

# Unsupervised Learning of Invariance Transformations

Aleksandar Vučković<sup>1</sup>, Benedikt Stock<sup>2</sup>, Alexander V. Hopp<sup>1</sup>,  
Mathias Winkel<sup>1</sup>, and Helmut Linde<sup>3,\*</sup>

<sup>1</sup>*Merck KGaA, Darmstadt, Germany*

<sup>2</sup>*University of Oxford, Oxford, United Kingdom*

<sup>3</sup>*Transylvanian Institute of Neuroscience, Cluj-Napoca, Romania*

\*Corresponding Author

July 25, 2023

The need for large amounts of training data in modern machine learning is one of the biggest challenges of the field. Compared to the brain, current artificial algorithms are much less capable of learning invariance transformations and employing them to extrapolate knowledge from small sample sets.

It has recently been proposed that the brain might encode perceptual invariances as approximate graph symmetries in the network of synaptic connections. Such symmetries may arise naturally through a biologically plausible process of unsupervised Hebbian learning.

In the present paper, we illustrate this proposal on numerical examples, showing that invariance transformations can indeed be recovered from the structure of recurrent synaptic connections which form within a layer of feature detector neurons via a simple Hebbian learning rule.

In order to numerically recover the invariance transformations from the resulting recurrent network, we develop a general algorithmic framework for finding approximate graph automorphisms. We discuss how this framework can be used to find approximate automorphisms in weighted graphs in general.

## 1. Introduction

Many cognitive tasks require an agent to establish a model of its surroundings based on sensory inputs. This involves the reconstruction of semantic and functional relationships inherent in the environment from data generated by perceptual processes which are not fully structure-preserving. Based on the assumption that several independent entities contribute to and interact in generating the percepts, it is a promising strategy to decorrelate this data and thus reveal some of the structure present in the environment.

An approach often followed in practice is to consider the observed data points as vectors in a suitable vector space and find a convenient base in which to represent them. Well-known algorithms based on

this idea are, for example, Principal Component Analysis [1–3], Independent Component Analysis [4, 5], Non-negative Matrix Factorization [6–8] and several variants of Sparse Coding [8–10]. The basis vectors found by such algorithms are typically called *atoms* and the collection of atoms is called *dictionary*, a nomenclature we will adhere to in the following.

When applied to natural data, these algorithms typically find atoms of significantly lower complexity than the entities they try to reconstruct. Even though there is, in fact, still a rich structure of correlations present after transforming to a basis of such simple atoms [11, 12], today’s algorithms are not able to fully exploit this implicit information in the construction of more complex entity representations.

The apparent difficulty in reconstructing complex entities from unstructured data stems to a large extent from the combinatorial explosion in the number of possibilities for such an entity to appear. The appealing approach of extending the dictionary to contain more complex atoms is impractical due to limited computing power and memory capacity and due to the much more fundamental problem that suitable atoms of a certain complexity do not appear sufficiently often in the data to be identified.

A promising approach is to model invariances present in the data explicitly. In practice, such invariances may include transformations like rotations or translations. Their explicit inclusion in the model ensures that observations that only differ by a transformation of this kind can still contribute to the formation of the same atoms and to discovering structure in new transformed observations. Examples of such models are Transform-invariant NMF [13], Transform-invariant Restricted Boltzmann Machines [14], or the generalized pooling mechanism presented in [15]. Yet, for such algorithms to work, the invariance transformations must be known explicitly in the first place. While this is the case for basic geometric transformations, it is desirable to have a general way of learning transformations from the data that does not require a-priori knowledge.

Several methods have been proposed to learn invariance transformations directly from data. Most of them rely on observing at least two versions of the same entity. Many of these approaches are based on the assumption of time continuity where two subsequent frames of data are likely to show the same objects. Infinitesimal transformations between images can be learned, e. g., via a temporal Hebbian learning rule [16], by analyzing the correlation structure between features in subsequent frames [17, 18], or by fitting an explicitly time-dependent sparse multi-layer model [19]. Other studies approached the problem by learning the generators of a Lie group to best approximate observations which differ by transformations chosen from the group [20–22].

Other approaches are contrastive learning, which attempts to model similarity or dissimilarity relations between data points [23], and labeled graph matching [24–27], which is a method for image recognition where objects are represented as sets of features with distance relations between them; features are represented by nodes and relations by edges, and some transformations can be efficiently dealt with in graph view.

Apparently, the brain implements a much more effective approach to handling invariance transformations, given that humans can recognize objects or sounds under a wide variety of perceptual alterations and based on only a few training samples. While the algorithms behind this ability are unknown to science, it is clear that the brain differs structurally in significant ways from today’s artificial neural networks. In particular, the brain relies heavily on recurrent neural connections: Even in the visual cortex, synapses from the lateral geniculate nucleus of the thalamus (i.e., the feed-forward connections) form only 5–10% of the excitatory synapses [28]. The overwhelming majority of neocortical synapses connect neurons with neighbouring excitatory neurons located in the same cortical area and thus constitute recurrent connections.

In a recent publication, it was proposed that unsupervised Hebbian learning might suffice to encode invariance transformations in this recurrent cortical network [29]. The central idea is that each invariance transformation is reflected in a graph symmetry of the network: An invariance transformation acting on the space of possible observations leaves the probability distribution from which samples are drawn unaltered. The correlations between features of these observations are effectively two-dimensional projections of this probability distribution and thus they are also invariant under the transformation. Hebbian learning on the synapses between feature detectors turns these correlations into a weighted graph of synaptic strengths, which is then also invariant under the transformation. The problem of finding invariance transformations based on a set of observations is thus reduced to finding (approximate) symmetries in the graph of recurrent synaptic strengths.

The scope of the present paper is to substantiate this theoretical concept by testing it on numerical examples. We construct a simple recurrent network with synthetic data and show that invariances in the training data can be retrieved from the graph of synaptic connections. While our transformation-finding algorithm is very different from how the brain works, it illustrates that the information about invariances is encoded in the symmetries of the graph. This encoding does in fact appear to be biologically plausible and the reader is referred to [29] for a discussion of related experimental observations from neuroscience. We emphasize that our study aims at demonstrating the feasibility of the described encoding scheme for invariances, but it cannot answer the question how this encoding can be used efficiently by the brain or by artificial algorithms to solve practically relevant computational problems.

This paper is organized as follows: We elaborate on the concept of invariance transformations in Section 2 and introduce the algorithmic framework for finding the invariances in Section 3. In Section 4 we present the results of our numerical study which constitute the first computational corroboration of the concept proposed in [29]. Pseudocode of the discussed algorithms can be found in Appendix A.

## 2. Invariance Transformations

We begin by defining the term *invariance transformation* following the assumptions made in [29].

We consider a learning process in which observations are repeatedly sampled from a discrete probability distribution  $\Psi : \mathfrak{F} \rightarrow [0, 1]$  defined on a *feature space*  $\mathfrak{F} = \{0, 1\}^n$  for some  $n \in \mathbb{N}$  and  $n \gg 1$ . In our discussions below, we interpret each dimension of  $\mathfrak{F}$  as a pixel in an image which can be off or on. Yet, as the name “feature space” suggests, the proposed concept is also applicable to the case where each dimension represents aggregated information like features in an image that are either present or not and can also be generalized to other perceptual modalities.

We define an invariance transformation as a permutation of the coordinates of  $\mathfrak{F}$  which leaves  $\Psi$  unaltered. More precisely, an invariance transformation is a permutation  $T : \mathfrak{F} \rightarrow \mathfrak{F}$  such that  $\Psi(Tx) = \Psi(x)$  for all  $x \in \mathfrak{F}$ .

This implies that the consecutive application of two or more transformations is also a valid transformation. As the inverse of a transformation is also a transformation, the set of transformations forms a subgroup of the symmetric group on  $n$  elements. Thus, the set of invariance transformations is finite and it is sufficient to find a set of generators of this subgroup.

Our objective is to learn invariance transformations based on a limited number of samples drawn from the probability distribution  $\Psi$ . Unfortunately, it is not possible to measure  $\Psi$  exactly as it is only possible to estimate it based on the samples that we draw from it. An approach to

dealing with this problem is based on the observation that symmetries of  $\Psi$  are “inherited” by its projections to lower-dimensional spaces. In particular, consider the one- and two-dimensional marginal distributions

$$\Psi_i(x_i) = \sum_{x_j \in \mathbb{Z}: j \neq i} \Psi(x_1, \dots, x_n) \quad \text{and} \quad \Psi_{i,j}(x_i, x_j) = \sum_{x_k \in \mathbb{Z}: k \neq i, j} \Psi(x_1, \dots, x_n) \quad (1)$$

for  $i, j \in \{1, \dots, n\}, i \neq j$ . Then, if  $T$  is an invariance transformation of  $\Psi$  acting as

$$T(x_1, \dots, x_n) = (x_{\tau(1)}, \dots, x_{\tau(n)}) \quad (2)$$

for some permutation  $\tau$  of  $\{1, \dots, n\}$ , we have

$$\begin{aligned} \Psi_i(y) &= \Psi_{\tau(i)}(y) \quad \text{for } y \in \{0, 1\} \quad \text{and} \\ \Psi_{i,j}(y) &= \Psi_{\tau(i), \tau(j)}(y) \quad \text{for } y \in \{0, 1\}^2. \end{aligned} \quad (3)$$

The proof of these statements is straight-forward and can be found in the appendix of [29]. Since the marginal distributions  $\Psi_{i,j}$  are defined on only four points each, a limited number of samples can suffice to estimate them with good accuracy.

Moreover, for fixed indices  $i, j$ , the distribution  $\Psi_{i,j}$  is uniquely characterized by  $\Psi_{i,j}(1, 1)$ ,  $\Psi_i(1)$  and  $\Psi_j(1)$ .

Our approach is to find approximate invariance transformations of these two-dimensional marginal distributions.

We now argue that approximations of symmetries of these marginal distribution can be found by calculating approximate automorphisms of a suitable node- and edge-weighted undirected graph  $\mathcal{G} = (V, E)$  with weighted adjacency matrix  $A$ . As the graphs considered in this paper will not have loops, we slightly misuse notation by interpreting the node weights as edges from a node to itself and hence via the corresponding diagonal entries of  $A$ . Using this notation, an automorphism of such a graph  $\mathcal{G}$  is a permutation  $\pi : V \rightarrow V$  of the nodes such that

1.  $(i, j) \in E$  if and only if  $(\pi(i), \pi(j)) \in E$ ,
2.  $A_{i,j} = A_{\pi(i), \pi(j)} = A_{\pi(j), \pi(i)}$  for all  $\{i, j\} \in E$ , and
3.  $A_{i,i} = A_{\pi(i), \pi(i)}$  for all  $i \in V$ .

Note that the problem of finding graph automorphisms is computationally equivalent to the closely related graph isomorphism problem, and that their exact complexity status is not known [30–32]. In particular, no polynomial-time algorithm for solving either of the problems is known.

In order to translate our problem into the language of graphs, we define the *concurrency graph*  $\mathcal{G}$  as the complete graph on the set  $V = \{1, \dots, n\}$  with weighted adjacency matrix  $A_{i,j} := \Psi_{i,j}(1, 1)$  for  $i, j \in V, i \neq j$  and node weights  $A_{i,i} := \Psi_i(1)$  for  $i \in V$ .

From this definition, it follows that each automorphism of the concurrency graph yields an invariance transformation of the two-dimensional and one-dimensional marginal distribution and vice versa. Thus, by approximating this graph from observed samples of  $\Psi$  and identifying approximate symmetries of its two-dimensional marginal distribution, we can find candidates for potential invariance transformations. Note however, that the inverse is not always true: Observing a symmetry of the marginal distribution does not imply the existence of a corresponding symmetry in  $\Psi$ . We give an example for this phenomenon in Section 4.1.

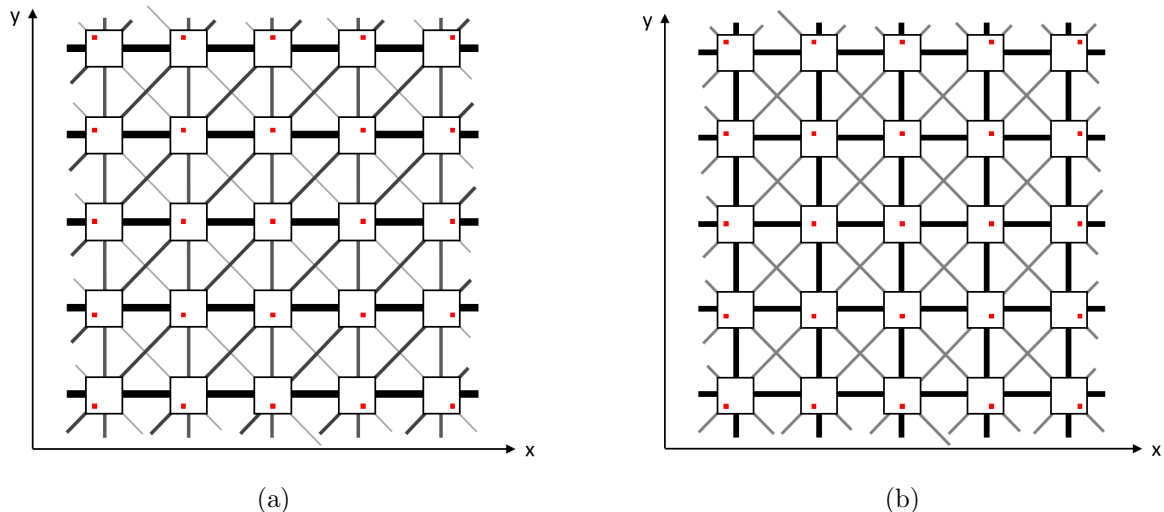


Figure 1: Two schematic examples of concurrence graphs. The feature space  $\mathfrak{F}$  in each example consists of pixels at different positions  $(x, y)$  in an image. Each little box represents a feature detector which responds to a single pixel and the position of the red dot within the box corresponds to the position of this pixel in the image plane. The line strength between boxes represents the correlation between the corresponding features and thus also the weight of an edge in the concurrence graph. Correlations are typically strongest between neighboring pixels and therefore lines between distant boxes are omitted for better readability.

Figure 1 shows a simple example of how certain invariances give rise to corresponding graph symmetries. The graph in Figure 1a exhibits translational symmetries but no rotational one. It might have been trained, for example, on images containing upright letters at random positions. The fact that connections between horizontal neighbors are stronger than those between vertical ones is then a direct reflection of the internal correlation structure of the letters used. For example, the letter “E” contains more horizontal line segments than vertical ones and thus induced stronger correlations between horizontal neighbors. The graph in Figure 1b is symmetric under translations as well as rotation by  $90^\circ$  and reflections. It might have been trained on images of symmetric letters at random positions and random orientations in  $90^\circ$  steps, for example.

### 3. Algorithmic Framework

#### 3.1. Mathematical Background

Surprisingly, there are only few other approaches for calculating approximate automorphisms of graphs. The vast majority of the research in this field of graph theory is concerned with either the related *graph isomorphism* problem or with finding *exact* automorphisms. Although there are results related to finding and repairing imperfect automorphisms by adding additional edges [33], by defining symmetry in graphs using other concepts than automorphisms [34, 35], or by applying techniques from the field of fractional graph theory [36–39], most techniques developed for the graph isomorphism problem cannot be applied for the graph automorphism problem. For more details on different techniques used for the exact and inexact graph isomorphism problem, we refer to [34, 40–42] and the references therein.

Our symmetry calculation algorithm employs *(mixed) integer programming*, a widely used tool in the area of discrete optimization [43, 44].

Our formulation is based on an integer program presented in [37] that allows for calculating exact graph automorphisms of a graph with adjacency matrix  $A$  by solving the feasibility problem Eq. (4).

$$\begin{array}{ll}
\text{find} & P \\
\text{subject to} & 0 = AP - PA \\
& P\mathbf{1} = \mathbf{1} \\
& P^T\mathbf{1} = \mathbf{1} \\
& P \in \{0, 1\}^{n \times n}
\end{array} \quad (4)
\qquad
\begin{array}{ll}
\text{min} & z \\
\text{subject to} & z \geq \|AP - PA\|_\infty \\
& P\mathbf{1} = \mathbf{1} \\
& P^T\mathbf{1} = \mathbf{1} \\
& P \in \{0, 1\}^{n \times n}
\end{array} \quad (5)$$

Here,  $\mathbf{1}$  denotes the vector with each entry being equal to 1 of suitable dimension. In this formulation, a matrix  $P \in \{0, 1\}^n$  is calculated subject to three conditions. The conditions  $P\mathbf{1} = \mathbf{1}$  and  $P^T\mathbf{1} = \mathbf{1}$  ensure that  $P$  is a permutation matrix, while the condition  $0 = AP - PA$  guarantees that  $P$  is in fact an automorphism of the graph.

Consequently, this problem has a solution if and only if  $P$  encodes an automorphism of a graph with adjacency matrix  $A$  [37].

However, we are not interested in finding exact, but approximate automorphisms.

To this end, we relax the constraint  $0 = AP - PA$  by minimizing the maximum norm of its right-hand side, which we achieve by bounding it by the continuous variable  $z$  and minimizing that instead.

This yields the mixed integer program (MIP) Eq. (5), where  $\|\cdot\|_\infty$  denotes the maximum norm.

Note that the non-linear expression in Eq. (5) can be reformulated into  $2n^2$  linear expressions.

The corresponding integer program thus has  $n^2 + 1$  variables (each entry of  $P$  as well as  $z$ ) and  $2n^2 + 2n$  constraints (bounding the value of  $z$  and forcing the row and column sums of  $P$  to be 1). Further note that this formulation is not only applicable for concurrence graphs as defined in Section 2 but general node- and edge-weighted undirected graphs.

### 3.2. Solving the MIP in Practice

For the remainder of this paper, we consider the special case of concurrence graphs. Thus, we let  $\mathcal{G}$  denote a concurrence graph and let  $A$  denote its weighted adjacency matrix with node weights on the diagonal.

For such a graph, solving Eq. (5) in practice turns out to be hard as the matrix  $A$  is typically dense and all  $n!$  permutation matrices  $P \in \mathbb{R}^{n \times n}$  are feasible solutions, i. e. solutions that fulfill all constraints. To overcome this difficulty, we implement an additional pre-processing routine. This routine constructs an incomplete permutation  $\tilde{P}$  by trying to fix the image of as many nodes of  $\mathcal{G}$  under  $\tilde{P}$  as possible. Formally, such an incomplete permutation is an injective function  $\tilde{P} : \tilde{V} \rightarrow V$  defined on a subset  $\tilde{V} \subseteq V$ . This guarantees that it is possible to extend an incomplete permutation  $\tilde{P}$  to a permutation on  $V$ .

This enables us to consider a reduced variant of the MIP in Eq. (5) that only needs to map nodes that are not already mapped by  $\tilde{P}$  onto nodes that are not already in the image of  $\tilde{P}$ . More precisely, given an incomplete permutation  $\tilde{P}_1 : \tilde{V}_1 \rightarrow V$ , the MIP has to find another incomplete permutation  $\tilde{P}_2 : \tilde{V}_2 := V \setminus \tilde{V}_1 \rightarrow V$  such that the function  $P : V \rightarrow V$  that acts as  $\tilde{P}_1$  on  $\tilde{V}_1$  and

as  $\tilde{P}_2$  on  $\tilde{V}_2$  is a permutation on  $V$ . For finding such a function  $\tilde{P}_2$ , in the reduced MIP only the rows corresponding to  $\tilde{V}_2$  and the columns corresponding to  $V \setminus \text{Im } \tilde{P}_1$  have to be considered, where  $\text{Im } \tilde{P}_1$  denotes the image of the function  $\tilde{P}_1$ .

In principle, any heuristic calculating an incomplete permutation can be chosen for the pre-processing step. The heuristic we chose is based on the following simple observation: If a transformation  $P$  maps node  $x$  to  $x'$ , then it can map any node  $z$  adjacent to  $x$  only to those nodes  $z'$  adjacent to  $x'$  for which the edge  $\{x, z\}$  has the same weight as the edge  $\{x', z'\}$ . That way, when committing to an image  $\tilde{P}(x)$  of node  $x$ , we not only gain information about  $x$  itself but also about the remaining possible images of all other nodes. The general concept is known as *constraint propagation* [45]. A pseudocode description of this algorithm is given in Appendix A, Algorithm 1.

We now describe the algorithm for actually finding the invariance transformations in more detail and refer to Appendix A, Algorithm 2, for a pseudocode description. Assume that we are given the set of nodes  $V = \{1, \dots, n\}$  and let  $L$  denote a set of bins, which form equivalence classes of edges. They are used to decide when two edge weights are considered to be identical, which is the case if and only if they belong to the same bin. We discuss the binning step in more detail later.

When trying to find an automorphism  $P$ , the algorithm maintains a set  $\tilde{P}_x$  of possible images for each node  $x \in V$ . Initially, when every node could potentially be mapped to any other, we have  $\tilde{P}_x = V$  for all  $x \in V$ . The algorithm then tries to refine these sets until for each node, this set has either size 0 or 1. Ideally, one would like to have  $|\tilde{P}_x| = 1$  for all  $x \in V$ , with the intersection of each pair of sets being empty. This then implies that each node has a unique matching partner, hence  $\tilde{P}$  is a “proper” permutation. However, as we deal with imperfect data, we accept that a certain “allowed percentage” of the nodes can not be mapped, which is encoded by the corresponding set  $\tilde{P}_x$  being empty. This percentage is called *fault tolerance*. Thus, if too many sets  $\tilde{P}_x$  are empty at some point, we discard the currently constructed incomplete permutation  $\tilde{P}$ .

The algorithm now works as follows. First, it checks whether the current incomplete permutation can be discarded. If this is not the case, it checks whether at least one node can still be potentially mapped to multiple other nodes, branches, and tests all these possibilities. To locally minimize the number of times the algorithm branches, it chooses the index  $x$  for which  $\tilde{P}_x$  has the least number of elements larger than one.

In each branching step trying to map  $x$  to some potential target  $x' \in \tilde{P}_x$ , the algorithm now applies the core idea to reduce the sets  $\tilde{P}_1, \dots, \tilde{P}_n$  even further. For all bins  $\ell \in L$ , it calculates the set  $S_\ell$  of all nodes  $z'$  such that the edge  $\{x', z'\}$  is in bin  $\ell$ . Now, for all edges  $\{x, z\}$  that are also in bin  $\ell$ , the node  $z$  can only be mapped to elements in  $S_\ell$ , so  $\tilde{P}_z$  is replaced by its intersection with  $S_\ell$ . This filtering procedure is applied recursively to further narrow down all remaining  $\tilde{P}_x$  that contain more than one element until any of the exit conditions above is met. Depending on which one it was, the incomplete permutation is either discarded or stored for potential later use in the reduced MIP.

## 4. Numerical Experiments

To verify our concepts, we implemented a prototype of the algorithmic framework presented in Section 3. We set up the integer program Eq. (5) using the Python package Pyomo [46, 47] and solved it using the academic solver SCIP [48], Version 7.0.3. The pre-processing was implemented explicitly and the handling of permutations was mainly done using SymPy [49] and NumPy [50]. The implementation that has been used for the study presented here is available on github [51].

## 4.1. Synthetic Data

Our experiments were executed on synthetic data, all of which consists of a variety of two-dimensional finite surfaces with binary or colored pixels. The opposing sides of these surfaces are connected with each other, so the surfaces are embedded on a torus. For simplicity, we depict these surfaces as two-dimensional Cartesian meshes.

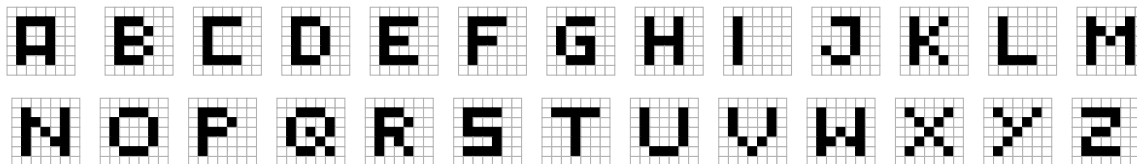


Figure 2: Letters composed of binary or colored pixels are used in our synthetic data.

For our experiments, we designed five different sets of synthetic data, also called *worlds*. A summary, as well as short-hand names for the worlds we used, can be found in Table 1. We consider four different sets of surfaces with  $20 \times 10$ ,  $20 \times 20$ ,  $15 \times 15$  and  $13 \times 7$  pixels. In each setting, an individual observation consists of two or three letters placed next to each other with one pixel space between them on the torus. Our embedding of the letters is shown in Figure 2. In the world T, each letter from Figure 2 is used.

Thus, the full data set consists of all pairs of letters at all possible positions. Some example observations from this world are shown in Figure 3. We also considered two worlds of size  $15 \times 15$ , called TR1 and TR2.

In TR1, again all letters are used, while in TR2, we only considered letters that do not exhibit axial symmetry in our representation, meaning only the 11 letters F, G, J, L, N, P, Q, R, S, Y, Z are used. In both cases, each group of letters can be rotated by  $0^\circ$ ,  $90^\circ$ ,  $180^\circ$  or  $270^\circ$ .

In TC, the same reduced set of letters is used, but each letter is assigned one of three colors and the size is reduced to  $13 \times 7$  pixels. Lastly, we considered TL, a world of size  $20 \times 20$  with three letters, rotations and translations. In addition to the scenario before, individual letters can move vertically as well relative to the word axis, meaning that if the word is rotated by  $90^\circ/270^\circ$ , the individual letter translations are horizontal. The third letter and additional degrees of freedom lead to the number of observations to be drastically higher than for the other worlds.

The properties of all worlds are summarized in Table 1.

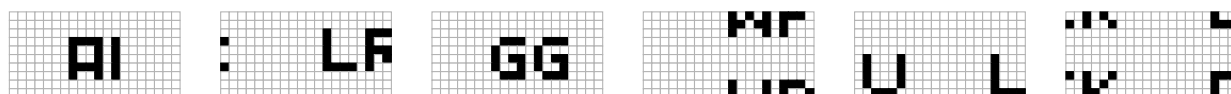


Figure 3: Example observations of the test world T. The entities are “words”, made up of two letters of the English alphabet placed next to each other.

The letter transformations described above, which are used to generate the observations, are related to the invariance transformations, which we hope to recover, in the following way. If a letter transformation can be expressed as a global permutation of pixels (like global rotations or translations) or global color change then it automatically leads to a symmetry of  $\Psi$  and thus an invariance transformation. This is not true for every letter transformation though: For example, a vertical shift of an individual letter (as in TL) cannot be written as one permutation matrix acting on  $\mathfrak{F}$ , because the position of the letters differ between the observations. Thus there is no reason to



Name	Size	Alphabet	Letters per obser- vation	Letter transformations		# of different possible observations	Expected graph symmetries
				Type	Count		
T	$20 \times 10$	full	2	T	200	135,200	400
TR1	$15 \times 15$	full	2	T, R	900	571,950	900
TR2	$15 \times 15$	F, G, ..., Z	2	T, R	900	104,850	900
TC	$13 \times 7$	F, G, ..., Z	2	T, C	546	99,099	1,092
TL	$20 \times 20$	full	3	T+, R	640,000	11,248,640,000	1,600

Table 1: Overview over the experiments performed to test the algorithmic framework. The letters R, C signify rotational and color change transformations, respectively. T indicates global translations only, while T+ means that horizontal translations are only applicable to all letters simultaneously and vertical translations can be applied to individual letters. Note that some combinations of letters like “OO” are symmetric under rotation and therefore the number of different possible observations can be smaller than the combinatorial product of the numbers of possible letters and transformations.

expect an invariance transformation or a symmetry of the concurrence graph to arise from such letter transformations. Consequently, the number of graph symmetries in TL is much smaller than the number of letter transformations (cf. Table 1).

Conversely, our particular method of finding invariances can lead to false positives. As explained in Section 2, we replace the hard problem of finding symmetries of  $\Psi$  by the simpler one of finding symmetries in its projections onto one or two coordinates of  $\mathfrak{F}$  (i.e. the concurrence graph), and while a symmetry of  $\Psi$  always gives rise to a symmetry of the concurrence graph, the opposite is not always true. In particular, given a torus and global translational symmetry on its surface, the marginal distribution does not change under a rotation by  $180^\circ$  around any one point:  $\Psi_i$  is the same for all  $i$  anyway, and  $\Psi_{ij}$  only depends on the relative vector from  $i$  to  $j$ . A  $180^\circ$  rotation maps  $(i, j)$  to  $(i', j')$ , inverting the relative vector. But since  $\Psi_{ij} = \Psi_{ji}$  we also have  $\Psi_{ij} = \Psi_{i'j'}$ , making the  $180^\circ$  rotation a symmetry of the concurrence graph. But if  $x \in \mathfrak{F}$  denotes some observation that does not possess rotational symmetry by itself, then a rotated version  $x_r \in \mathfrak{F}$  of this feature does not necessarily fulfill  $\Psi(x_r) = \Psi(x)$ , meaning that  $\Psi$  itself does not possess this rotational symmetry. Consequently, the  $180^\circ$  rotation is a false positive for distributions  $\Psi$  which are invariant under translation but not under rotation and therefore in the worlds T and TC we expect our algorithm to find twice as many graph symmetries as there are letter transformations, cf. Table 1.

## 4.2. Specific Test Cases and Parameters

Figure 4 summarizes the overall workflow of our numerical studies. For all experiments we first generated synthetic data sets given by observations of the worlds described in Table 1. These data sets contained different percentages of all possible observations in order to be able to study how effectively the method generalizes from few observations. To perform a broad analysis, the used ratios ranging from 5% up to 100% in increments of 5% or 10% for all worlds but TL, adjusting the grid resolution in a trade-off of exploration of possible performance variance and runtime. For the world TL, our goal was to test the limits of our method, we considered a range from 0.004% to 5%. For each data set, we then calculated the concurrence graph and applied Algorithm 2 to find the invariance transformations.

The algorithm depends on several hyperparameters, namely the error value limit, as well as the bandwidth and fault tolerance. The latter two parameters define the resilience of the algorithm

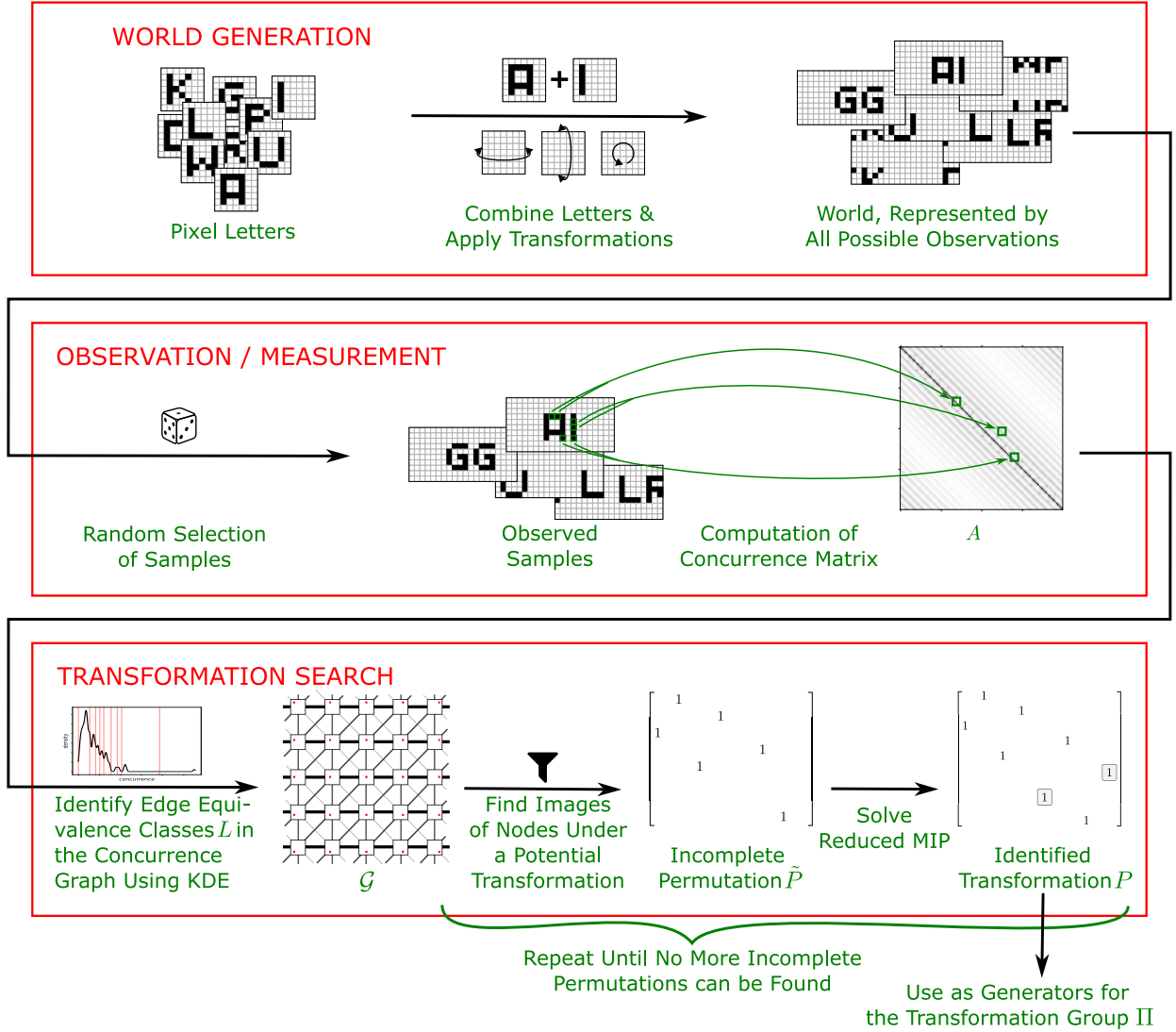


Figure 4: Workflow of the numerical experiments

w.r.t. perturbances in the edge weights of the adjacency matrix by defining the classes of weights which are considered the same (i.e. binning) as well as the tolerance to possible misclassifications, while the error value limit is explained in more detail below. We chose bins of variable sizes using a kernel density estimation, a non-parametric method to approximate the underlying density function from a given set of samples [52]. We used a Gaussian kernel and performed each experiment with a fixed bandwidth.

From the edge weight density estimation, we defined the bins to be the halfway-points between two consecutive maxima. Additionally, we defined a-priori that edges with zero weight are grouped separately.

For our experiments, it is sufficient to find a suitable set of generators of the respective symmetry group. For the worlds with only translational and rotational invariances, one such set can consist, for example, of three permutations. These represent horizontal, vertical and rotational transformations, respectively the  $180^\circ$  rotation. In the experiment involving colored letters, this set has to be extended by two additional generators. They realize the color changes via an exchange of either all three or just two colors.

In the presented framework, solving the MIP is by far the hardest part and it is desirable to minimize the number of times it has to be solved. To this end, we employed the following strategy: We maintain a set  $\mathcal{G}$  of generators for the symmetry group as well as the symmetry group  $\Pi$  itself which we both initialize as empty sets. We then calculate the set of all incomplete permutations using the pre-processing step described in Section 3.2 resp. Algorithm 1 in Appendix A and iterate over all members of this set: for each incomplete permutation  $\tilde{P}$  which is then only defined on a subset  $\tilde{V}$  of  $V$ , we check if there is already a permutation  $P \in \Pi$  that acts exactly as the currently considered incomplete permutation  $\tilde{P}$  on  $\tilde{V}$ . If this is the case, then there is no need to complete  $\tilde{P}$  and we continue with the next incomplete permutation. Otherwise, we complete  $\tilde{P}$  using the reduced MIP and obtain a full permutation  $P : V \rightarrow V$ . We then verify if this permutation is a sufficiently good approximate automorphism by calculating the deviation value  $\|AP - PA\|_\infty$  and checking if it is smaller than a value which we will call the error value limit. If this is the case, then  $P$  is added to the set of generators, i.e., we set  $\mathcal{G} := \mathcal{G} \cup \{P\}$ . Consequently, we calculate all permutations that can now be created using this larger set of generators, include these in  $\Pi$  and iterate. This exploitation of the group structure of the permutations enables us to find a significant number of new permutations by solving the MIP only a few times.

A pseudocode summary of this algorithm can be found in Appendix A, Algorithm 2.

### 4.3. Results and Discussion

We found that pre-processing enabled us to find transformations in cases in which solving the MIP (5) by itself was computationally intractable because of the huge number of feasible solutions. We attribute this to the structure of the worlds from which we sampled our data, since only very specific transformations are present in our experiments. For translations, fixing  $\tilde{P}(i)$  uniquely determines all  $\tilde{P}(j)$  for  $j \neq i$ . For rotations, this is not as strict as our observations are embedded on a torus. However, this does not trivialize the problem of finding these invariances as the matrix  $A$  and the concurrence graph itself generally does not reflect any information about the structure of the world. Nevertheless, a commitment to a specific mapping reduced the possible targets of the other nodes considerably, since the structure of the worlds allowed effective propagation of the consequent constraints. In particular, committing to incompatible maps of only two nodes is often enough for the algorithm to return early and to try and find another way to construct the permutation.

For the worlds T, TR1, TR2 and TC with few possible observations, we are able to reliably find the correct set of transformations, even if only 20% or 30% of the observations are available. For the TL world, we succeeded in finding the transformations when 499,982 unique observations, i.e., 0.004% of all possible observations of this world were available. Especially in this range of percentages, the choice of a suitable bandwidth and fault tolerance parameter are important and not trivial, as can be with the example of the experiments conducted on TR1, see Figure 5. While specifically the fault tolerance has little to no influence on finding the transformations if the observed statistics are sufficiently good, setting it too low resp. too high can result in not all resp. too many transformation generators being calculated. Here, used bandwidth was  $6.4633 \cdot 10^{-5}$ ; the values chosen for the other worlds were in the same order of magnitude, yet specific to the worlds themselves and similarly extensive experiments on the worlds T, TR2 and TC allowed for the same conclusions.

It is not surprising that both a low fault tolerance and a limited number of observations result in too few found transformations. Interestingly, our algorithm often identifies additional permutations as transformations for intermediate percentages and a high fault tolerance. This happens as the statistics are only slightly disturbed in this range of percentages, resulting in permutations that are “nearly” transformations having an error value that is below the error value limit. An example

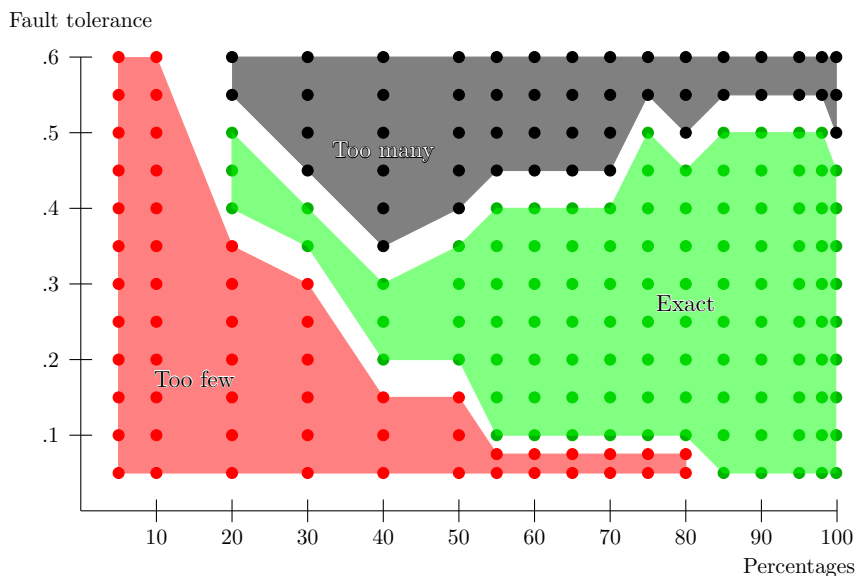


Figure 5: Results of the experiments performed in the  $15 \times 15$ -sized world with translational and rotational invariance (TR1). We used a KDE bandwidth of  $6.4633 \cdot 10^{-5}$  and an error value limit of 0.01 for all experiments. Different colors of the symbols denote that too few/the correct number of/too many transformations were calculated. The colored regions are included to improve the presentation of the results.

is a mirror operation along either the horizontal or the vertical axis as most of our letters either exhibit such a mirror symmetry directly or are only a few pixels off, see Figure 2. For this reason, we decided to exclude letters exhibiting axial symmetry in the two experiments TR2 and TC.

With this reduced set of observations, the algorithm then does not find too many transformations compared to the case where all letters are used for creating the observations. However, finding the expected transformations is then harder, as the total number of unique observations is smaller and statistics deteriorate faster.

Values for the bandwidth and the error value limit which are too high might cause permutations to be accepted which do not correspond to invariance transformations as defined in Section 2, while values which are too low might preclude (some) invariance transformations from being found.

## 5. Conclusion

In this paper, we numerically confirmed and illustrated that invariance transformations can be learned in an unsupervised way from the statistics of input data and without prior assumptions on the perceptual modality. By translating the features that are present in the data and their concurrences into a graph, these transformations can be considered as approximate graph automorphisms. We implemented a prototype that uses a pre-solving heuristic and a mixed integer program in order to verify that these invariance transformations can be found in practice. The concept of directly defining approximate graph algorithms via solutions of a mixed integer program is a novel approach, and the authors are not aware of any publications investigating approximate automorphisms in this way.

While the presented concept for inferring invariance transformations is inspired by the field of neuroscience, the algorithmic framework is certainly very different from how biological brains work. However, the presented theory and algorithm establish that invariance transformations can in principle be encoded in the structure of the graph of recurrent synaptic connections.

## A. Algorithms

```

Input: concurrence graph  $\mathcal{G}$ , bins  $L$ 
Output: set found_trafos containing all possible transformations
1 Algorithm calculate_incomplete_permutations( $\mathcal{G}, L$ )
2   found_perms :=  $\emptyset$ 
3    $\tilde{P}_i := V$  for all  $i \in V$  //  $\tilde{P}_z$ : set of potential targets for  $z$ 
4   calculate_perms( $\tilde{P}_1, \dots, \tilde{P}_n$ )
5   return found_perms
1 Subroutine calculate_perms( $\tilde{P}_1, \dots, \tilde{P}_n$ )
2   if  $|\tilde{P}_i| = 0$  for more than an “fault tolerance (percentage)” of nodes  $i \in V$  then
3     return // incompl. permutation is discarded
4   else if there are nodes  $z \in V$  with  $|\tilde{P}_z| > 1$  then
5      $x \leftarrow$  node  $z$  minimizing  $|\tilde{P}_z|$  s.t.  $|\tilde{P}_z| > 1$  // node with few pot. targets
6     foreach  $x' \in \tilde{P}_x$  do
7        $\hat{P}_z \leftarrow \tilde{P}_z$  for all  $z \in V$  // copy potential targets
8       foreach bin  $\ell \in L$  do
9          $S_\ell \leftarrow$  set of nodes  $z'$  adjacent to  $x'$  s.t. edge  $\{x', z'\}$  is in bin  $\ell$ 
10        foreach edge  $\{x, z\}$  in bin  $\ell$  do
11           $\hat{P}_z \leftarrow \tilde{P}_z \cap S_\ell$ 
12        calculate_perms( $\hat{P}_1, \dots, \hat{P}_n$ )
13   else
14     Add  $\tilde{P}$  to found_perms

```

**Algorithm 1:** The pre-solving routine for calculating incomplete permutations.

```

Input: concurrence graph  $\mathcal{G}$ , bins  $L$ 
Output: set of generators
1  $\Pi \leftarrow \emptyset$  // initialize empty group of found permutations
2 found_perms  $\leftarrow$  calculate_incomplete_permutations( $\mathcal{G}, L$ )
3 foreach  $\tilde{P} \in$  found_perms do
4   if  $\tilde{P}$  does not match any permutation in  $\Pi$  then
5     Calculate complete permutation  $P$  from  $\tilde{P}$  using the reduced MIP
6     if maximum entry of  $|AP - PA|$  does not exceed error value limit  $\varepsilon$  then
7       Update  $\Pi$  with  $P$ 
8 return generators of  $\Pi$ 

```

**Algorithm 2:** The algorithm for calculating the invariance transformations.

## References

- [1] R. Bro and A. K. Smilde. “Principal Component Analysis”. In: *Analytical Methods* 6.9 (2014), pp. 2812–2831. DOI: 10.1039/C3AY41907J.
- [2] I. T. Jolliffe. “Principal Component Analysis”. In: *Encyclopedia of Statistics in Behavioral Science*. Ed. by B. S. Everitt and D. C. Howell. Chichester, UK: John Wiley & Sons, Ltd, Oct. 15, 2005, bsa501. DOI: 10.1002/0470013192.bsa501.
- [3] I. T. Jolliffe and J. Cadima. “Principal Component Analysis: A Review and Recent Developments”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374.2065 (Apr. 13, 2016), p. 20150202. DOI: 10.1098/rsta.2015.0202.
- [4] M. S. Lewicki and T. J. Sejnowski. “Learning Overcomplete Representations”. In: *Neural Computation* 12.2 (2000), pp. 337–365. DOI: 10.1162/089976600300015826.
- [5] A. Hyvärinen and E. Oja. “Independent Component Analysis: Algorithms and Applications”. In: *Neural Networks* 13.4-5 (June 2000), pp. 411–430. DOI: 10.1016/S0893-6080(00)00026-5.
- [6] D. D. Lee and H. S. Seung. “Learning the parts of objects by non-negative matrix factorization”. In: *Nature* 401.6755 (Oct. 1999), pp. 788–791. DOI: 10.1038/44565.
- [7] Weixiang Liu, Nanning Zheng, and Xiaofeng Lu. “Non-Negative Matrix Factorization for Visual Coding”. In: *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03)*. International Conference on Acoustics, Speech and Signal Processing (ICASSP'03). Vol. 3. Hong Kong, China: IEEE, 2003, pp. III-293–6. DOI: 10.1109/ICASSP.2003.1199270.
- [8] P. Hoyer. “Non-Negative Sparse Coding”. In: *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*. Neural Networks for Signal Processing XII. 2002 IEEE Signal Processing Society Workshop. Martigny, Switzerland: IEEE, 2002, pp. 557–565. DOI: 10.1109/NNSP.2002.1030067.
- [9] M. Elad. *Sparse and Redundant Representations*. Springer, 2010.
- [10] H. Lee et al. “Efficient Sparse Coding Algorithms”. In: *Advances in Neural Information Processing Systems*. Ed. by B. Schölkopf, J. Platt, and T. Hoffman. Vol. 19. MIT Press, 2007.
- [11] W. S. Geisler. “Visual Perception and the Statistical Properties of Natural Scenes”. In: *Annual Review of Psychology* 59.1 (Jan. 2008), pp. 167–192. DOI: 10.1146/annurev.psych.58.110405.085632.
- [12] H. E. Gerhard, F. A. Wichmann, and M. Bethge. “How Sensitive Is the Human Visual System to the Local Statistics of Natural Images?” In: *PLoS Computational Biology* 9.1 (Jan. 24, 2013). Ed. by K. P. Kording, e1002873. DOI: 10.1371/journal.pcbi.1002873.
- [13] A. Šošić and M. Winkel. *Transform-Invariant Non-Negative Matrix Factorization (TNMF)*. <https://github.com/emdgroup/tnmf>. Version 0.1.0. Dec. 2021.
- [14] K. Sohn and H. Lee. “Learning Invariant Representations with Local Transformations”. In: *ICML*. 2012.
- [15] F. Anselmi et al. “Unsupervised learning of invariant representations”. In: *Theoretical Computer Science* 633 (June 2016), pp. 112–121. DOI: 10.1016/j.tcs.2015.06.048.
- [16] P. Földiák. “Learning Invariance from Transformation Sequences”. In: *Neural Computation* 3.2 (June 1991), pp. 194–200. DOI: 10.1162/neco.1991.3.2.194.

- [17] M. Bethge, S. Gerwinn, and J. H. Macke. “Unsupervised learning of a steerable basis for invariant image representations”. In: *Electronic Imaging 2007*. Ed. by B. E. Rogowitz, T. N. Pappas, and S. J. Daly. San Jose, CA, USA, Feb. 15, 2007, p. 64920C. DOI: 10.1117/12.711119.
- [18] R. Memisevic. “Learning to Relate Images”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (Aug. 2013), pp. 1829–1846. DOI: 10.1109/TPAMI.2013.53.
- [19] C. F. Cadieu and B. A. Olshausen. “Learning Intermediate-Level Representations of Form and Motion from Natural Movies”. In: *Neural Computation* 24.4 (Apr. 2012), pp. 827–866. DOI: 10.1162/NECO\_a\_00247.
- [20] R. P. N. Rao and D. L. Ruderman. “Learning Lie Groups for Invariant Visual Perception”. In: *Proceedings of the 11th International Conference on Neural Information Processing Systems*. NIPS’98. Denver, CO: MIT Press, 1998, pp. 810–816.
- [21] X. Miao and R. P. N. Rao. “Learning the Lie Groups of Visual Invariance”. In: *Neural Computation* 19.10 (Oct. 2007), pp. 2665–2693. DOI: 10.1162/neco.2007.19.10.2665.
- [22] T. Cohen and M. Welling. “Learning the Irreducible Representations of Commutative Lie Groups”. In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML’14. Beijing, China: JMLR.org, 2014, II–1755–II–1763.
- [23] P. H. Le-Khac, G. Healy, and A. F. Smeaton. “Contrastive Representation Learning: A Framework and Review”. In: *IEEE Access* 8 (2020), pp. 193907–193934. DOI: 10.1109/ACCESS.2020.3031549.
- [24] C. von der Malsburg and E. Bienenstock. “Statistical Coding and Short-Term Synaptic Plasticity: A Scheme for Knowledge Representation in the Brain”. In: *Disordered Systems and Biological Organization*. Ed. by E. Bienenstock, F. F. Soulié, and G. Weisbuch. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 247–272. DOI: 10.1007/978-3-642-82657-3\_26.
- [25] C. von der Malsburg. “Pattern recognition by labeled graph matching”. In: *Neural Networks* 1.2 (Jan. 1988), pp. 141–148. DOI: 10.1016/0893-6080(88)90016-0.
- [26] G. Westphal, C. von der Malsburg, and R. P. Würtz. “Feature-Driven Emergence of Model Graphs for Object Recognition and Categorization”. In: *Applied Pattern Recognition*. Ed. by H. Bunke, A. Kandel, and M. Last. Red. by J. Kacprzyk. Vol. 91. Series Title: Studies in Computational Intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 155–199. DOI: 10.1007/978-3-540-76831-9\_7.
- [27] M. Lades et al. “Distortion invariant object recognition in the dynamic link architecture”. In: *IEEE Transactions on Computers* 42.3 (Mar. 1993), pp. 300–311. DOI: 10.1109/12.210173.
- [28] R. J. Douglas and K. A. Martin. “Recurrent neuronal circuits in the neocortex”. In: *Current Biology* 17.13 (July 2007), R496–R500. DOI: 10.1016/j.cub.2007.04.024.
- [29] H. Linde. *Does the Brain Infer Invariance Transformations from Graph Symmetries?* Accepted for publication in *IEEE Transactions on Cognitive and Developmental Systems*. 2021. URL: <https://arxiv.org/abs/2111.06174>.
- [30] J. Köbler, U. Schöning, and J. Torán. “Graph Isomorphism Is Low for PP”. In: *Computational Complexity* 2.4 (Dec. 1992), pp. 301–330. DOI: 10.1007/BF01200427.
- [31] J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity*. Progress in Theoretical Computer Science. Boston: Birkhäuser, 1993. 160 pp.
- [32] V. Arvind and P. P. Kurur. “Graph Isomorphism Is in SPP”. In: *Information and Computation* 204.5 (May 2006), pp. 835–852. DOI: 10.1016/j.ic.2006.02.002.



- [33] I. Leifer et al. *Symmetry-Driven Link Prediction in Networks through Pseudobalanced Coloring Optimization*. Nov. 15, 2021. URL: <http://arxiv.org/abs/2111.07821> (visited on 11/18/2021).
- [34] F. Morone and H. A. Makse. “Symmetry Group Factorization Reveals the Structure-Function Relation in the Neural Connectome of *Caenorhabditis Elegans*”. In: *Nature Communications* 10.1 (Dec. 2019), p. 4961. DOI: 10.1038/s41467-019-12675-8.
- [35] P. Boldi, I. Leifer, and H. A. Makse. *Quasifibrations of Graphs to Find Symmetries in Biological Networks*. Nov. 12, 2021. URL: <http://arxiv.org/abs/2111.06999> (visited on 11/18/2021).
- [36] K. Kersting et al. “Power Iterated Color Refinement”. In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*. AAAI’14. Québec City, Québec, Canada: AAAI Press, 2014, pp. 1904–1910.
- [37] R. Takapoui and S. Boyd. *Linear Programming Heuristics for the Graph Isomorphism Problem*. Nov. 2, 2016. URL: <http://arxiv.org/abs/1611.00711>.
- [38] E. R. Scheinerman and D. H. Ullman. *Fractional Graph Theory*. John Wiley & Sons, Inc, 2008.
- [39] M. Grohe et al. “Color Refinement and Its Applications”. In: *An Introduction to Lifted Probabilistic Inference*. Ed. by G. Van den Broeck, K. Kersting, and S. Natarajan. Neural Information Processing Series. The MIT Press, 2021, p. 20.
- [40] N. Shervashidze et al. “Weisfeiler-Lehman Graph Kernels”. In: *Journal of Machine Learning Research* 12.77 (2011), pp. 2539–2561.
- [41] B. D. McKay and A. Piperno. “Practical Graph Isomorphism, II”. In: *Journal of Symbolic Computation* 60 (Jan. 2014), pp. 94–112. DOI: 10.1016/j.jsc.2013.09.003.
- [42] Z. Abu-Aisheh et al. “An Exact Graph Edit Distance Algorithm for Solving Pattern Recognition Problems:” in: *Proceedings of the International Conference on Pattern Recognition Applications and Methods*. International Conference on Pattern Recognition Applications and Methods. Lisbon, Portugal: SCITEPRESS - Science and and Technology Publications, 2015, pp. 271–278. DOI: 10.5220/0005209202710278.
- [43] A. Schrijver. *Theory of Linear and Integer Programming*. Nachdr. Wiley-Interscience Series in Discrete Mathematics and Optimization. Chichester Weinheim: Wiley, 2011. 471 pp.
- [44] M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer Programming*. Vol. 271. Graduate Texts in Mathematics. Cham: Springer International Publishing, 2014. DOI: 10.1007/978-3-319-11008-0.
- [45] S. J. Russell and P. Norvig. *Artificial intelligence : a modern approach*. Edition: Fourth edition, global edition ISBN: 9781292401171 Place: Harlow Series: Pearson series in artificial intelligence. 2021.
- [46] M. L. Bynum et al. *Pyomo – Optimization Modeling in Python*. Third. Vol. 67. Springer Science & Business Media, 2021.
- [47] W. E. Hart, J.-P. Watson, and D. L. Woodruff. “Pyomo: Modeling and Solving Mathematical Programs in Python”. In: *Mathematical Programming Computation* 3.3 (2011), pp. 219–260.
- [48] G. Gamrath et al. *The SCIP Optimization Suite 7.0*. Technical Report. Optimization Online, Mar. 2020.
- [49] A. Meurer et al. “SymPy: Symbolic Computing in Python”. In: *PeerJ Computer Science* 3 (Jan. 2017), e103. DOI: 10.7717/peerj-cs.103.

- [50] C. R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.
- [51] A. Vučković et al. *pyAGA - Python package for Approximate Graph Automorphisms*. <https://github.com/emdgroup/pyAGA>. Version 0.1.0. Sept. 2022.
- [52] Y.-C. Chen. “A tutorial on kernel density estimation and recent advances”. In: *Biostatistics & Epidemiology* 1.1 (2017), pp. 161–187. DOI: 10.1080/24709360.2017.1396742.