TU WIEN Informatics

Visualising the Permutation Space

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Bachelorstudium Informatik

eingereicht von

Mag. Christian-Jürgen Gruber
Matrikelnummer 00121092

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Mitwirkung: Associate Prof. Dr.in Renata Georgia Raidou
Univ.Prof. Dipl.-Inform. Dr.rer.nat. Martin Nöllenburg
Projektass. Dipl.-Ing. Alexander Dobler, BSc

Wien, 11. November 2025

Christian-Jürgen Gruber	Eduard Gröller

Informatics

Visualising the Permutation Space

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Informatics

by

Mag. Christian-Jürgen Gruber
Registration Number 00121092

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Assistance: Associate Prof. Dr.in Renata Georgia Raidou
Univ.Prof. Dipl.-Inform. Dr.rer.nat. Martin Nöllenburg
Projektass. Dipl.-Ing. Alexander Dobler, BSc

Vienna, November 11, 2025

Christian-Jürgen Gruber Eduard Gröller

Erklärung zur Verfassung der Arbeit

Mag. Christian-Jürgen Gruber

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel" habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 11. November 2025

Christian-Jürgen Gruber

Abstract

Seriations, i.e. arranging multidimensional data in a list that follows a logical sequence, are part of a technique called Exploratory Data Analysis. They reduce dimensions and give experts a way to assess images and other complex data, enabling them to compare their content and find similarities and differences. But only a few are used at a regular basis, e.g. the historgram seriation. The tool described here tries to add a more wholesome approach by showing all possible seriations in the permutation space. The space grows factorially with the input size, i.e. the number of pixels, and by far not all are meaningful. The goal is to look for patterns in the space, as patterns may indicate further interesting seriations beside the historgram ones. The question is whether one is able to see such patterns when ordering the permutations and/or comparing two ordered spaces. Based on mathematical findings and algorithmic constraints, it will be shown that some patterns can be found even in a simplified setting using images with a size of 3-by-2.

Contents

[1]cf. [Lai88]. The French publication is historically the first to make the connection.

CHAPTER 1

Introduction

When dealing with images, there are various analytical tools to measure and categorize these images. One of the most prominent ones is the historgram and it uses seriations of images, that is, the pixels are put in a list ordered ascendingly by density value thereby reducing the dimension to one. As will be seen the histogram corresponds to a whole set of seriations. This is also done for voxels and is possible for multidimensional data. Here we deal exclusively with 2-D images.

The seriation of an image can be done in different ways and will produce different seriated outcomes. For a histogram, the pixels are placed in a list and ordered by their density value from lowest to highest, resulting in a certain seriation. When using the same image, but ordering the pixels according to the Hilbert curve, the density values play no role in the order, but just the position in the image, resulting in a different seriation of the same image. Both seriations reduce dimensions and presuppose some order. Emphasising the different properties, leads to divergent and informal definitions.

Definition 1.0.1 (Seriation: Proposal A)**.** x is a seriation if and only if x is a bijective function from an n-dimensional to the 1-dimensional space, where $n \in \mathbb{N}$

Definition 1.0.2 (Seriation: Proposal B)**.** If we see it as a set of points, then x is seriation iff x is a bijective function from a given set onto itself, leading us directly to the definition of a permutation.[1]

Proposal B is the preferred one in our context and serves as basis for the formal definition.

The two serations considered here represent the same image, but with a different pixel order. They are part of the permutation space that can be spawn from the first seriation by enumerating all different orderings of the image's pixels. We use the aforementioned

[1]For a more formal approach see Chapter 2

well known seriations to analyse images. Still, there are a lot of possible seriations and the question is whether there cannot be others that can be as good for an analysis as these. Finding new seriations that can be useful in analysing images, has not been largely considered because the space of possible seriations, namely the permutation space, grows factorially with the input size. The proposed way to find new ones will be through patterns in the permutation space that can show us clusters of interesting serations. These will have to be investigated later on in order to estimate their usefulness.

The idea of reordering something in many ways can be traced back to Roman times and hte link will be established in Section 1.1 by referrring to Quntilian, a Latin grammarian. Chapter 2 on Mathematical Preliminaries helps us set the base for understanding the algorithms and the visualisations as well as showing the constraints and possibilities to overcome them. Thereafter, we focus on what the visualisation of the permutation space can and shall do, as well as what patterns there are, and what the user can do (see Chapter 3). Very shortly, the architecture, the algorithms and the sites, where the software can be downloaded from, will be shown in section 3.4. Every software needs its test cases to illustrate the correctness and in our case test images to look for patterns (see Chapter 4). Having set the environment, we will discuss the question, whether patterns can be seen (see Section 4.1). In the last Chapter (5), we will round up the discussion on the permutation space, its possible patterns, and an outlook for larger spaces.

1.1 The Origin of *Permutation*

Permutation stems from the latin word *permutatio* which itself is a noun built from the verb *permutare*. *permutare* consists of a prefix *per* and the verb *mutare*. *mutare* is normally translated as change, and the prefix makes the meaning stronger, leading to it to change thoroughly.[2] In general, it means a change, an alteration, or barter. But that has nothing to do with the term as used here. A further search allows for one mention of *permutatio* that is very close to the meaning used here. Quintilian speaks in his *Institutio Oratoria*, an introduction to the art of rhethorics, not only about rhetorical techniques, but at the beginning in the first chapter also about teaching children to read and write. In section 1.1.24-25 he writes the following:

> “neque enim mihi illud saltem placet, quod fieri in plurimis video, ut litterarum nomina et contextum prius quam formas parvuli discant. obstat hoc agnitioni earum non intendentibus mox animum ad ipsos ductus, dum antecedentem memoriam sequuntur. quae causa est praecipientibus, ut etiam, cum satis adfixisse eas pueris recto illo quo primum scribi solent contextu videntur, retro agant rursus et **varia permutatione** turbent, donec litteras qui instituuntur facie norint non ordine. quapropter optime sicut hominum pariter et habitus et nomina edocebuntur.”([Qui]) [3]

[2] For more information on *permutare* see [pera]. For the noun *permutatio*, see [perb].

[3] Here comes the translation as found in Perseus: “At any rate I am not satisfied with the course

The pupils are taught the letters in a certain order and by learning them in this order they are not induced to learn the shapes exactly as they already know which letter comes next. And Quntilian goes further in stating that the same teachers must then resort to giving the pupils the letters in reverse order and to making various reorderings of the letters in order for pupils to learn the shapes as well. After all, pupils need to learn how to recognise the letters in various surroundings.

There we can find all the properties of a permutation. First, there is the alphabet, the set of characters, and the first permutation, which is not called a *permutatio*, is the order of the alphabet. But all the other reorderings of the letters are called exactly that, in fact denoting a bijective function from the set of letters onto itself.

(which I note is usually adopted) of teaching small children the names and order of the letters before their shapes. [25] Such a practice makes them slow to recognise the letters, since they do not pay attention to their actual shape, preferring to be guided by what they have already learned by rote. It is for this reason that teachers, when they think they have sufficiently familiarised their young pupils with the letters written in their usual order, reverse that order or rearrange it in every kind of combination, until they learn to know the letters from their appearance and not from the order in which they occur. It will be best therefore for children to begin by learning their appearance and names just as they do with men." See [Qun]

CHAPTER 2

Mathematical Preliminaries

The use of permutations and the attempt, to not only measure the permutation space, but to visualise it, needs mathematical context. Seriations have something to do with permutations and the connections between the two will be shown, based on their formal definitions.

For the next part, the permutation space grows very fast and visualising it has its limits. As soon as the size of the permutation space reaches over a 1000 possibilities a visualisation faces the limits of the screen size. The same problem occurs if the cardinality of the underlying set gets too large to be shown on a screen. Still more pressing in this case are the limits of the human visual perception. These stumbling blocks will be dealt with in Section 3.

The fast growing space has a more pressing issue: its computational viability. As we will see, computing all the permutations is NP-complete and part of this section will deal with possibilities to give an overview of the permutation space of any set A where $|A|$ leads to a permutation space P too big to be calculated in certain time and space limits.

2.1 n-Seriation

Broadly speaking a seriation puts elements of a set in some kind of order. In archaeology, for example, seriation is a method to analyse the findings by ordering them chronologically, as far as that is possible. In historical sciences finding the correct chronology can be difficult and there often remains some uncertainty.

Now we can abstract from the details in the example and define seriation. A seriation can be thought of as one n-tuple from a set T of n-tuples or as a set A and an order relation R defined on the set A. The n-tuples can be derived from any set.

Definition 2.1.1 (Seriation)**.** S is a seriation on a set $A \iff S$ is a linear order on A.

A seriation is therefore an ordering of a given set that satisfies the trichotomy, that either aSb, $a = b$ or bSa holds for all a, $b \in A$. (cf. [Das14, 22])

When working with black-white images the set A is a set of tuples (a, b) where a is a coordinate tuple (x, y) and b is an integer value, more precise $b \in \mathbb{N}\ \ \&\ \ 0 \leq b \leq 255$. It can be ordered according to the value of each object or the location or both. An image is a two-dimensional object and serialising it, i.e. creating a seration A, transforms it into a one-dimensional object, Switch can be seen as a dimensional reduction, thereby simplifying the data, but also changing the information we can gather from the higher-dimensional ordering.

2.2 Permutation

A permutation is a bijective function from the set A to itself.

Definition 2.2.1 (Permutation)**.** x is a permutation $\iff$ x is a bijective function σ: $A \to A$.

In our case A is not the set of density values, as using the density values (for a greyscale image with more than 256 pixels, at least two pixels must have the same density value), would create ambiguity, but the set of unique indices from the seriation.

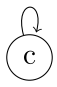
2.3 Notational Remarks

There are three common notations for permutations based on its definition as a bijective function. As a basis, we will use the following n-tuple (a, b, c, d, f, g). A permutation σ can be noted in

1. a Cauchy or two line notation: $\sigma = \begin{pmatrix} a & b & c & d & f & g \\ b & d & c & f & g & a \end{pmatrix}$
2. a one line notation, which is just the second line of the Cauchy notation: $\sigma = (\ b,\ d,\ c,\ f,\ g,\ a\ )$
3. a cycle notation: $\sigma = (abdfg)(c)$

The cycles in the cycle notation correspond to the number of directed graphs needed to convey the same information. Based on the example above, the first directed graph (V, E) is defined by $V = \{a, b, d, f, g\}$ and $E = \{(a, b), (b, d)(d, f), (f, g), (g, a)\}$, and the second by $V = \{c\}$ and $E = \{(c, c)\}$.

1.
$$\begin{tikzcd} a \arrow[r] & b \arrow[r] & d \arrow[r] & f \arrow[r] & g \arrow[llll, bend right] \end{tikzcd}$$

2. 

At this point, the matrix notation shall be mentioned for completeness only.

2.4 Distance Metrics

There are always $n!$ permutations for a set A with $|A| = n$. Those permutations are per se all equally interesting, but in our case we can define one permutation as the basis and see how far away the other permutations are. This is interesting as far as distance gives us some indication how far away we are from the original order, or the first seriation. Every distance metric will indicate different distances from the original permutation and thereby create different patterns in the visualization of the permutation space. A metric M needs to fulfill three properties:

1. $M(\sigma, \sigma) = 0$ (the distance of an object to itself is always 0),
2. $M(\sigma, \tau) \geq 0$ (the distance to some other objects needs to be 0 or greater),
3. the triangle inequality: "for two permutations σ and τ, the metric M should satisfy $M(\sigma, \tau) \leq M(\sigma) + M(\tau)$" ([KV10, 572]. The straight line between two points is always smaller or equal to the distance of going through a third point).

The order created by the metric should be invariant towards multiplication by some positive factor and towards relabeling. More important in our case when dealing with images are e.g. the density values and they should be taken into account for the distance and penalize the distance if the density values of two pixels are far apart, switching a black pixel, with its density value 0, and a white pixel representing 255 changes the image much more than two pixels with values 125 and 130. [KV10, 571] calls this property richness. The two basic distances considered are Kendall's Tau and Spearman's Footrule, which will build the basis. Kendall's tau is the number of mutual inversions between two permutations σ and τ and is defined as:

Definition 2.4.1 (Set of Inversion Pairs)**.** $\mathbf{IP} = \{(x, y) : \tau(x) > \tau(y), \sigma(x) < \sigma(y)\}$

Definition 2.4.2 (Kendall's Tau)**.** $d_K(\sigma, \tau) = |\mathbf{IP}|$

Spearman's footrule is the absolute distance based on the indices and defined as:

Definition 2.4.3. $d_S(\sigma, \tau) = \sum_{x \in [n]} |\sigma(x) - \tau(x)|$

These two metrics help us estimate how far a certain permutation *locally* is from the original seriation. But for an image, if the density value is similar or equal to the density value of the permuted pixel, then the distance should tend to 0. What we need is a measure that takes the density value into account, more broadly speaking, a data-oriented distance measure.

Introducing weighted variants of Kendall's Tau and Spearman's Footrule alleviates the posed problem and both can be defined as follows, where the weight is the element similarity:

Definition 2.4.4 (Element Similarity)**.** $\forall x, y \in \mathbb{R}$: $d_{sim}(x, y) = |x - y|$

Element similarity is a metric because it satisfies the triangular equation. Every number $x \in \mathbb{R}$ lies somewhere on the real number line. If we take three random real numbers x, y, z and calculate $d_{sim}(x, z)$, $d_{sim}(x, y)$, and $d_{sim}(y, z)$, then $|x - z|$ is the shortest possible distance between x and z. We have two cases to consider for y. First, suppose y lies between x and z, then $|x - y| + |y - z|$ is equal to $|x - z|$, thereby not violating the triangular equation. Secondly, suppose y is outside the interval $[x, z]$. In this case $|x - y| > |x - z|$ or $|y - z| > |x - z|$ holds, fulfilling the triangular equation as the second term cannot be smaller than zero, because the distance is defined via the absolute distances. Therefore, element similarity fulfills the triangular equation and can be regarded as a metric. We are only interested in the weighted version for element similarity, neglecting position weights.

Definition 2.4.5 (Weighted Kendall's Tau)**.** $d_{wK}(\sigma, \tau) = |\mathbf{IP}| + \sum_{(x,y)\in\mathbf{IP}} d_{sim}(\tau(x), \tau(y))$

Definition 2.4.6 (Weighted Spearman's Footrule)**.** $d_{wS}(\sigma, \tau) = \sum_{x\in[n]}(|\sigma(x) - \tau(x)| * d_{sim}(\sigma(x), \tau(x)))$

2.5 Lehmer Code – Factoradic[1]

The Lehmer Code, a representation for permutations, is the basis for explicitly picking certain permutations from the permutation space. This enables us to randomly pick permutations from the permutation space, if the permutation space gets too big for calculating and showing it fully. It is a way to deal with the fast growing time and space complexity when having to do with permutations.

The factoradic system is short for the factorial number system similar to other number systems such as the binary system or the decimal system. They are all representation systems for numbers. We can calculate the factorial number from a decimal number by a series of divisions. Here we will see a short example by calculating the factorial number

[1]cf. [Lai88]. The French publication is historically the first to make the connection.

for 22_{10}:

$$\begin{aligned} 22 &\mod 1 = 0 && (2.1)\\ 22 &\mod 2 = 0 && (2.2)\\ 11 &\mod 3 = 2 && (2.3)\\ 3 &\mod 4 = 3 && (2.4)\\ & && (2.5) \end{aligned}$$

Now we can read the factorial equivalent of 22_{10}, which is $3200_!$. The corresponding decimal number of $3200_!$ can be calculated as follows: $3 * 3! + 2 * 2! + 0 * 1! + 0 * 0! = 22$.

The Lehmer Code representation of a permutation shows us the number of permutations for each place compared to the original order of the list. The Lehmer Code is, therefore, exactly as long as the original list. Let us use a list of four objects (a,b,c,d) and a certain Lehmer code for that list like (1110). This code represents the following permutation: (b, c, d, a). The number at each position indicates the necessary inversion from the original list (a,b,c,d), that is: one takes the index position 1 from (a,b,c,d) resulting (b), then again the index position[2] 1 from (a,c,d) resulting in (b, c), ... until we have (b, c, d, a).

For any list of n objects, every permutation of that list has a rank r, which is defined as $r \in [1, n!] \wedge r \in \mathbb{N}$. By subtracting 1 from the rank r, we arrive at a natural number p. p is the "signe figuratif, considéré comme représentant un nombre écrit dans le système factoriel" ([Lai88, 179]). By calculating the factorial representation of p we get the first digits of the Lehmer Code. If the factoradic has as many digits as the list is long, then this is the Lehmer Code. If the factoradic representation of p is shorter than the list, then it reperesents the rightmost digits of the Lehmer Code. By filling the remaining positions in the Lehmer Code with zeros, we arrive at the final Lehmer Code and will be able to directly get the permutation with rank r. We use our example above and try to find the permutation with rank 10:

$$\begin{aligned} p &= r - 1 && (2.6)\\ 9 &= 10 - 1 && (2.7)\\ 9_{10} &= 1110_! && (2.8)\\ & && (2.9) \end{aligned}$$

As a result, the permutation with rank 10 is (b, c, d, a).[3]

2.6 Application of the Theoretical Basis

The metrics are used in the tool for ordering the permutation space and showing how far one seriation is away from another one. The Lehmer Code is needed for the cases,

[2]As customary in informatics, the index starts at 0.

[3]The concrete algorithms shall be outlined in Section 3.4.

where we just want to pick a sample of the permutation space. This happens as soon as the input size $n \geq 7$. Then the permutation space already has 7! members, i.e. 5040 permutations.

CHAPTER 3

Visualisation of the Permutation Space

3.1 The Goal

The goal was to provide a tool to visualise the permutation space of a real 2-D or 3-D image. This would have shown all possible seriations and not only the typical ones like histograms. By showing the permutation space, a typical user should have been able to identify patterns, if there are any to be found. This kind of analysis is first and foremost exploratory and needs a tool to facilitate the EDA (Exploratory Data Analysis).[1]

The space-constraint of the tool's design is twofold. First, the available space is a computer screen and the pixels one can represent and the human eye. In order to clearly identify patterns, one needs to see every pixel of the given image. For an image with $256x256$ the length of one permutation is 2^{16}, which is already too large for presenting the data in a way to see every pixel on one line. Secondly, we have to look at the space in terms of memory space. For a greyscale image every pixel needs 1 byte and therefore a whole seriation has 2^{16} bytes = 2^6 Kb. For the whole permutation space that would amount to $2^6 \times (2^{16})!$ Kb, thereby going beyond any memory capacity built into computers that are not supercomputers. To be clear, we are permuting based on the indices, but the data sent over the network or kept in memory for showing the permutation space with the pixel values, needs the pixel values and the basis for calculating the size needs to be the memory size of a pixel.

As for time, tests have shown, as expected, a steep upwards time curve directly proportional to the input size. The only viable solution to demonstrate the tool and to use it for exploring the permutation space is to reduce the input size drastically. The input is

[1]"Loosely speaking, any method of looking at data that does not include formal statistical modelling and inference falls under the term exploratory data analysis" ([Sel18, 61])

greyscale images with a size of $3x2$. The number of permutations then is $6! = 720$, with 6 grey values per line and 720 lines. This is also the visualisation used. Each line represents a permutation and has in its present form 6 rectangular boxes filled accordingly with grey values. This basic variant had to be chosen, so as not to prejudice the user.

3.2 Patterns

In order to see patterns we need to know what patterns are, what we are looking for, and how we can spot them. Mathematically speaking,

> "Let S_n denote the set of permutations $\pi = a_1 a_2 ... a_n$ of $[n] = 1, 2, ..., n$, and let $S = \bigcup_{n \geq 0} S_n$. We denote by $\pi(k)$ the entry a_k. And if $\pi \in S_n$ then we say that π has length n. Two sequences of distinct integers, $a_1 a_2 ... a_n$ and $b_1 b_2 ... b_n$, are said to be order isomorphic whenever they satisfy $a_i < a_j$ if and only if $b_i < b_j$ for all $1 \leq i < j \leq n$. We say that $\sigma \in S_n$ contains a copy of $\pi \in S_k$ as a pattern if there is a subsequence of σ order isomporphic to π. For example, $\sigma = 436152$ contains the pattern $\pi = 132$ because of the copy 365"[2] ([DDJ+12, 2760])

This definition is based on order-isomorphic subsequences that can have different grey values, where we would not visually perceive a pattern, because the stimulus does not provide it. For us, patterns need to be defined as based on our perception. "Nolan's definition . . . lists two necessary and sufficient conditions for such a specification: 'p is a pattern if, and only if: (1) p has an organizing principle, and (2) the organizing principle entails repetition.' "([TT14, 296])[3]

The user shall be enabled to find organising principles and repetitions in the data once the space is visualised. Patterns may not emerge when the user only sees some random generation of the permutation space. To support the user, there is the possibility of selecting a certain permutation and choosing a metric for ordering all other permutations ascendingly according to their distance from the chosen permutation.

Every user may see patterns that can be either objective or subjective. It is necessary to differentiate between two kinds of patterns:

1. the ones that are the external stimulus, whether visual or auditive, itself and thereby can be verified by others as well, and

2. those that are "some other percept residing in the mind of the perceiver" ([TT14, 296]), and, hence, not measurable and subjective.

[2]The pattern π consists of three numbers that have the same order as the numbers $3, 5, 6$ in the permutation σ. Therefore σ contains π.

[3]We have to cite Nolan according to [TT14], because the given website reference is no longer available.

What we are looking for here is the objective kind of patterns. Even there, it is the case that one needs to distinguish between domain experts and those, who have just a bit or no knowledge in the domain. Experts tend to ask specific questions, that lead to more relevant observations in EDA and the patterns they see can be shown to be statistically more relevant.[4]

Our tool is, therefore, designed to be used by experts helping them to find patterns if there are any to be found. For now we do not know if patterns can be discovered, because no one has tried to visualise the permutation space of an image, the way it is done here. Histograms and spectra are widely accepted graphical EDA methods and are in our case serialisations of image data. The visualisation provides a way to identify all the histograms and spectra in the permutation space and when choosing a permutation with a distance metric it even allows the user to see the distance from the chosen permutation.

3.3 A short introduction of the tool

The users does not need to be logged in, to use the application. Upon entering the correct url the user is presented with a toolbar header. The url must be determined by the user when starting the docker container for the frontend. In its right corner one can choose between single and double image analysis and it is possible to display the legend for the color codes for histogram and spectrum permutations as well as the distance indicator.

Single image analysis gives the user the possibility to load one image and have the unordered and if desired ordered permutation space. If she chooses double image analysis, she will be able to upload a second image and thereby can compare at most four permutation spaces two for each image.

Directly beneath, a file chooser allows to get an image from the file system, which will be displayed in a thumbnail below. Last, but not least, there is a button to load all permutations unordered at first (Figure 3.1).

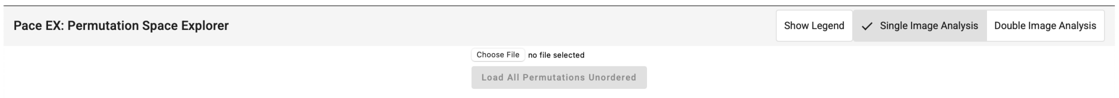
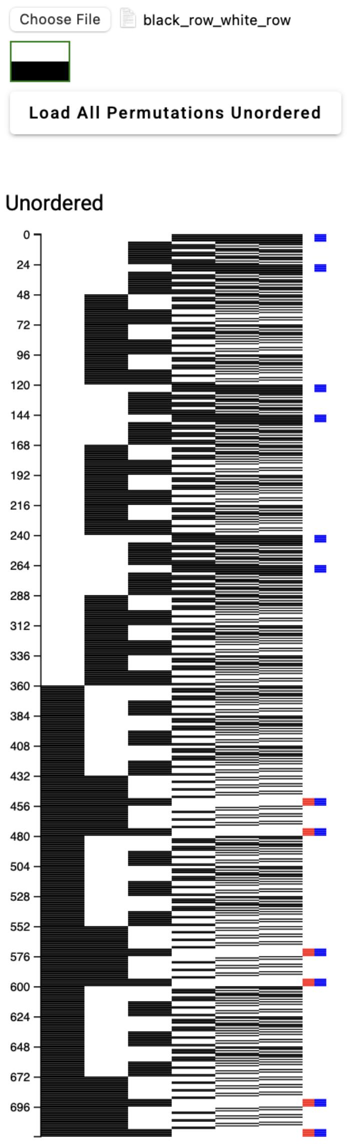


Figure 3.1: The application start

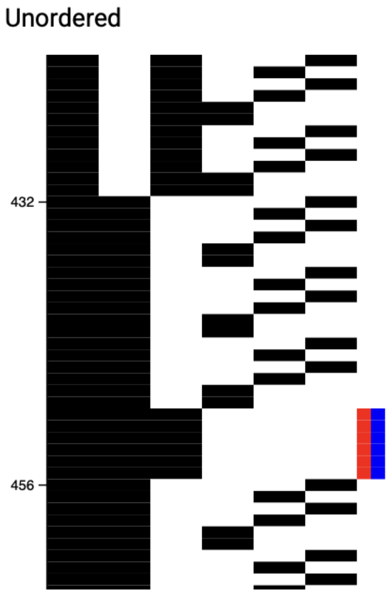
The visualisation of the unordered permutation space shows all permutations starting from the pixel list constructed by putting one row after the other. The permutations

[4]See [SHOP19, 1509] and [SHOP19, 1516].

have the order in which they are produced by the algorithm (see Chapter 3.4). To the right special permutations are indicated. Histograms are marked red and spectra blue. Depending on the image it is also possible that red and blue indications are there (see Figure 3.2a and Figure 3.2b). A spectrum and a histogram permutation may be identical, e.g. when an image with a black row and white row is permuted, then all permutations with the black pixels at the beginning are histogram permutations and spectra as well. The user may zoom in to get a better picture of the surroundings of a single permutation.



(a) The application after having chosen an image and having pressed the load button



(b) An unordered permutation space visualisation close up

Figure 3.2: Application Intro

By clicking on a permutation the user chooses it and opens a dialog for selecting a distance metric, for ordering the permutations space (see Figure 3.3a). The visualisation is shown at the right side of the unordered permutation and has to its right an extra bar with a gradient in green indicating the distance from the chosen permutation (see

Figure 3.3b).

In the ordered space the permutations are grouped by their distance to the chosen permutation. The percentage and absolute value of the distance are given at the end of each distance group. From there, it is easily visible how many of them are in one distance group compared to another distance group (see Figure 3.4a). The user can zoom in here as well (see Figure 3.4b).

In order to be able to compare two images and their respective permutation spaces, the user can choose 'Double Image Analysis', which gives a good overview and first impression (see Figure 3.4c). For a more detailed analysis, different images and metrics should be compared. With the double image analysis and the zoom functionality she is able to compare special regions or to even compare the same image with two different metrics or two different images with the same distance metric.

3.4 The Setup of the Software

3.4.1 Architecture

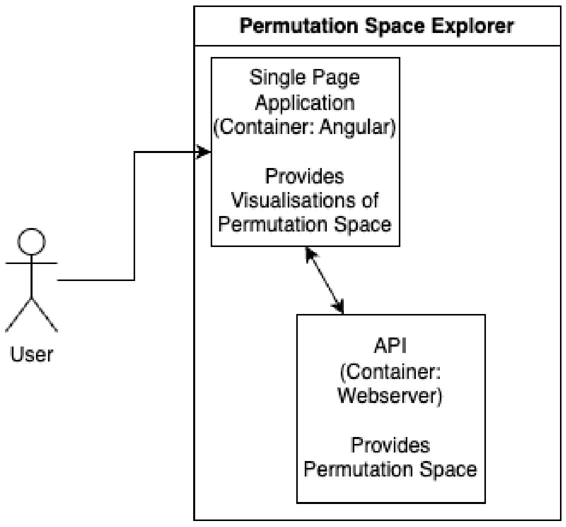
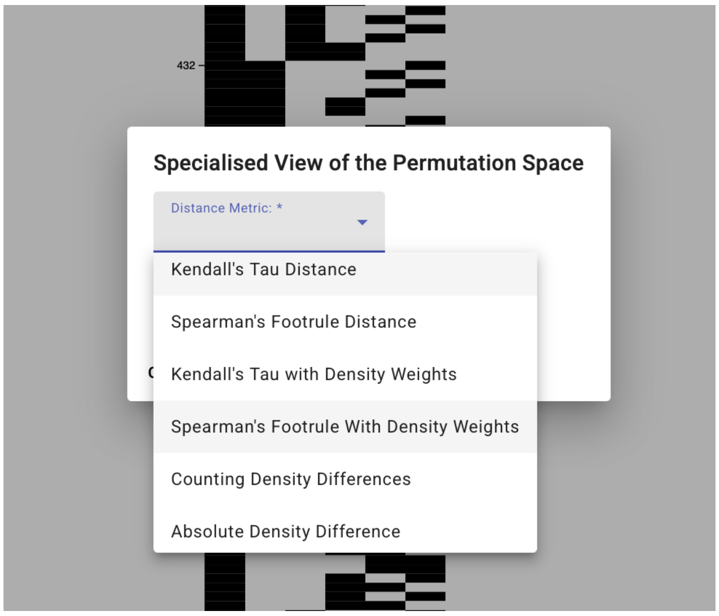

Permutation Space Explorer
Single Page Application (Container: Angular)
Provides Visualisations of Permutation Space
API (Container: Webserver)
Provides Permutation Space
User

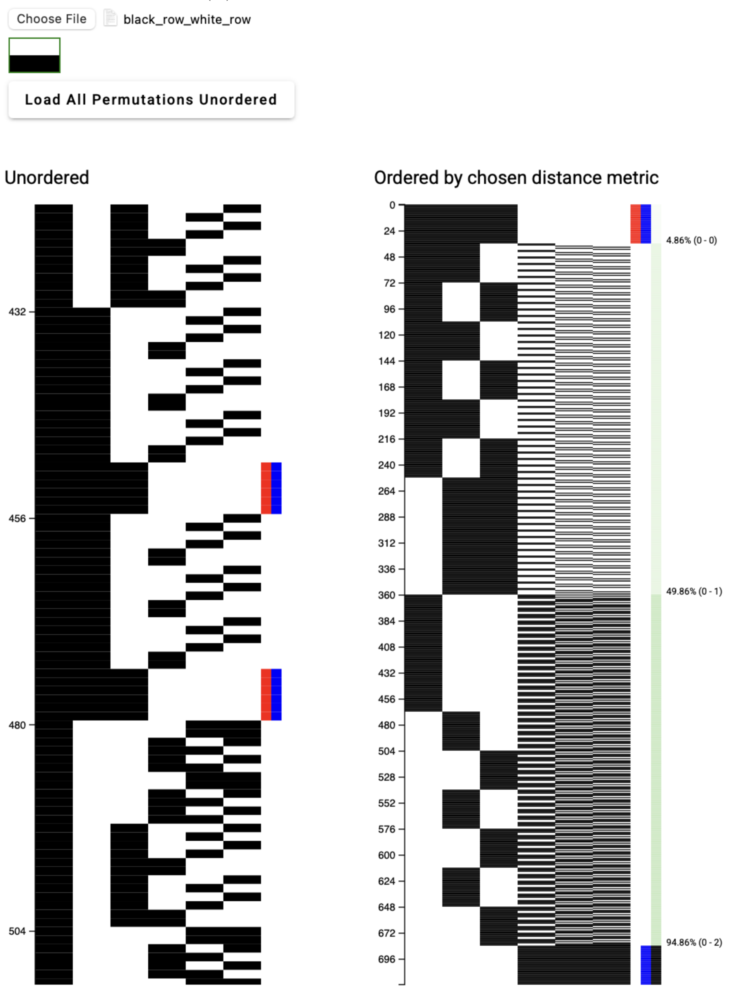
Figure 3.5: Software Architecture

The software is built from two parts. This setup distributes the work load between the visualisation part and the permutation space calculation. The user communicates via a single page application with an API server, which provides a list of all permutations for an image (see Figure 3.5).

The single page application is written in typescript using the Angular Framework 19.0.0. The API server is written in Python using FastApi. The code is hosted in BitBucket. D3.js

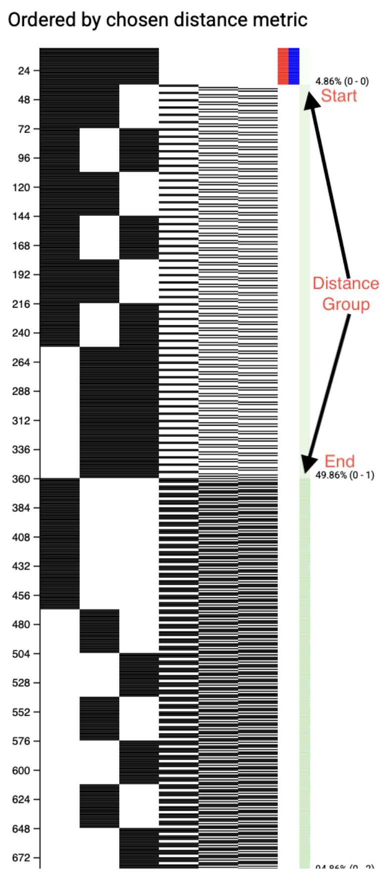

432
Specialised View of the Permutation Space
Distance Metric: *
Kendall's Tau Distance
Spearman's Footrule Distance
Kendall's Tau with Density Weights
Spearman's Footrule With Density Weights
Counting Density Differences
Absolute Density Difference

(a) Selecting a metric

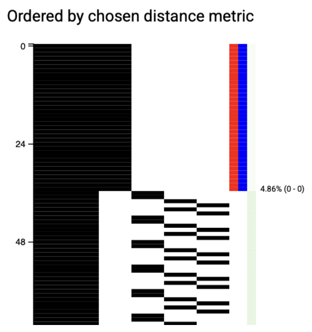


(b) Unordered and ordered space

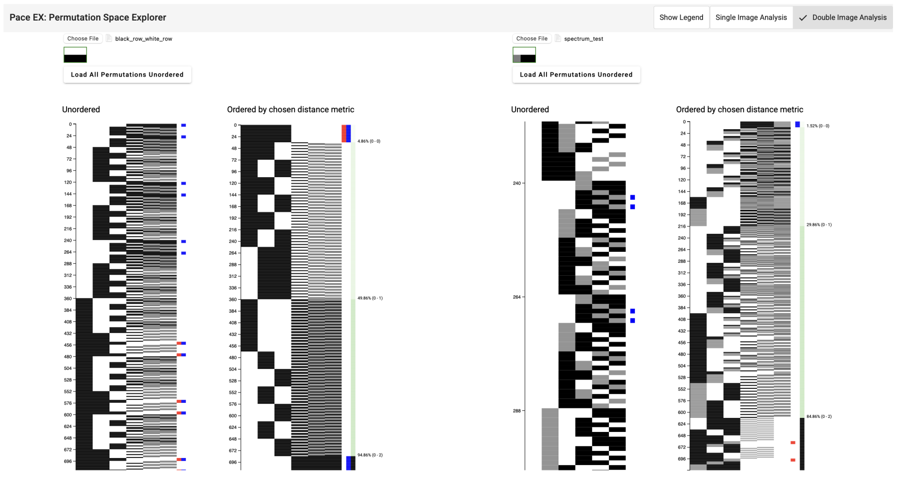
Figure 3.3: Application continued



(a) An ordered permutation space



(b) An ordered permutation space close up



(c) A full double image analysis

Figure 3.4: Application end

is used as a visualisation framework. Both parts can be downloaded from DockerHub[5] and started as docker containers.

3.4.2 Algorithms

The mathematical basis for the following algorithms has been discussed in Section 2. For the formulation of the algorithms we use predefined functions used in Python code, but available in other programming and scripting languages as well. The names should be self explanatory except for *divmod*, which is explained in the algorithm via a comment.

3.4.2.1 Python Itertools permutations (list, cycles)

The algorithm used for generating the unordered permutation space is from the Itertools library in Python. The documentation for the algorithm can be found at [Pyt]. It uses the indices of the given list, and not the values. The cycles parameter is optional and is not used in the tool, because the permutations generated shall be generated for the whole list. By setting the cycles parameter to an integer $l <$ size of list, one could do partial permutations.

[5]Here are the links: https://hub.docker.com/r/petesouchos/app-permutation-space-backend/tags and https://hub.docker.com/r/petesouchos/app-permutation-space-frontend/tags.

3.4.2.2 Lehmer Code via the Factorial Number System

Algorithm 3.1: Algorithm to calculate a certain permutation given a position number $p <$ len(given list)! using the Lehmer Code

```
input  : originalListLength $\in \mathbb{N}$
input  : positionNumber $\in \mathbb{N}$
output : result: the permutation corresponding to the positionNumber

sizeOriginalList $\leftarrow$ originalListLength $+ 1$;
tmp $\leftarrow$ positionNumber;
originalListIndices $\leftarrow$ range(0, sizeOriginalList $- 1$);
factoradic $\leftarrow$ empty list;
result $\leftarrow$ empty array;
$i \leftarrow 1$;
// The factoradic, the number in the factorial number
    system, is calculated based on modulo division, where
    the result is a tuple of the integer result and the
    remainder.
while tmp $< 0$ do
    (intResult, remainder) $\leftarrow$ divmod(tmp, $i$);
    factoradic.append(remainder);
    tmp $\leftarrow$ intResult;
    $i \leftarrow i + 1$;
end
// In case the factoradic is too short we fill it with 0s
while len(factoradic) $<$ originalListLength do
    factoradic.append(0);
end
// The pop() guarantuees that the last value in the
    factoradic list is used first.  The value of the
    original list is taken from the original list and put
    into the result list.
while size(originalListIndices) $> 0$ do
    index $\leftarrow$ factoradic.pop();
    result.append(originalListIndices[index]);
    originalListIndices $\leftarrow$ delete(originalListIndices, index);
end
// The result list contains the correct order of values
    for the permutation with the given position number (i.e.
    the rank)
return result;
```

3.4.2.3 Distance Metrics

Algorithm 3.2: Kendall's Tau: calculateDistanceKendallTau

```
input : originalList l of density values $v \in \mathbb{N} \wedge v \in [0, 255]$
input : permutation: a list of lists with each two entries, one for the original
        index position and one for density value $v \in \mathbb{N} \wedge v \in [0, 255]$
output: The distance $d \in \mathbb{R}^+$

count ← 0;
permutationOrder ← list of all first entries of the sublists of permutation, i.e. the
  original index positions;
permutationOrderOriginal ← list of all first entries of the sublists of originalList, i.e.
  the original index positions;
pairs ← itertools.combinations(range(0, len(permutationOrder)), 2);
for $x, y \in$ pairs do
    $a$ ← permutationOrderOriginal[$x$] − permutationOrderOriginal[$y$];
    $b$ ← permutationOrder[$x$] − permutationOrder[$y$];
    if $a! = b$ then
        count = count + 1;
    end
end
return count;
```

Algorithm 3.3: Kendall's Tau with density weights: calculateKTWithVolumeWeights

```
input  : originalList l of density values $v \in \mathbb{N} \wedge v \in [0, 255]$
input  : permutation: a list of lists with each two entries, one for the original
         index position and one for density value $v \in \mathbb{N} \wedge v \in [0, 255]$
output : The distance $d \in \mathbb{R}^+$

1  count $\leftarrow 0$;
2  permutationOrder $\leftarrow$ list of all first entries of the sublists of permutation, i.e. the
     original index positions;
3  permutationOrderOriginal $\leftarrow$ list of all first entries of the sublists of originalList, i.e.
     the original index positions;
4  densityValues $\leftarrow$ list of all second etnries of the sublists of permutation, i.e. the
     density values;
5  densityValuesOriginal $\leftarrow$ list of all second etnries of the sublists of originalList, i.e.
     the density values;
6  pairs $\leftarrow$ itertools.combinations(range(0, len(permutationOrder)), 2);
7  for $x, y \in$ pairs do
8      $a \leftarrow$ permutationOrderOriginal[$x$] $-$ permutationOrderOriginal[$y$];
9      $b \leftarrow$ permutationOrder[$x$] $-$ permutationOrder[$y$];
10     if $a! = b$ then
11         count = count + abs(densityValues[$x$] $-$ densityValuesOriginal[$x$]);
12     end
13 end
14 return count;
```

Algorithm 3.4: Spearman's Footrule: calculateSF

```
input  : originalList l of density values $v \in \mathbb{N} \wedge v \in [0, 255]$
input  : permutation: a list of lists with each two entries, one for the original
         index position and one for density value $v \in \mathbb{N} \wedge v \in [0, 255]$
output : The distance $d \in \mathbb{R}^+$
1 result ← [abs($i_1[0] - i_2[0]$) for $i_1, i_2 \in$ zip(originalList, permutation)];
2 // zip(list1, list2) forms a new list of tuples, where the
     first tuple has the first elements of both list, the
     second tuple the second elements of each list, and so
     forth.
3 return sum (result);
```

Algorithm 3.5: Spearman's Footrule with density weights: calculateSFWithDensityWeights

```
input  : originalList l of density values $v \in \mathbb{N} \wedge v \in [0, 255]$
input  : permutation: a list of lists with each two entries, one for the original
         index position and one for density value $v \in \mathbb{N} \wedge v \in [0, 255]$
output : The distance $d \in \mathbb{R}^+$

result ← 0;
permutationOrder ← list of all first entries of the sublists of permutation, i.e. the
  original index positions;
densityValues ← list of all second etnries of the sublists of permutation, i.e. the
  density values;
lengthOfValues ← len(permutationOrder);
for $i$ ← 0..lengthOfValues do
    sumA ← 0;
    sumB ← 0;
    for $j$ ← 0..lengthOfValues do
        if $j \leq i$ then
            sumA ← sumA +abs(densityValues[$i$] − densityValues[$j$]);
        end
        if permutationOrder[$j$] ≤ permutationOrder[$i$] then
            sumB ← sumB +abs(densityValues[$i$] − densityValues[$j$]);
        end
    end
    result ← result + densityValues[$i$] ∗ abs(sumA − sumB);
end
return result;
```

Algorithm 3.6: Hamming Distance: calculateHammingDistance

```
input  : originalList l of density values $v \in \mathbb{N} \land v \in [0, 255]$
input  : permutation: a list of lists with each two entries, one for the original
         index position and one for density value $v \in \mathbb{N} \land v \in [0, 255]$
output : The distance $d \in \mathbb{R}^+$

1 count ← 0;
2 result ← [abs($i_2[1] - i_1[1]$) for $i_1, i_2 \in$ zip(originalList, permutation) if
    $i_1[1] < i_2[1]$];
3 if result is not empty then
4     count ← len (result);
5 end
6 return count;
```

Algorithm 3.7: Distance based on the difference in density values: calculate-DistanceBySubtractingDensityValues

```
input  : originalList l of density values $v \in \mathbb{N} \land v \in [0, 255]$
input  : permutation: a list of lists with each two entries, one for the original
         index position and one for density value $v \in \mathbb{N} \land v \in [0, 255]$
output : The distance $d \in \mathbb{R}^+$

1 count ← 0;
2 result ← [abs($i_2[1] - i_1[1]$) for $i_1, i_2 \in$ zip(originalList, permutation) if
    $i_1[1] < i_2[1]$];
3 if result is not empty then
4     count ← reduce (add, result, 0);
5 end
6 return count;
```

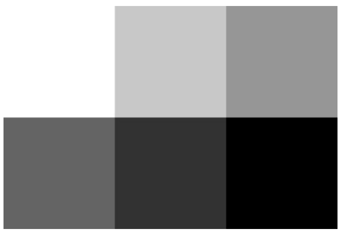
CHAPTER 4

Test Images

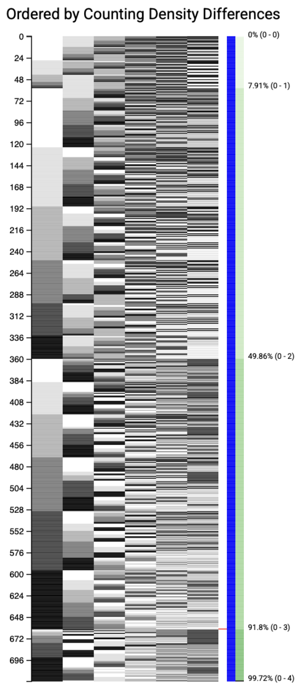
In an ideal world the images would be real world images that can be edited, transformed, filtered, and so on. Here very small images have to suffice, because we would like to see the whole permutation space.

The test images chosen are 3 by 2 in resolution. This is the ideal size for showing all permutations. We chose eleven different test images, where each image is slightly different from the next. The idea is to see differences and patterns in the permutations, when small changes in the image occur. The images highlight certain properties that must hold and, therefore, they check the correctness of the visualisation. For test-1-1-1-1-1-1.jpeg (see Figure 4.1a), e.g., every permutation is a spectrum and the set of Hamming distances has the same cardinality as the set of Kendall's Tau distances. A pattern is visible here, that can be observed in both distance metrics. The pattern consists of bright to dark in every column per distance group (see Figure 4.1b and Figure 4.1c). One also sees clearly that every permtutation is a spectrum (marked blue at the right side), but only one permutation is a histogram (marked red at the right side). In both metrics, the histogram's position is near the end (see Figure 4.2a and Figure 4.2b). The other test images are built the same by enlarging regions by one pixel at a time. We, therefore, have the test images as shown in Table 4.1

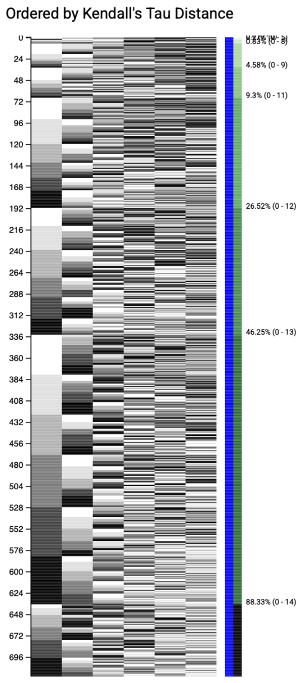
The more interesting test images would be real world images and one could transform them in some regions or all of them, in order to compare the original and the transformed one. As mentioned in Section 3.1 the time and space constraints are easily reached with permutations. Showing the whole permutation space is for a first phase more important than the image size, in order to get a sense of what is shown and what can be seen and expected from this tool.



(a) All pixels differ.



(b) Hamming Distance



(c) Kendall's Tau

Figure 4.1: An exemplary test image

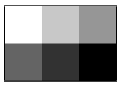
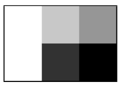
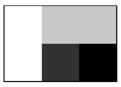
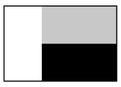
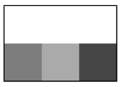
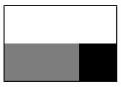
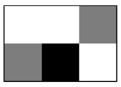
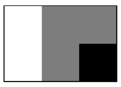
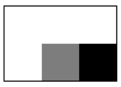
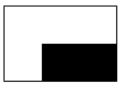
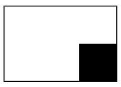
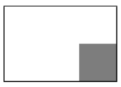
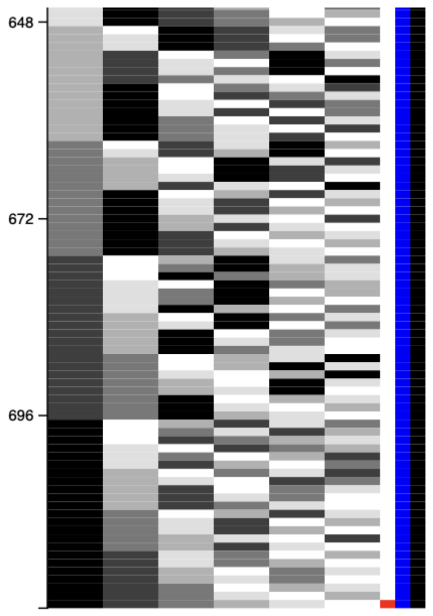
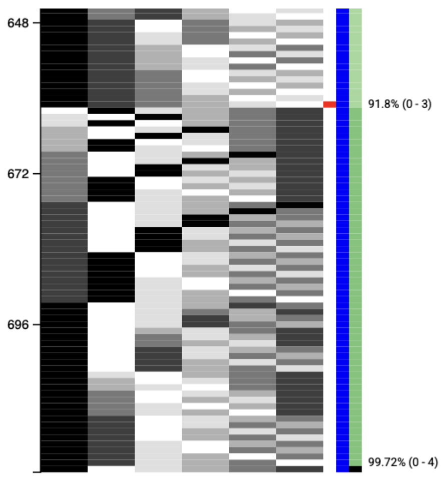
Image Name	Image
test-1-1-1-1-1-1.jpeg	
test-2-1-1-1-1.jpeg	
test-2-2-1-1.jpeg	
test-2-2-2.jpeg	
test-3-1-1-1.jpeg	
test-3-2-1.jpeg	
test-3-2-1-mixed.jpeg	
test-2-3-1.jpeg	
test-4-1-1.jpeg	
test-4-2.jpeg	
test-5-1.jpeg	
test-5-1-middle-grey.jpeg	
test-6.jpeg	

Table 4.1: Test Images



(a) Histogram position in Kendall's Tau



(b) Histogram position in Hamming Distance

Figure 4.2: Histogram positions in Test Images

4.1 Pattern Discovery

We have already discussed what patterns there can be, and we used one of the definitions for our purpose. To summarise, we need an organising principle and repetition which can be perceived visually from the permutation space or the comparison of the produced visualisations. The question "Can we see patterns?" translates to, whether we can see at least one organising principle in some region of the visualisation and whether this principle includes repetition. The tool gives an unordered view of the permutation space and patterns only emerge when using images with similar pixel order (see Figure 4.3). If we change the pixel order more significantly, patterns are not anymore discernible (see Figure 4.4). Even in the ordered view, it depends on the chosen metric for the single image analysis, and also the chosen permutation when comparing two images.

For the metrics used here, only some of them are relevant for seeing patterns:

- Hamming Distance, aliased here with Counting Density Differences
- Absolute Density Difference
- Kendall's Tau with Density Weights
- Spearman's Footrule with Density Weights

Metrics, where we could not see any patterns:

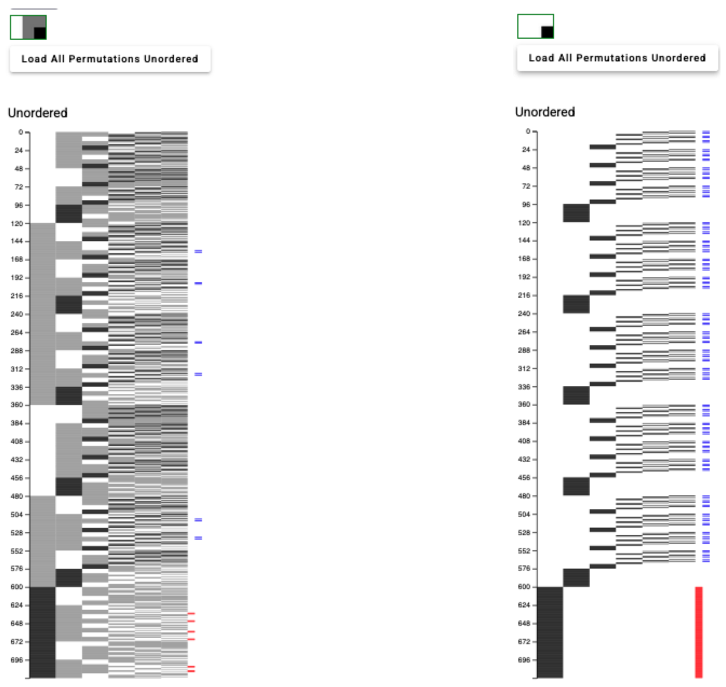


Figure 4.3: Different pixels but same order

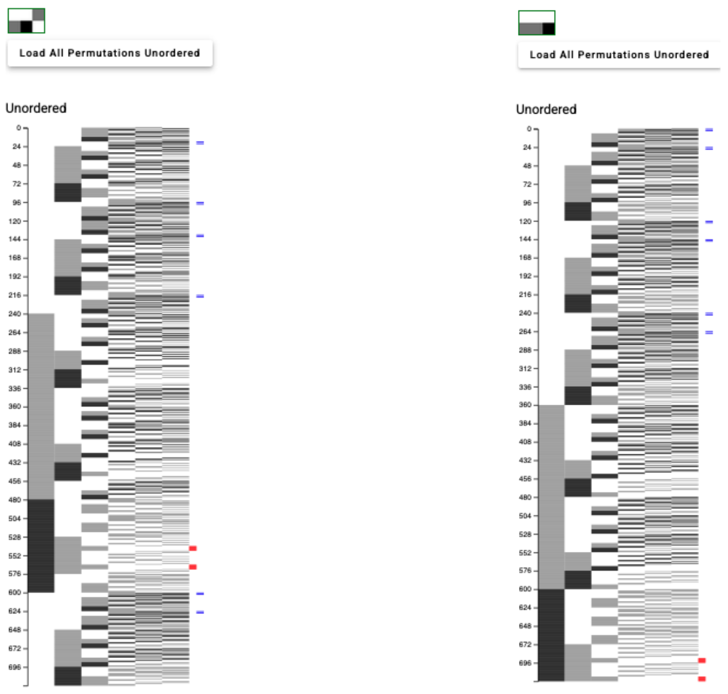

Load All Permutations Unordered
Unordered
Load All Permutations Unordered
Unordered

Figure 4.4: Different pixel order

- Kendall's Tau
- Spearman's Footrule

The difference between the two sets of metrics stems from the fact that the second set does not take the pixel values into account at all. The Hamming distance uses them at least to compare for equality.

For images with no duplicate pixel values even using the ordered view does not give us more insight. Using Kendall's Tau and the Hamming distance to compare the permutation spaces for the test-1-1-1-1-1-1.jpeg image shows in no case any organising principle. As soon as we move on to the test-2-1-1-1-1.jpeg, moving same value pixel groups can be observed, but only for distance metrics from the second group. The difference is simply the provided group of pixels with the same grey value. Even if the region is not there in the image, while preserving the pixel values but in different order, the group movement can be observed and brings an organising principle, with the group repetition. This observation is a gradual one. With larger regions or multiple pixels with the same value patterns get more visible.

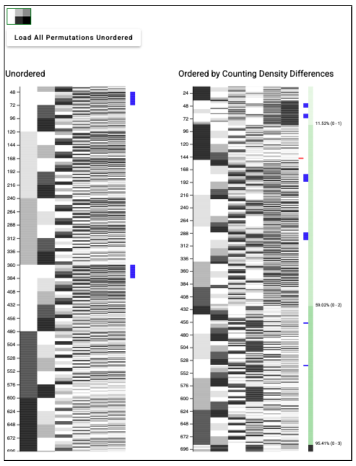
Using histogram- or spectrum-permutations helps in identifying patterns as chosen permutations helps. As a rule, unordered starting points generate more random spaces. In order to see this and the development more clearly, the Tables 4.2 and 4.3 need to be looked at.

The organising principle for every column in both metrics is the change from one end of the grey scale spectrum to the other. The positions of the spectra and histograms change as well, but that inherently depends on the chosen permutation, the metric and the image structure. Given a certain combination of these three factors it is easy to explain the given positions. For example, let us choose the 4-2 example with the black pixels at the start and end of the row as chosen permutation (Figure 4.5)). It is clear that in the Hamming distance the histogram and spectrum permutations only have distance 1 and not 2. Therefore both of them are in the first half. On the other hand, if we choose a spectrum permutation to begin with (Figure 4.6)), then the histogram permutation will be in the second half towards the end. If the image had four black pixels and two white ones, the histogram and spectrum permutations would coincide and both be at the same positions (see Figure 4.7a and Figure 4.7b).

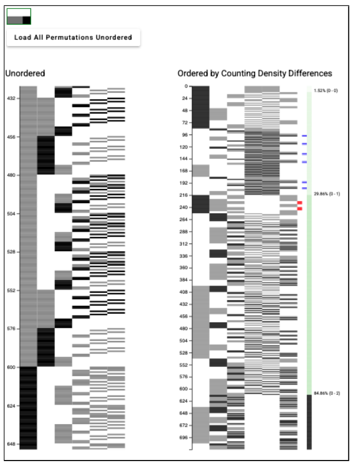
The results depend on the choices made by the user. In general, one can say that the unordered view has its own structure due to the algorithm used by the application to generate the permutation space. It is, therefore, predictable. Only the ordered view gives more insights and makes it possible to see patterns. The patterns themselves are not always easily discernible and develop gradually from image to image. We could identify a pattern in the ordered view when using a metric from the first set, where all metrics take the density values into account. The case of the noisy image, where each pixel has a different value, shows that in such cases no patterns are to be observed.

1. Any non special permutation + Hamming:

   a) 2-1-1-1-1:


Load All Permutations Unordered
Unordered
Ordered by Counting Density Differences

   b) 3-2-1:


Load All Permutations Unordered
Unordered
Ordered by Counting Density Differences

   c) 4-2:

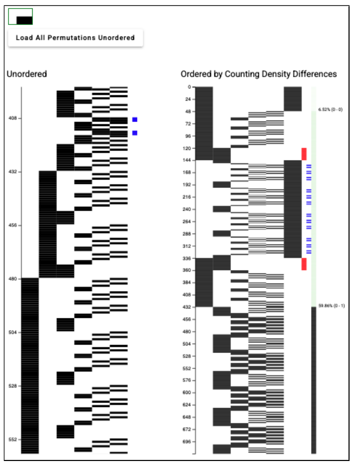
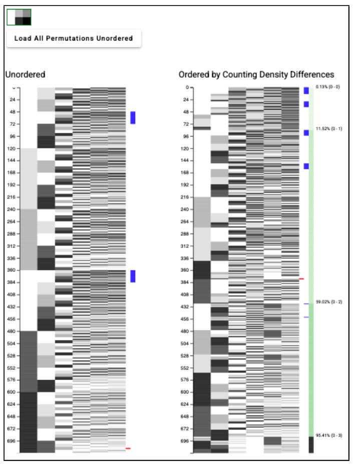

Load All Permutations Unordered
Unordered
Ordered by Counting Density Differences

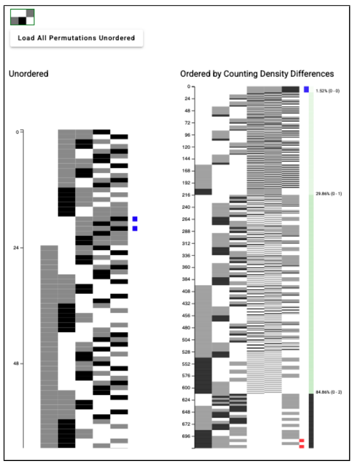
Figure 4.5: No special Permutation and Hamming

2. Spectrum + Hamming:

   a) 2-1-1-1-1:


Load All Permutations Unordered
Unordered
Ordered by Counting Density Differences

   b) 3-2-1:


Load All Permutations Unordered
Unordered
Ordered by Counting Density Differences

   c) 4-2:

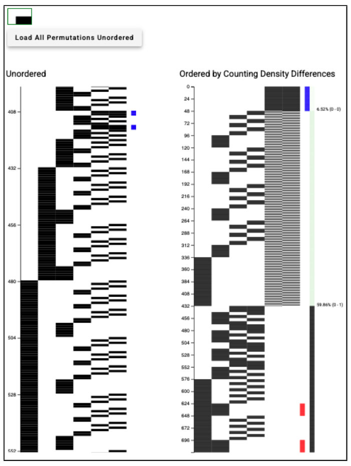
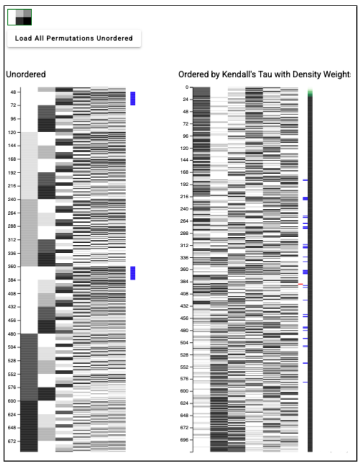

Load All Permutations Unordered
Unordered
Ordered by Counting Density Differences

Figure 4.6: Spectrum and Hamming Distance

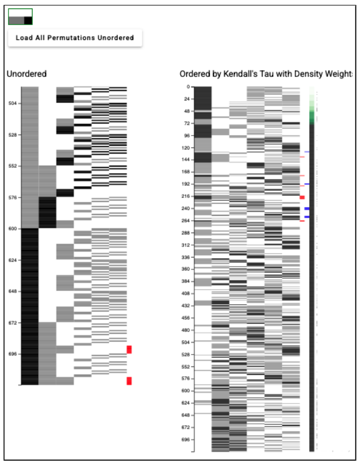
Table 4.2: Ordered and Unordered Permutation Spaces for Comparison Part I

3. Any non special permutation + Kendall's Tau with Density Weights:

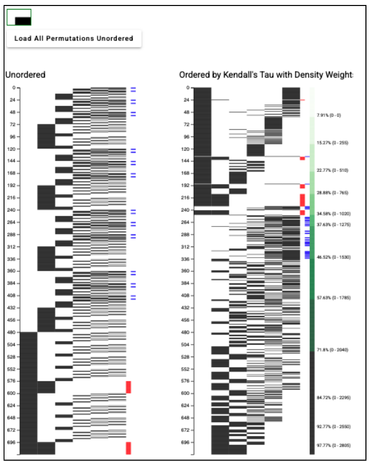
   a) 2-1-1-1-1:


Load All Permutations Unordered
Unordered
Ordered by Kendall's Tau with Density Weight

   b) 3-2-1:

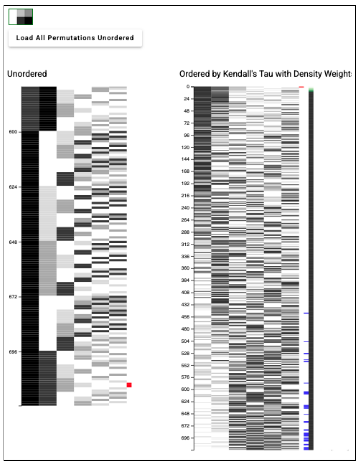

Load All Permutations Unordered
Unordered
Ordered by Kendall's Tau with Density Weight

   c) 4-2:

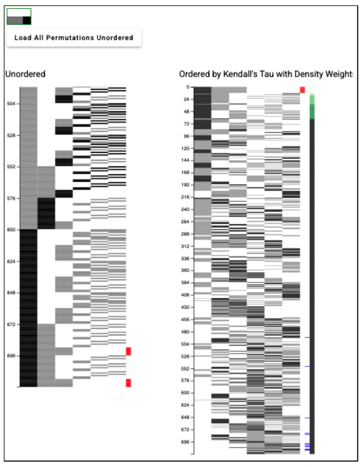

Load All Permutations Unordered
Unordered
Ordered by Kendall's Tau with Density Weight

4. Histogram + Kendall's Tau with Density Weights:

   a) 2-1-1-1-1:


Load All Permutations Unordered
Unordered
Ordered by Kendall's Tau with Density Weight

   b) 3-2-1:


Load All Permutations Unordered
Unordered
Ordered by Kendall's Tau with Density Weight

   c) 4-2:

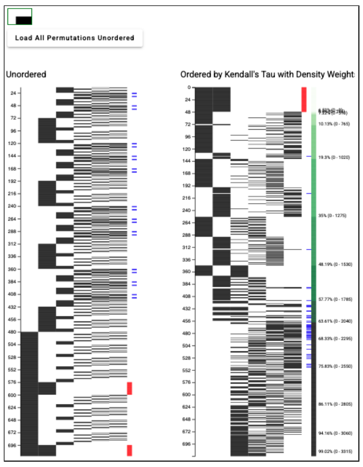
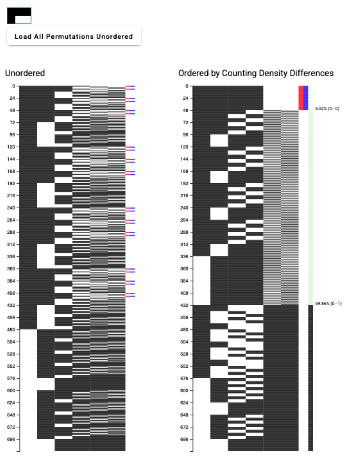
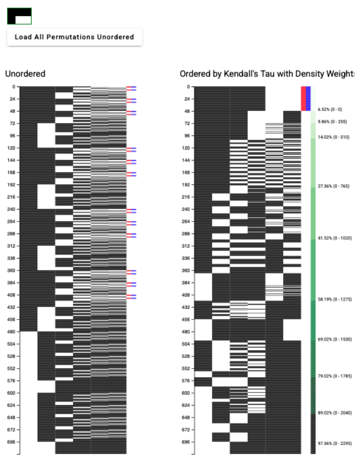

Load All Permutations Unordered
Unordered
Ordered by Kendall's Tau with Density Weight

Table 4.3: Ordered and Unordered Permutation Spaces for Comparison Part II


Load All Permutations Unordered
Unordered
Ordered by Counting Density Differences
6.52% (0 - 0)
59.86% (0 - 1)

(a) Hamming Distance



(b) Kendall's Tau with Density Weights

Figure 4.7: Comparison of two metrics with 4 black and 2 white pixels

CHAPTER 5

Conclusions and Future Work

The investigation started with the idea to visualise the whole permutation space for 2-D images and even for 3-D objects. At an early stage and from theoretical considerations it became clear, that this is not viable in the first phase. The whole space cannot be completely displayed for these objects with the selected encodings.

Still, having chosen the parameters based on the images to be analysed, one is able to discern organising principles with repetitions and therefore, to see patterns. The test images are only 3x2, a negligible size compared to real world images. It would be interesting to look at the permutation of larger images with a lot of structure.

In order to achieve this, there are several possibilities: some at the user interaction level and some at the application and more algorithmic level.

1. On the user level:
    a) The user can/has to choose a region of interest as basis for the permutation space
    b) The user inspects the whole permutation space for larger regions of the image and can zoom in to the pixel level
2. On the application level:
    a) We could use the Lehmer Code to show only randomly chosen permutations from the permutation space, or pick certain permutations to get an evenly distributed sample.
    b) We could resize the image, but there one has to live with data loss. Resizing means using filters to merge regions and thereby reducing and changing the original pixel values. For an analysis this poses a serious problem, because the anaylsis is not any more directly based on the original image.

When using larger images one can add the Hilbert Curve as special permutation. The Hilbert case takes into account the locality of the pixels. In this scenario one could use the Euclidean distance as a metric or the Manhattan distance. The options had all been implemented, but discarded in a last review. The first priority has been to show the whole permutation space of an image on one screen without scrolling. The test images are not in the correct shape for constructing the Hilbert Curve (the algorithm needs an $n \times n$ matrix) and distance metrics based on the locality of the pixels would also need larger images to be more expressive and would be especially interesting if special permutations were indicated that are solely based on the locality.

To sum up, the work done here can be considered as a basis, but needs to be developed further with domain experts, including the indicated options in order to handle laer dlrow atad[1].

[1]i.e. real world data

Overview of Generative AI Tools Used

I have not used any AI tools for generating code or text.

List of Figures

List of Tables

List of Algorithms

Bibliography

[Das14] Abhijit Dasgupta. *Set Theory : With an Introduction to Real Point Sets*. Birkhäuser, 2014.

[DDJ$^+$12] Theodore Dokos, Tim Dwyer, Bryan P. Johnson, Bruce E. Sagan, and Kimberly Selsor. Permutation patterns and statistics. *Discrete Mathematics*, 312(18):2760–2775, 2012. https://www.sciencedirect.com/science/article/pii/S0012365X12002361 last accessed 2025-10-05.

[KV10] Ravi Kumar and Sergei Vassilvitskii. Generalized distances between rankings. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, page 571–580, New York, NY, USA, 2010. Association for Computing Machinery. https://doi.org/10.1145/1772690.1772749 last accessed 2025-10-05.

[Lai88] C.-A. Laisant. Sur la numération factorielle, application aux permutations. *Bulletin de la Société Mathématique de France*, 16:176–183, 1888. http://archive.numdam.org/articles/10.24033/bsmf.378/ last accessed 2025-10-05.

[pera] permutare. https://logeion.uchicago.edu/permutare. last accessed 2025-10-05.

[perb] permutatio. https://logeion.uchicago.edu/permutatio. last accessed 2025-10-05.

[Pyt] Python documentation on itertools. https://docs.python.org/3/library/itertools.html#itertools.permutations. last accessed 2025-10-05.

[Qui] Quintilian. https://artflsrv03.uchicago.edu/philologic4/Latin/navigate/126/2/3/25/. last accessed 2025-10-05.

[Qun] Qunitilian english translation. https://www.perseus.tufts.edu/hopper/text?doc=Perseus%3Atext%3A2007.01.0060%3Abook%3D1%3Achapter%3D1%3Asection%3D24. last accessed 2025-10-05.

[Sel18] Howard J. Seltman. *Experimental design and analysis*. 2018. https://www.stat.cmu.edu/~hseltman/309/Book/Book.pdf last accessed 2025-10-05.

[SHOP19] Rafael Savvides, Andreas Henelius, Emilia Oikarinen, and Kai Puolamäki. Significance of patterns in data visualisations. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 1509–1517, New York, NY, USA, 2019. Association for Computing Machinery. https://doi.org/10.1145/3292500.3330994 last accessed 2025-10-05.

[TT14] Eva R. Toussaint and Toussaint Godfried T. What is a pattern? In *Proceedings of Bridges 2014: Mathematics, Music, Art, Architecture, Culture*, 2014. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=7b5766cf3b67f57531c72043e04d872af556ec73 last accessed 2025-10-05.