respectively by construction. ■

5.1.3 Contrapositive

While the previous two propositions are useful, they don't help us find transitions, since they both require us to have already found a transition. What is useful is their contrapositives, which will help us eliminate possible transitions.

Proposition 23 (Contrapositive of Proposition 20)**.** *Suppose a transition matrix T decomposes into partial transitions such that $T = A + B$. If there exists $\mathbf{v} \in P$ such that either*

$$\|A\mathbf{v}\| > \max_{\mathbf{u}\in Q} \|\mathbf{u}\| \text{ or } \|B\mathbf{v}\| > \max_{\mathbf{u}\in Q} \|\mathbf{u}\|$$

then T is not a transition for $P \to Q$.

Proposition 24 (Contrapositive of Proposition 22)**.** *Suppose a transition matrix T decomposes into partial transitions such that $T = A + B$. Let I_A and I_B be the corresponding identity decomposition such that $I_A T = A$ and $I_B T = B$. If there exists $\mathbf{v} \in P$ such that for every $\mathbf{u} \in Q$ either*

$$A\mathbf{v} \neq I_A\mathbf{u} \text{ or } B\mathbf{v} \neq I_B\mathbf{u}$$

then T is not a transition for $P \to Q$.

Both of these propositions have the same idea in mind, we can say a matrix T cannot be a transition based upon what it does to a particular vector. Furthermore, by using the partial transition decomposition idea, we can determine subsets of candidate transition matrices that cannot be transitions.

Proposition 23 allows us to determine when a matrix T cannot be a transition by using the fact that transition matrices must map one point array onto another. We could check whether $T\mathbf{v} = \mathbf{u}$ for every combination of $\mathbf{v} \in P$ and $\mathbf{u} \in Q$ but in order to do this we must check $|P| \cdot |Q|$ things. But we can make the following observation: If $\|T\mathbf{v}\| > \|\mathbf{u}\|$ then $T\mathbf{v} \neq \mathbf{u}$. Thus if $\|T\mathbf{v}\| > \max_{\mathbf{u}\in Q} \|\mathbf{u}\|$ then $T\mathbf{v} \neq \mathbf{u}$ for *every* $\mathbf{u} \in Q$ and so we cannot map P onto Q. The advantage of this method is that we only need to check $|P|$ things since the $\max_{\mathbf{u}\in Q} \|\mathbf{u}\|$ is simply a constant for a given point array Q. And since $\|A\mathbf{v}\| \leq \|T\mathbf{v}\|$ and $\|B\mathbf{v}\| \leq \|T\mathbf{v}\|$ we actually only need to check this with partial transitions.

We use proposition 24 only if proposition 23 fails to tell us that T is not transition. This proposition essentially checks whether $T\mathbf{v} = \mathbf{u}$ for every combination of $\mathbf{v} \in P$ and $\mathbf{u} \in Q$. We want to do this last since otherwise we lose the speedup that the previous condition allows for by precomputing the maximal norm of a point array.

5.1.4 Parallelization

We can easily parallelize this problem. Since our approach here is to take a list of potential transition matrices, narrow it down through various conditions, and then attempt to find a B_0 and B_1 matrix, we can parallelize when attempting to find a B_0 and B_1 matrix. The idea is to use task-based parallelism.[25] Each task for this program has the following form: "For a potential transition matrix T, see if there exists a B_0 and B_1 matrix such that $TB_0 = B_1$." To actually parallelize this in C++ programming language we use the `OpenMP` API in order to implement task based parallelism. Therefore the C++ program has the following structure in order to find transitions from $P \to Q$:

1. Generate a list potential transition matrices called C.
2. Create task for each T matrix in C. Each task will be answering the yes/no question: Is T a transition matrix for $P \to Q$?
3. Process all tasks.
4. Collect all data from tasks and output which T matrices (if any) are transitions for $P \to Q$.

[25] See chapter 1 of [Pac11] for more on task-based parallelism.

The C++ program allowed us to get a basic understanding of the problem and generate some transitions between real point arrays. However, this program is not entirely optimal. The primary issue is that this program cannot say for certain whether no transition exists or if we simply haven't found a transition yet. Additionally, it still is slow in many cases despite all of the speed ups gaining from our conditions and parallelization. These issues will be largely fixed by our program in Python.

5.2 An Exhaustive Approach Using Python

The next programming approach is to actually solve the matrix equations. For this we use the Python programming language along with the `sympy` library, a symbolic mathematics library for Python.[26] This program aims to solve the linear matrix equations in an efficient way that the C++ program couldn't do. The primary idea is that now that we can solve equations and determine if no solution exists, we can reduce the number of equations we need to check.

5.2.1 Pair Checking

Recall proposition 16 on page 13 and its contrapositive. This proposition is telling us that what matters for solving a matrix equation is how the columns of B_0 and B_1 correspond with each other. Thus in a way, each column is independent of each other column. Let us then consider point arrays P and Q. Suppose P is minimally generated by $(\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n)$ and Q is minimally generated by vectors $(\mathbf{t}', \mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n)$. For simplicity, let us fix the order of the generators. Then the set of equations that describe a transition from $P \to Q$ are of the form

$$T \cdot \begin{bmatrix} | & | & | & \cdots & | \\ \mathbf{t} & \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ | & | & | & \cdots & | \end{bmatrix} = \begin{bmatrix} | & | & | & \cdots & | \\ \mathbf{t}' & \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n \\ | & | & | & \cdots & | \end{bmatrix}.$$

While this is general form, we note that since $(\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n)$ minimally generates P, then $(G_0\mathbf{t}, G_1\mathbf{v}_1, G_2\mathbf{v}_2, \ldots, G_n\mathbf{v}_n)$ also minimally generates P, where $G_i \in \mathcal{I}$. Thus the general form of an equation that describes a transition from $P \to Q$ has the form

$$T \cdot \begin{bmatrix} | & | & | & \cdots & | \\ G_0\mathbf{t} & G_1\mathbf{v}_1 & G_2\mathbf{v}_2 & \cdots & G_n\mathbf{v}_n \\ | & | & | & \cdots & | \end{bmatrix} = \begin{bmatrix} | & | & | & \cdots & | \\ H_0\mathbf{t}' & H_1\mathbf{u}_1 & H_2\mathbf{u}_2 & \cdots & H_n\mathbf{u}_n \\ | & | & | & \cdots & | \end{bmatrix}$$

where G_i and H_i are all elements of $\mathcal{I}$. But because of proposition 16, we should first look at the equations

$$\begin{aligned} TG_0\mathbf{t} &= H_0\mathbf{t}' \\ TG_1\mathbf{v}_1 &= H_1\mathbf{u}_1 \\ &\vdots \\ TG_n\mathbf{v}_n &= H_n\mathbf{u}_n. \end{aligned}$$

Because of the contrapositive of proposition 16, this tells us we ought to find pairs of (G_i, H_i) such that the equation $TG_i\mathbf{v}_i = H_i\mathbf{u}_i$ has a solution (and of course changing substituting in the translation vectors if $i = 0$). Therefore we will create many subsets of $\mathcal{I} \times \mathcal{I}$.

$$\begin{aligned} TG_0\mathbf{t} = H_0\mathbf{t}' &\implies \{(G, H) \in \mathcal{I} \times \mathcal{I} \mid TG\mathbf{t} = H\mathbf{t}' \text{ has a solution.}\} \\ TG_1\mathbf{v}_1 = H_1\mathbf{u}_1 &\implies \{(G, H) \in \mathcal{I} \times \mathcal{I} \mid TG\mathbf{v}_1 = H\mathbf{u}_1 \text{ has a solution.}\} \\ &\vdots \\ TG_n\mathbf{v}_n = H_n\mathbf{u}_n &\implies \{(G, H) \in \mathcal{I} \times \mathcal{I} \mid TG\mathbf{v}_n = H\mathbf{u}_n \text{ has a solution.}\} \end{aligned}$$

[26] See [MSP+17] for more on the `sympy` library.

This gets our idea solidified, but our icosahedral orbits of vector (i.e. $\mathcal{I}\mathbf{v}$) are smaller than the icosahedral group. So it is more efficient to directly look at the cartesian product of icosahedral orbits.

$$\begin{aligned} TG_0\mathbf{t} = H_0\mathbf{t}' &\implies \{(\mathbf{t}_0, \mathbf{t}_0') \in \mathcal{I}\mathbf{t} \times \mathcal{I}\mathbf{t}' \mid T\mathbf{t}_0 = \mathbf{t}_0' \text{ has a solution.}\} \\ TG_1\mathbf{v}_1 = H_1\mathbf{u}_1 &\implies \{(\mathbf{v}, \mathbf{u}) \in \mathcal{I}\mathbf{v}_1 \times \mathcal{I}\mathbf{u}_1 \mid T\mathbf{v} = \mathbf{u} \text{ has a solution.}\} \\ &\vdots \\ TG_n\mathbf{v}_n = H_n\mathbf{u}_n &\implies \{(\mathbf{v}, \mathbf{u}) \in \mathcal{I}\mathbf{v}_n \times \mathcal{I}\mathbf{u}_n \mid T\mathbf{v} = \mathbf{u} \text{ has a solution.}\} \end{aligned}$$

Let us make a definition for these new sets we've constructed.

Definition 25 (Icosahedral Vector Pair Set)**.** Let $\mathcal{G} = \mathcal{I}, A_4, D_{10}, \text{or } D_6$ and T be the general form of $Z(\mathcal{G})$. Then the *icosahedral vector pair set* for vectors $\mathbf{v}$ and $\mathbf{u}$ is defined as

$$V(\mathbf{v}, \mathbf{u}) := \{(\mathbf{v}', \mathbf{u}') \in \mathcal{I}\mathbf{v} \times \mathcal{I}\mathbf{u} \mid T\mathbf{v}' = \mathbf{u}' \text{ has a solution.}\}$$

With this structure in mind, we can describe how the Python program finds transitions. Let P and Q be point arrays where P is minimally generated by $(\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n)$ and Q is minimally generated by vectors $(\mathbf{t}', \mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n)$. Let T be general form for whichever symmetry we wish to preserve. First, create all of the vector pair sets $V(\mathbf{t}, \mathbf{t}'), V(\mathbf{v}_1, \mathbf{u}_1), \ldots, V(\mathbf{v}_n, \mathbf{u}_n)$. Now we want to search through the cartesian product $V(\mathbf{t}, \mathbf{t}') \times V(\mathbf{v}_1, \mathbf{u}_1) \times \cdots \times V(\mathbf{v}_n, \mathbf{u}_n)$. This cartesian product can be mapped onto all possible point array matrices for P and Q that can have a solution.[27] The mapping we use is

$$((\mathbf{t}_0, \mathbf{t}_0'), (\mathbf{v}_1, \mathbf{u}_1), \ldots, (\mathbf{v}_n, \mathbf{u}_n)) \mapsto \left(\begin{bmatrix} | & | & | & \cdots & | \\ \mathbf{t}_0 & \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ | & | & | & \cdots & | \end{bmatrix}, \begin{bmatrix} | & | & | & \cdots & | \\ \mathbf{t}_0' & \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n \\ | & | & | & \cdots & | \end{bmatrix} \right)$$

However, there is an issue with this map, which is that there might not be a shared solution between all the vector pairs. By construction of the icosahedral vector pair sets we are guaranteed that each column can be mapped to its corresponding column with a T matrix, but it is not guaranteed that there exists a shared T matrix that can map every column to its corresponding column.[28] Thus as a last step we need to check whether the equation

$$T \cdot \begin{bmatrix} | & | & | & \cdots & | \\ \mathbf{t}_0 & \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ | & | & | & \cdots & | \end{bmatrix} = \begin{bmatrix} | & | & | & \cdots & | \\ \mathbf{t}_0' & \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n \\ | & | & | & \cdots & | \end{bmatrix} \tag{3}$$

can be solved for T.

This method will find us a transition if one exists, however, this method is still largely a brute force approach and quickly becomes inefficient. We can drastically improve the efficiently of the program by implementing a depth first search with pruning.

5.2.2 Depth first search

Our motivation for a depth first search is that we wish to build up the B_0 and B_1 matrices one vector at a time. Building up the matrices one vector at a time looks like the following procedure: If we have a equation of the form

$$T \cdot \begin{bmatrix} | & | & | & \cdots & | \\ \mathbf{t} & \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ | & | & | & \cdots & | \end{bmatrix} = \begin{bmatrix} | & | & | & \cdots & | \\ \mathbf{t}' & \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n \\ | & | & | & \cdots & | \end{bmatrix}$$

[27] Since by construction of the icosahedral vector pair sets, we eliminate all possible pairs of point array matrices for which proposition 17 on page 14 applies.

[28] See proposition 16 on page 13 and footnote 19 on page 13.

that can be solved, then in order to add one vector we need to figure out if the equation

$$T \cdot \begin{bmatrix} | & | & | & \cdots & | & | \\ \mathbf{t} & \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n & \mathbf{v}_{n+1} \\ | & | & | & \cdots & | & | \end{bmatrix} = \begin{bmatrix} | & | & | & \cdots & | & | \\ \mathbf{t}' & \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n & \mathbf{u}_{n+1} \\ | & | & | & \cdots & | & | \end{bmatrix}$$

can be solved. When adding a vector to these matrices, it only makes sense to add vectors $\mathbf{v}_{n+1}$ and $\mathbf{u}_{n+1}$ if the equation $T\mathbf{v}_{n+1} = \mathbf{u}_{n+1}$ can be solved because of the Matrix Equation Solving proposition (proposition 16). However, even if this equation can be solved, adding the vector might not work.[29] Figure 6 gives us a visual idea of how this idea of depth first search is used within the program to build up the B_0 and B_1 matrices. We want to traverse this tree in the most efficient way possible. In order to do so, we don't try every possible path and instead only continue down the tree if the equation can be solved with the smaller B_0 and B_1 matrices (this is the "pruning" part of depth first search with pruning). If adding on a vector to the B_0 and B_1 matrices fails, we backtrack up the tree and try a different path. If we exhaust all possibilities within this search tree and do not find any B_0 and B_1 matrices, then we have shown there does not exist a transition between these point arrays.

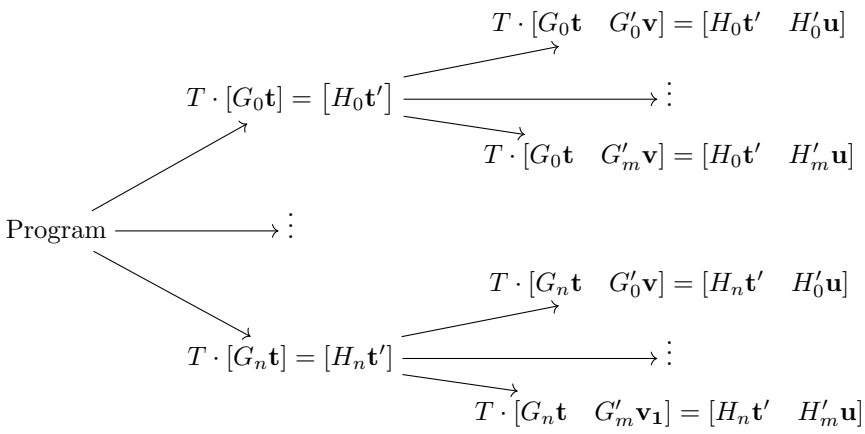


Figure 6: Diagram showing the idea of a depth first search for finding a transition matrix and its point array matrices.

5.2.3 Parallelization

Parallelizing the depth first approach in Python takes more thought than in the C++ approach. What we can do for this approach is to split up the depth first search tree as shown in figure 7 on the next page. Because our goal is to search the entire tree, we split up the tree at the first level. Therefore in our example we would split into multiple trees as shown in figure 7 on the following page. In general, we split at the first level, which means we split based on upon the number of elements in the icosahedral vector pair set $V(\mathbf{t}, \mathbf{t}')$. To implement this form of parallelism, we use the `multiprocessing` library within Python. In fact this parallelism is still task based parallelism, with the task being checking a given depth first search tree to see if there exists B_0 and B_1 matrices such that the equation $TB_0 = B_1$ can be solved for T.

[29] See footnote 28.

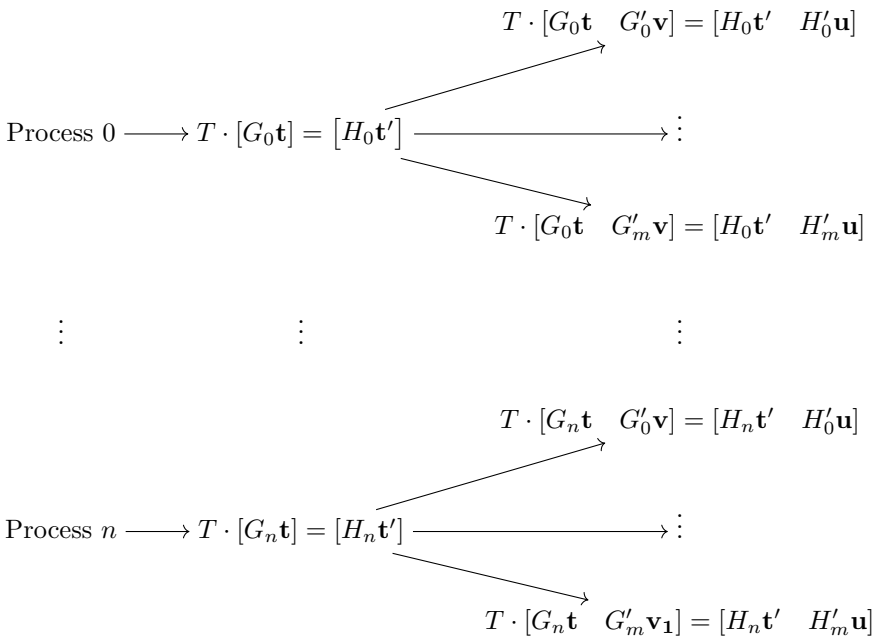


Figure 7: Parallelization of the depth first search.

6 Results

These computational methods have allowed us to find transitions between point arrays that characterize real viruses. An important result we had found early in this process is that we were able to reproduce the same four D_6 transition matrices for the CCMV virus given in [ICS+12]. We also attempted to find transitions between viruses with larger point arrays than CCMV such as the Turnip Crinkle Virus and Escherichia virus HK97. In most cases, it is impossible to find any transition between the native and mature point arrays of viruses with large point arrays. However, what can do is instead see if there exists a transition between the outer layer of these point arrays. The outer layers are themselves point arrays, so we can use the same methodology on these smaller point arrays. These outler layer transitions are much more common than finding transitions between the entire point array.

The most important result from these methods is that we are able to determine whether any symmetry preserving transitions exist between any of the possible 55 standard one-base point arrays.[30] There are $55^2 = 3,025$ possible one-base to one-base point array cases, since we can freely choose any one of the 55 standard for both P and Q. From determining all possible transitions we make the following conclusions for the 55 one-base point arrays:

- It is impossible to have a nontrivial transition that preserves icosahedral symmetry. We can only have an icosahedral transition if the two point arrays are the same (and so T is the identity matrix).
- There exists D_6 transitions for a majority of the possible one-base transitions (3,001 out of 3,025). Interestingly, all cases where there a D_6 transition does not exist involve the point array 24.
- Transitions for D_{10} and A_4 are much less common, with transitions existing only in 1,277 out of 3,025 for D_{10} and 309 out of 3,025 existing for A_4.

[30] See `https://github.com/RochX/virus-research-paper/blob/main/one_base_table.csv` for the table that shows what is possible for a transition from $P \to Q$ when both point arrays are one-base.

While these results are strong, from this we get the stronger result that there do not exist any nontrivial icosahedral transitions between *any* two point arrays. We get this from the fact that any lifted point arrays with any number of base vectors must be built up from these one-base point arrays. Proposition 16 then tells us that in order to have transition between two point arrays, there must exist transitions between all of the one-base point arrays that make up the larger point arrays. This allows us to conclude that there cannot exist any nontrivial icosahedral transitions.

6.1 Future Directions

While finding whether a transition exists for all possible 3,025 one-base to one-base cases gives us a very comprehesive overview, there are still more directions one could take with this research. One could be looking at what these transitions are doing in 3-dimensions. We only found transitions and point arrays in their 6-dimensional form, but we could project them into 3-dimensions with the matrix

$$\begin{bmatrix} \phi & 0 & -1 & 0 & \phi & 1 \\ 1 & \phi & 0 & -\phi & -1 & 0 \\ 0 & 1 & \phi & 1 & 0 & \phi \end{bmatrix}$$

where ϕ is the golden ratio given by $\phi = \frac{1}{2}(1+\sqrt{5})$ [ICS+12].

Another direction would be to investigate all possible two-base point array transitions. These point arrays are minimally generated by vectors $(\mathbf{t}, \mathbf{v}, \mathbf{u})$. While in theory there would be $55^2 = 3,025$ two-base point arrays, in reality there are less than that, because the base vectors must share translation vectors. Notice how the one-base point arrays can be categorized into three categories, based on if the share a translation vector, shown in table 2. We can only combine point arrays

Translation Vector	Point Arrays	Total Number
f	$1-6, 14-21, 31-41$	25
b	$7-10, 22-26, 42-49$	17
s	$11-13, 27-30, 50-55$	13

Table 2: Table describes which point arrays are grouped together in order to make two-base point arrays.[31]

if they share the same translation vector. For example, we can combine point arrays 1 and 2, but not point arrays 1 and 7 since they use different translation vectors. Therefore the total number of two-base point arrays is

$$\frac{25 \cdot 24 + 17 \cdot 16 + 13 \cdot 12}{2} = 514$$

since we don't want to use the same base vector twice and since the order of the generating vectors does not matter. If we wanted to comprehesively check if transitions exist for all two-base point arrays, we would need to check all $514^2 = 264,196$ possible cases, exponentially harder than the 3,025 one-base cases we checked. This would take multiple days using the program described in this paper even with the optimizations used. A future direction could be to search for a more efficient way one could check all of these two-base cases.

7 Acknowledgements

I would like to acknowledge the funding from the Heyl Scholarship Research Fund that allowed to me do this research. I would also like to thank research mentors Dr. Stephen Oloo and Dr. Dave Wilson for their help and support during the research process. Finally, I would like to thank Dr. Dave Wilson and Dr. Sandino Vargas-Perez for allowing me to use the Kalamazoo College supercomputer Jigwe in order to run my code.

[31] See section A.2 for definitions of the translation vectors.

A Appendix

A.1 Extra Information on Point Arrays

When talking about point arrays in this paper, we only considered finding transitions between point arrays whose minimal generating lists are of the same size for simplicity. However, it is often the case that we wish to find transitions between point arrays whose minimal generating lists are not of the same size. Suppose P is minimally generated by $(\mathbf{t}, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n)$ and Q is minimally generated by $(\mathbf{t}', \mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m)$ with $n \neq m$. What we need to do is add vectors to the generating list of the smaller list until they have the same number of vectors. Let's suppose $n < m$. Then to create a generating list for P of size m, it must have the form $(\mathbf{t}, \mathbf{v}_{i_1}, \mathbf{v}_{i_2}, \ldots, \mathbf{v}_{i_m})$ where $i_j \in \{1, 2, \ldots, n\}$ for all j. The idea is that we choose m vectors of the form $\mathbf{v}_k$ such that the generating list creates the point array P. The same idea occurs when $n > m$.

A.2 List of All One-Base Point Arrays

We have stated in this paper that there exist exactly 55 one-base point arrays (i.e. point arrays of the form $\mathcal{I}\mathbf{v} \cup (\mathcal{I}\mathbf{v} + \mathcal{I}\mathbf{t})$). In this section we will explicitly list these 55 point arrays in their lifted point array form. Let the vectors $\mathbf{s}, \mathbf{b}$, and $\mathbf{f}$ be given by

$$\mathbf{s} := \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{b} := \begin{bmatrix} \frac{1}{2} \\ -\frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ -\frac{1}{2} \\ \frac{1}{2} \end{bmatrix} \quad \mathbf{f} := \begin{bmatrix} \frac{1}{2} \\ 0 \\ 0 \\ -\frac{1}{2} \\ 0 \\ 0 \end{bmatrix}$$

and the matrix D be given by

$$D = \frac{1}{2} \begin{bmatrix} 1 & 1 & -1 & -1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & 1 \\ -1 & 1 & 1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Table 3 on the next page gives the list of all 55 one-base point arrays.

A.3 The Code

The C++ and Python code referred to in section 5 can be found at `https://github.com/RochX/virus-research` and `https://github.com/RochX/virus-research-python` respectively. The LaTeX code used to create this paper can be found at `https://github.com/RochX/virus-research-paper`.

Point Array Number	Minimal Generating List	Point Array Number	Minimal Generating List
1	$(\mathbf{f}, D\mathbf{s})$	$\vdots$	$\vdots$
2	$(\mathbf{f}, \frac{1}{2}D^2\mathbf{s})$	29	$(\mathbf{s}, \mathbf{b})$
3	$(\mathbf{f}, \mathbf{s})$	30	$(\mathbf{s}, D^{-1}\mathbf{b})$
4	$(\mathbf{f}, \frac{1}{2}D\mathbf{s})$	31	$(\mathbf{f}, 2D\mathbf{f})$
5	$(\mathbf{f}, \frac{1}{2}\mathbf{s})$	32	$(\mathbf{f}, D^2\mathbf{f})$
6	$(\mathbf{f}, \frac{1}{2}D^{-1}\mathbf{s})$	33	$(\mathbf{f}, 2\mathbf{f})$
7	$(\mathbf{b}, D\mathbf{s})$	34	$(\mathbf{f}, D\mathbf{f})$
8	$(\mathbf{b}, \mathbf{s})$	35	$(\mathbf{f}, 2D^{-1}\mathbf{f})$
9	$(\mathbf{b}, D^{-1}\mathbf{s})$	36	$(\mathbf{f}, \mathbf{f})$
10	$(\mathbf{b}, D^{-2}\mathbf{s})$	37	$(\mathbf{f}, \frac{1}{2}D\mathbf{f})$
11	$(\mathbf{s}, D\mathbf{s})$	38	$(\mathbf{f}, D^{-1}\mathbf{f})$
12	$(\mathbf{s}, \mathbf{s})$	39	$(\mathbf{f}, \frac{1}{2}\mathbf{f})$
13	$(\mathbf{s}, D^{-1}\mathbf{s})$	40	$(\mathbf{f}, D^{-2}\mathbf{f})$
14	$(\mathbf{f}, D^2\mathbf{b})$	41	$(\mathbf{f}, \frac{1}{2}D^{-1}\mathbf{f})$
15	$(\mathbf{f}, \frac{1}{2}D^3\mathbf{b})$	42	$(\mathbf{b}, 2D\mathbf{f})$
16	$(\mathbf{f}, D\mathbf{b})$	43	$(\mathbf{b}, 2\mathbf{f})$
17	$(\mathbf{f}, \frac{1}{2}D^2\mathbf{b})$	44	$(\mathbf{b}, 2D^{-1}\mathbf{f})$
18	$(\mathbf{f}, \mathbf{b})$	45	$(\mathbf{b}, \mathbf{f})$
19	$(\mathbf{f}, \frac{1}{2}D\mathbf{b})$	46	$(\mathbf{b}, 2D^{-2}\mathbf{f})$
20	$(\mathbf{f}, \frac{1}{2}\mathbf{b})$	47	$(\mathbf{b}, D^{-1}\mathbf{f})$
21	$(\mathbf{f}, \frac{1}{2}D^{-1}\mathbf{b})$	48	$(\mathbf{b}, 2D^{-3}\mathbf{f})$
22	$(\mathbf{b}, D^2\mathbf{b})$	49	$(\mathbf{b}, D^{-2}\mathbf{f})$
23	$(\mathbf{b}, D\mathbf{b})$	50	$(\mathbf{s}, 2D\mathbf{f})$
24	$(\mathbf{b}, \mathbf{b})$	51	$(\mathbf{s}, 2\mathbf{f})$
25	$(\mathbf{b}, D^{-1}\mathbf{b})$	52	$(\mathbf{s}, 2D^{-1}\mathbf{f})$
26	$(\mathbf{b}, D^{-2}\mathbf{b})$	53	$(\mathbf{s}, \mathbf{f})$
27	$(\mathbf{s}, D^2\mathbf{b})$	54	$(\mathbf{s}, 2D^{-2}\mathbf{f})$
28	$(\mathbf{s}, D\mathbf{b})$	55	$(\mathbf{s}, D^{-1}\mathbf{f})$
$\vdots$	$\vdots$		

Table 3: All 55 one-base point arrays

B References

[GJ+10] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[ICS+12] Giuliana Indelicato, Paolo Cermelli, David G. Salthouse, Simone Racca, Giovanni Zanzotto, and Reidun Twarock. A crystallographic approach to structural transitions in icosahedral viruses. *J. Math. Biol.*, 64(5):745–773, 2012.

[KT09] Thomas Keef and Reidun Twarock. Affine extensions of the icosahedral group with applications to the three-dimensional organisation of simple viruses. *Journal of mathematical biology*, 59:287–313, 2009.

[LR88] LS Levitov and J Rhyner. Crystallography of quasicrystals; application to icosahedral symmetry. *Journal de Physique*, 49(11):1835–1849, 1988.

[MSP+17] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017.

[Pac11] Peter S. Pacheco. *An introduction to parallel programming*. Morgan Kaufmann, 2011.

[Wil20] David P. Wilson. Unveiling the hidden rules of spherical viruses using point arrays. *Viruses*, 12(4):1–33, 2020.