Master's Paper of the Committer on Computational and Applied Mathematics,
The University of Chicago

Schur Net: effectively exploiting local structure for equivariance in higher order graph neural networks

Qingqi Zhang

Advisor(s): Risi Kondor

Approved _____

Date _____

Sep-13, 2024

Abstract

In graph representation learning, message passing neural networks are the most popular architecture due to its accuracy, efficiency and scalability. Recent works have shown that extending the message passing paradigm to subgraphs communicating with other subgraphs, especially via *higher order* messages, can boost the expressivity of such architecture. However, most of previous works constrained the message passing to obey equivariance w.r.t S_m, the symmetry group for local subgraph, which is too restrictive. We resolve this problem by observing that equivariance is only needed w.r.t. the automorphism group of subgraph, and then devise a simple algorithm which finds possible equivariant maps directly from eigenvalue decomposition of graph Laplacian, bypassing the complicated steps of finding automorphism group and decomposition of its permutation representation to *irreps*. This allows effective and efficient construction of equivariant maps on any of the user-defined subgraph template. Empirically, we show that this approach is flexible, extends the previous equivariant maps w.r.t. S_m with minimum extra computational costs and boosts the performance of GNNs on chemical datasets.

Contents

Acknowledgement

Firstly, I would like to express my sincere thanks to my collaborator Richard Xu. We worked together on the experiments of the project and had insightful discussions about ideas, implementations, and the theory. His perseverance and calmness even in hard and urgent times (like before the conference deadline), have tremendously helped us to carry the project out.

I'd like to express my heartfelt gratitude to my advisor, Prof. Risi Kondor, for his patience and hard work on the backend–developing a whole C++-Cuda based software to provide a standard interface for the implementation. Also, Prof. Kondor brought the idea of *Schur* Net up and formulated the theory and provided insightful suggestions on the implementations and experiments. Moreover, his enthusiasm for high-impact academic research has always been inspiring me to go forward.

I'd also like to thank Andrew Hands, for his generous help and valuable advice for our projects, which guided us through the initial and middle stages of experiments.

I'm also profoundly grateful for the many wonderful individuals who have guided me and accompanied me throughout my master's journey in CAM. To name a few, Prof. Eric Baer provided me with a lot of suggestions on choosing courses; Prof. Guillaume Bal, Prof. Mary Silber, Prof. Chao Gao, Prof. Lek-Heng Lim, Prof. Zhiyuan Li, Prof. Gregory Lawler, etc have provided me great lectures and insightful discussions that brings me to the world of applied mathematics. Sida Li, Bill Wang, Jiaheng Chen, Su Hyeong Lee, Yufei Fan, Tracy Wu, Yuxi He, Tianlong Huang, etc studied together with me and accompanied me along my master's journey. I get tremendous help and inspiration from those great people. There are many more to think about, but I have to pause.

Last but not least, I am profoundly in debt to my family, who gave me the firmness and most generous support throughout my study journey, even sometimes at the cost of sacrificing themselves. Without them, I can't come to UChicago, and I will be nowhere near accomplishing my master's degree. They're the ones who create opportunities for me to pursue my lifelong goal.

1. Introduction

Graph neural networks (GNNs) have been the state-of-the-art algorithm in tasks on graph data, e.g. molecule property prediction, network analysis, protein classification [1, 2, 3]. Among the GNNs, message passing neural networks (MPNNs) are the most popular type [2, 4], due to their easy to-implementation and scalability. However, a series of both empirical [5, 6] and theoretical results [7, 8] have shown that it's hard for MPNNs to capture long-term interactions or functional groups such as cycles, and their expressive power are upper bounded by 1-WL test [7]. This is because of the fundamental limitation of MPNNs, that individual vertices send messages only to their neighbor, making MPNNs only aware of the tree-structured neighborhood. This problem is especially acute in domains such as chemistry, where the presence or absence of specific local structural units (functional groups) directly influences the behavior and properties of the molecules.

Recent papers proposed to address this issue by extending the message passing paradigm to also allow for message passing between vertices and edges [9] or between subgraphs, treated as a single unit [10, 11, 12]. However, if the subgraphs are still represented by a scalar, the added expressive power of these networks is limited.

Subsequently, a class of architecture called *higher-order* GNNs emerges [13, 14, 15], which treats subgraphs or tuples of vertices as a whole, and represents them by higher-order tensors, e.g. $T \in R^{m\times\cdots\times m}$. This enables message passing in a more detailed manner. For example, if two benzene rings (6-cycle of carbon atoms) in a molecule intersected in one edge, and they're both represented by first-order tensor $T \in R^{6*c}$, where each row corresponds to one carbon atom and c is the feature dimension, the message can be sent directly between the two intersected nodes, while other atoms could be treated differently. This allows GNNs to learn better the local structures (functional groups) and how they interact with each other.

As detailed in the related work section, there are two lines of research of developing *higher-order* GNNs, one based on k-Weisfeiler Lehman test [16, 17], the other concerning about selecting different domain-specific template subgraphs to be represented by higher-order tensors [15]. Both stem from the seminal work on defining the most general linear equivariant map w.r.t S_n (the symmetry group) between k-th order tensors [13]. However, we notice a fundamental limitation of this characterization in the context of graph representation learning, i.e., **the maps found by [13] are equivariant to the whole symmetry group S_n for any such tensors, without even the need of graph topology.** The addition of graph topology as side information should make the equivariance condition less restrictive and enable the finding of more equivariant maps.

Indeed, the authors of Autobahn architecture [18] already observed that the transformation of (higher-order) representation of local subgraphs only needs to be equivariant to the automorphism group of the subgraph (instead of full symmetry group). While this observation gives the most generally possible equivariant maps, it's hard to implement in practice: one needs to find the automorphism group and

compute the irreducible representations (irreps) which often needs to be done by hand and one also needs to give a canonical ordering of the subgraph, which makes it hard to incorporate a diverse template of subgraphs. Indeed, in the original paper, the authors only use paths of length three to six and cycles of size five and six.

In our paper, we also build on the perspective of leveraging local topology information to extend the space of equivariant maps, but we borrow ideas from spectral graph theory for constructing such maps for any subgraph's automorphism group, resulting in a simple and unified algorithm that indeed only involves one EVD to every subgraph's Laplacian matrix. We first lay the background knowledge about group representation theory in section 2 and background about GNN and equivariance in section 3, then we discuss a general group theoretical approach to find the space of equivariant maps in section 4. Built on this approach, we introduce *Schur* layer to construct equivariant maps w.r.t. automorphism group by EVD of graph Laplacian in section 5. Then section 6 discusses the related works and section 7 gives empirical results for *Schur* layer. Finally, a conclusion and limitations are discussed in section 8.

2. Preliminaries: group theory

2.1 Linear representation of a group

This section briefly introduces the group representation theory that will be used in this paper, most of the materials as well as a more extensive treatment can be found in [19, 20].

Definition 1. Let G be a finite group[1], V be a vector space over the field $\mathbf{C}$, and $\mathbf{GL}(V)$ be the space of invertible linear map of V onto itself. A *linear representation* of G in V is a homomorphism ρ from the group G into the group $\mathbf{GL}(V)$. In other words, ρ associate each element $s \in G$ with $\rho(s) \in \mathbf{GL}(V)$ such that:

$$\rho(st) = \rho(s) \cdot \rho(t) \quad \text{for } s, t \in G$$

Observe that:

$$\rho(1) = 1, \quad \rho(s^{-1}) = \rho(s)^{-1}$$

And in the paper, we use $\rho(s)$ or ρ_s interchangeably. When we fix a basis (e_i) in V, we can associate $\rho(s)$ with its matrix representation R_s, with both representing the same representation of G on V. Sometimes, like in the study of character, we use R_s more often than $\rho(s)$.

Definition 2. Let ρ, ρ' be two representations of the same group G in vector spaces V and V'. We say they are *isomorphic* if there exists a linear isomorphism $\tau : V \to V'$ which "transforms" ρ into ρ', that is:

$$\tau \circ \rho(s) = \rho'(s) \circ \tau \quad \text{for all } s \in G$$

1. We only consider finite groups in this thesis

In matrix form, assume ρ_s, ρ'_s associate with R_s, R'_s, this means there exists an invertible matrix T such that:

$$T \cdot R_s = R'_s \cdot T, \quad \text{for all } s \in G$$

So R_s, R'_s for all $s \in G$ is similar by the same transition matrix T.

This essentially says the action of ρ and ρ' on V and V' respectively are the same once we identify the correspondence between elements by map τ.

Definition 3. Let $\rho : G \to \mathbf{GL}(V)$ be a linear representation and W be a vector subspace of V. Suppose W is *stable* under the action of G, i.e., for any $x \in W, s \in G$, we have $\rho_s(x) \in W$. Then the restriction ρ_s^W of ρ_s to W is a representation of G on W. This is said to be a *subrepresentation* of V.

Basic results show that ρ can be decomposed to subrepresentations completely, i.e., until each subrepresentation can't be decomposed, this is called *irreducible representation*.

Definition 4. Let $\rho : G \to \mathbf{GL}(V)$ be a linear representation of G. We say that it is *irreducible* if V has no stable subspace other than the trivial ones (0 and V). ρ is called *irreducible representation* or *irrep* of G.

Theorem 1 (Complete reducibility)**.** *Every representation is a direct sum of irreducible representations. Concretely, Let $\rho : G \to \boldsymbol{GL}(V)$ be a linear representation, V can be decomposed into $V = W_1 \oplus \cdots \oplus W_k$, where the direct sum means both the direct sum of vector space and sum of group actions, i.e., let $\rho^1, \ldots, \rho^k$ the corresponding* irreps*, $x \in V = x_1 + \cdots + x_k$ where $x_i \in W_i$, then $\rho_s(x) = \rho_s^1(x_1) + \cdots + \rho_s^k(x_k)$.*

Remark 1. In matrix form, suppose under basis $\{e_i\}_{i=1\ldots n}$, ρ_s correspond to matrix R_s, choose another basis $\{v_i\}_{i=1\ldots n}$ by combining basis (v_j^i) of W_i, let the basis transform matrix be M, i.e., $(e_1, \ldots, e_n) = (v_1, \ldots, v_n)M$, and suppose under basis (v_j^i), ρ^i correspond to matrix R_s^i, then

$$R_s = M \begin{pmatrix} R_s^1 & 0 & \cdots & 0 \\ 0 & R_s^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R_s^k \end{pmatrix} M^T, \quad \text{for all } s \in G$$

The basis transform matrix M simultaneously block diagonalizes R_s for all $s \in G$. This is particularly useful in the design of equivariant maps later.

Finally, we need to the concept of *tensor product* of two representations.

Definition 5. The *tensor product* of two vector space V_1, V_2 is denoted by $V_1 \otimes V_2$, characterized by a map $(x_1, x_2) \mapsto x_1 \cdot x_2$ of $V_1 \times V_2 \to V_1 \otimes V_2$. The map satisfies the two following conditions:

(i) $x_1 \cdot x_2$ is linear in each of the variables x_1 and x_2.

(ii) If (e_{i_1}) is a basis of V_1 and (e_{i_2}) is a basis of V_2, the family of products $e_{i_1} \cdot e_{i_2}$ is a basis of $V_1 \otimes V_2$ (So $\dim(V_1 \otimes V_2) = \dim(V_1) \cdot \dim(V_2)$).

Note that $V_1 \otimes V_2$ is spanned by $e_{i_1} \cdot e_{i_2}$, but contains elements that are not just a "product" of the form $x_1 \cdot x_2, x_1 \in V_1, x_2 \in V_2$, but are a sum of those product, e.g., $x_1 \cdot x_2 + y_1 \cdot y_2 + \cdots$. That's why the dimension of $V_1 \otimes V_2$ is much larger than the dimension of $V_1 \times V_2$, where the latter is $\dim(V_1) + \dim(V_2)$.

Definition 6. Let $\rho^1 : G \to \mathbf{GL}(V_1)$ and $\rho^2 : G \to \mathbf{GL}(V_2)$ be two linear representations of G, the *tensor product* of ρ_1 and ρ_2 are a representation $\rho : G \to \mathbf{GL}(V_1 \otimes V_2)$ defined by:

$$\rho_s = \rho_s^1 \otimes \rho_s^2$$

i.e.,

$$\rho_s(x_1 \cdot x_2) = \rho_s^1(x_1) \cdot \rho_s^2(x_2) \quad \text{for } x_1 \in V_1, x_2 \in V_2$$

In matrix form, let (e_{i_1}) be a basis for V_1 and $r_{i_1 j_1}(s)$ be the matrix of ρ_s^1 w.r.t this basis, and define (e_{i_2}) and $r_{i_2 j_2}(s)$ the same way. We have:

$$\rho_s^1(e_{j_1}) = \sum_{i_1} r_{i_1 j_1}(s) \cdot e_{i_1}, \quad \rho_s^2(e_{j_2}) = \sum_{i_2} r_{i_2 j_2}(s) \cdot e_{i_2}$$

so:

$$\begin{aligned} \rho_s(e_{j_1} \cdot e_{j_2}) &= \rho_s^1(e_{j_1}) \cdot \rho_s^2(e_{j_2}) \\ &= \sum_{i_1, i_2} r_{i_1 j_1}(s) \cdot r_{i_2 j_2}(s) \cdot e_{i_1} \cdot e_{i_2} \end{aligned}$$

Thus the matrix of ρ_s is $(r_{i_1 j_1}(s) \cdot r_{i_2 j_2}(s))_{i_1 i_2, j_1 j_2}$; it is just the *tensor product* or *Kronecker product* of the matrices $r_{i_1 j_1}(s)$ and $r_{i_2 j_2}(s)$

2.2 Character theory

In the following, we introduce character theory, which is a very convenient tool to study representation.

Definition 7. Let $\rho : G \to \mathbf{GL}(V)$ be a linear representation of G. The *character* of ρ, denote by $\chi_\rho : G \to \mathcal{C}$ is the sum of eigenvalues of ρ_s:

$$\chi_\rho(s) = \mathrm{Tr}(\rho_s) = \sum_i \lambda_i(s) = \sum_i (R_s)_{ii}$$

where $\mathrm{Tr}(\cdot)$ is the trace operator of a matrix or linear map.

Proposition 1. *If χ is the character of a representation ρ of degree n (degree is the dimension of the vector space V), we have:*

$$\begin{aligned} &(i) \quad \chi(1) = n, \\ &(ii) \quad \chi(s^{-1}) = \chi(s)^* \quad \text{for } s \in G, \\ &(iii) \quad \chi(tst^{-1}) = \chi(s) \quad \text{for } s, t \in G. \end{aligned}$$

where z^* is the conjugate of complex number z.

The proof can be found in Chapter 2.1 of [19].

Proposition 2. *Let $\rho^1 : G \to \boldsymbol{GL}(V_1)$ and $\rho^2 : G \to \boldsymbol{GL}(V_2)$ be two linear representations of G, and let χ_1 and χ_2 be their characters. Then:*

(i) The character χ of the direct sum representation $V_1 \oplus V_2$ is equal to $\chi_1 + \chi_2$.

(ii) The character ψ of the tensor product representation $V_1 \otimes V_2$ is equal to $\chi_1 \cdot \chi_2$.

Proof. Let us be given ρ^1 and ρ^2 in matrix form: R^1_s, R^2_s. The representation $V_1 \oplus V_2$ is then given by

$$R_s = \begin{pmatrix} R^1_s & 0 \\ 0 & R^2_s \end{pmatrix}$$

whence $\mathrm{Tr}(R_s) = \mathrm{Tr}(R^1_s) + \mathrm{Tr}(R^2_s)$, that is $\chi(s) = \chi_1(s) + \chi_2(s)$.
We proceed likewise for (ii): using the matrix of $\rho^1 \otimes \rho^2$, we have

$$\chi_1(s) = \sum_{i_1} r_{i_1 i_1}(s), \quad \chi_2(s) = \sum_{i_2} r_{i_2 i_2}(s),$$

$$\psi(s) = \sum_{i_1, i_2} r_{i_1 i_1}(s) r_{i_2 i_2}(s) = \chi_1(s) \cdot \chi_2(s).$$

□

Lemma 1 (Schur's lemma)**.** *Let $\rho^1 : G \to \boldsymbol{GL}(V_1)$ and $\rho^2 : G \to \boldsymbol{GL}(V_2)$ be two* irreducible representations *of G, and let f be a linear mapping of V_1 into V_2 such that $\rho^2_s \circ f = f \circ \rho^1_s$ for all $s \in G$. Then:*

(1) If ρ^1 and ρ^2 are not isomorphic, then $f = 0$.

(2) If $V_1 = V_2$ and $\rho^1 = \rho^2$, f is a homothety *(i.e., a scalar multiple of identity).*

Remark 2. In case (2) where ρ^1 and ρ^2 are isomorphic (but not identical) by a map $\tau : V_1 \to V_2$ satisfying $\tau \circ \rho^1_s = \rho^2_s \circ \tau$, f became scalar times this isomorphism map: $f = \lambda\tau, \lambda \in \mathcal{C}$

The proof can be found in Chapter 2.2 of [19]. It's directly built on the defining property of *irrep* that the only stable subspaces of V_1 (or V_2) are just $0, V_1$ (or $0, V_2$). $\rho_s^2 \circ f = f \circ \rho_s^1$ is called *equivariance* property of mapping f, which is the focus of our paper. The importance of Schur's lemma is it characterizes all the equivariant maps between two *irreps*, it either just zero when they're not isomorphic, or the isomorphism map times a scalar. We'll see later that constructing equivariant maps in the context of tensors on a graph is just decomposing the permutation representation of the symmetry group or the automorphism group into *irreps* and utilizing this result.

We can define an inner product of complex-valued functions on G by:

$$(\phi|\psi) = 1/g \sum_{t \in G} \phi(t)\psi(t)^*, g \text{ being the order of } G.$$

Characters are functions on G (indeed class functions where each conjugate class has the same value). The following result shows that *irreducible* characters form an orthonormal system under this inner product.

Fact 1. *(i) If χ is the character of an irreducible representation, we have $(\chi|\chi) = 1$ (i.e., χ is "of norm 1").*

(ii) If χ and χ' are the characters of two nonisomorphic irreducible representations, we have $(\chi|\chi') = 0$ (i.e., χ and χ' are orthogonal).

Fact 2. *Let V be a linear representation of G, with character ϕ, and suppose V decomposes into a direct sum of irreducible representations:*

$$V = W_1 \oplus \cdots \oplus W_k.$$

Then, if W is an irreducible representation with character χ, the number of W_i isomorphic to W is equal to the scalar product $(\phi|\chi)$.

This shows the number of *irrep* W appears in the decomposition of V (alas multiplicity of W) only depend on the type of representation on W and V, but not depend on the chosen decomposition.

Fact 3. *Two representations are isomorphic if and only if their characters are the same.*

Proof. On one hand, two isomorphic representations has matrices that are similar, i.e., $R_s^1 = TR_s^2T^{-1}$ for all $s \in G$, thus $\mathrm{Tr}(R_s^1) = \mathrm{Tr}(R_s^2)$. On the other hand, if two representations have the same character, by the previous result, they decompose into the same number of *irrep* of each kind, so they're isomorphic. □

Fact 4. *Let G be a group of order g and $W^1, \ldots, W^h$ are* irreps *with degree $n_1, \ldots, n_h$, then $g = \sum_i n_i^2$*

Definition 8 (Canonical decomposition)**.** Let $\rho : G \to \mathbf{GL}(V)$ be a linear representation of G and $W^1, \ldots, W^h$ are *irreps*. And suppose V decompose into direct sum of *irreps* by:

$$V = W_1^1 \oplus \cdots \oplus W_{\kappa_1}^1 \oplus \cdots \oplus W_1^h \oplus \cdots \oplus W_{\kappa_h}^h$$

where $\kappa_1, \ldots, \kappa_h$ are the multiplicities of $W^1, \ldots, W^h$. We group the same type of W_i together into $V_i = W_1^i \oplus \cdots \oplus W_{\kappa_i}^i$, then

$$V = V_1 \oplus \cdots \oplus V_h$$

are called the *canonical* decomposition of V. It has a nice property that it does not depend on the initially chosen decomposition of V into *irreps* and is uniquely determined by V itself.

The proofs of the above-mentioned facts can be found in Chapters 2.3 and 2.4 of [19], and the way to construct canonical decomposition can be found in Chapter 2.6 of the same book.

2.3 Examples

2.3.1 Irreducible representations of dihedral group D_n

The dihedral group D_n is the group of rotations and reflections of the plane that preserve a regular polygon with n vertices. Let r be rotation and s be reflection, we have:

$$r^n = 1, \quad s^2 = 1, \quad rs = sr^{-1}$$

D_n is generated by r and s, and the order of D_n is $2n$. The automorphism group of the n-cycle graph is D_n.

Case 1: n even First there are 4 representations of degree 1, obtained by assigning ± 1 to r and s in all possible ways. They're all irreducible since they're of degree 1. Their characters $\psi_1, \psi_2, \psi_3, \psi_4$ are:

	r^k	sr^k
ψ_1	1	1
ψ_2	1	-1
ψ_3	$(-1)^k$	$(-1)^k$
ψ_4	$(-1)^k$	$(-1)^{k+1}$

Then we consider representations of degree 2. Let $w = e^{2\pi i/n}$ (the unit rotation corresponding to r) and h be an arbitrary integer. Define a representation ρ^h of D_n by setting:

$$\rho^h(r^k) = \begin{pmatrix} w^{hk} & 0 \\ 0 & w^{-hk} \end{pmatrix}, \quad \rho^h(sr^k) = \begin{pmatrix} 0 & w^{-hk} \\ w^{hk} & 0 \end{pmatrix}.$$

One can verify by calculation that this is indeed a representation and $\rho^h, \rho^{(h+n)}$ are the same, $\rho^h, \rho^{(n-h)}$ are isomorphic. Hence we may assume $0 \leq h \leq n/2$. Let χ^h be its character. When $h = 0$ and $h = n/2$, ρ^h is reducible: $\chi^0 = \psi_1 + \psi_2$ and $\chi^{n/2} = \psi_3 + \psi_4$. For $0 < h < n/2$, the representation ρ^h is irreducible: since $w^h \neq w^{-h}$, the only lines stable under $\rho^h(r)$ are the coordinate axes, and these are not stable under $\rho^h(s)$. We can calculate its character directly:

$$\chi_h(r^k) = w^{hk} + w^{-hk} = 2\cos(\frac{2\pi hk}{n})$$
$$\chi_h(sr^k) = 0$$

Using fact 4, we can see that the irreducible representations of degree 1 and 2 constructed above are all the *irreps* of D_n since $4 \times 1 + ((n/2 - 1) \times 4 = 2n$.

Case 2: n odd There are only 2 representations of degree 1, since we can only assign 1 to r (otherwise $r^n = -1 \neq 1$). Their characters ψ_1, ψ_2 are:

	r^k	sr^k
ψ_1	1	1
ψ_2	1	-1

For *irrep* of degree 2, we utilize the ρ^h defined in the n even case and observe that when $0 < h < n/2$ (actually $0 < h \leq (n-1)/2$), ρ^h are irreducible and pairwise nonisomorphic. Using fact 4 again, we see that these representations are the only *irreps* of D_n when n is odd, since $2 \times 1 + \frac{1}{2}(n-1) \times 4 = 2n$.

Example 1. (1) The group D_6 has 4 *irreducible* representations of degree 1, with characters $\psi_1, \psi_2, \psi_3, \psi_4$ and 2 *irreducible* representations of degree 2, with characters χ_1 and χ_2.

(2) The group D_5 has 2 *irreducible* representations of degree 1, with characters ψ_1, ψ_2 and 2 *irreducible* representations of degree 2, with characters χ_1 and χ_2.

2.3.2 Irreducible representation of S_n

The symmetry group S_n consists of all bijections between a set of n elements. A graph with n vertices is invariant under arbitrary relabeling of the vertices, thus endowed with a symmetry of S_n. That's why we're interested in finding *irreps* of S_n.

In general, the *irreps* of S_n can be quite complicated and is indexed by the integer partition $\lambda \vdash n$, where $\lambda = (\lambda_1, \ldots, \lambda_l), \sum_i \lambda_i = n$. For example, $n = 8$, $\lambda_1 = (4, 3, 1), \lambda_2 = (3, 2, 1, 1, 1)$ are integer partitions of n. This comes from the fact that the number of *irreps* equal to the number of conjugacy classes of the group.

For example, the partition $\lambda = (n)$ corresponds to the *trivial* representation and $\lambda = (n-1, 1)$ corresponds to the representation on $\mathbb{1}^\perp \subset R^n$ (the orthogonal complement of all one's vector in R^n) by permutating the coordinates. In fact, the

permutation representation of S_n on R^n (permutating coordinates of the vectors) decomposes to the direct sum of those two *irreps*. The third *irrep* that is easy to identify is the one indexed by $\lambda = (1, 1, \ldots, 1)$, which gives a 1-dim representation mapping $\sigma \in S_n$ to its sign (1 for even permutation -1 for odd).

Other *irreps* are mostly complicated to describe, and we refer the reader to [20] for the construction of all *irreps* (also known as Young's natural representations) of S_n and their properties.

3. Graph neural networks and equivariance w.r.t S_m

In this section, we'll present the framework of graph neural networks (GNNs) and the concept of equivariance.

3.1 Graph neural networks

In graph representation learning, the goal is to learn a representation for each graph or each node. Formally, given a graph $G = (V, E, X)$, where V is the node set with $|V| = n$, E is the edge set and $X \in R^{n*d}$ is node features (edge features could also be added but omitted here), a function $f : G \to R^{d'}$ (or $f : G \to R^{n \times d'}$ gives a representation of the graph (or nodes), which is then used for downstream task like graph (or node) property prediction.

A GNN learns the map f by stacking layers of neural networks on G, and among them, the most prominent type is message passing neural networks (MPNNs). In the traditional MPNNs, a node representation $h_v^{(l)} \in \mathbb{R}^d$ is maintained in each layer for all $v \in V$ and updated by aggregating messages from neighboring nodes:

$$a_v^{(l)} = \text{AGGREGATE}^{(l)} \left(\left\{ h_u^{(l-1)} : u \in \mathcal{N}(v) \right\} \right),$$

$$h_v^{(l)} = \text{COMBINE}^{(l)} \left(h_v^{(l-1)}, a_v^{(l)} \right),$$

where AGGREGATE$(\cdot)$ is an order-invariant function like summation or maximum and COMBINE$(\cdot)$ is usually a linear transform followed by non-linear activation. For example, in [1],

$$h_v^{(l)} = \eta \left(\left(\sum_u \left(\left\{ h_u^{(l-1)} : u \in \mathcal{N}(v) \right\} \right) + h_v^{(l-1)} \right) W^{(l)} \right) \tag{1}$$

$$H^{(l)} = \eta(AH^{(l-1)}W^{(l)}) \tag{2}$$

where $\eta(\cdot)$ is the sigmoid function, A is the adjacency matrix of G, $H^{(l-1)}$ is a row stack of node feature $h_u^{(l-1)}$, and $W^{(l)}$ are learnable weights in the l-th layer. See figure 1 for a slightly more compicated example.

Later works extend the message passing framework to *higher-order* message passing, incorporating representations not just for one node and is a scalar, but for a tuple

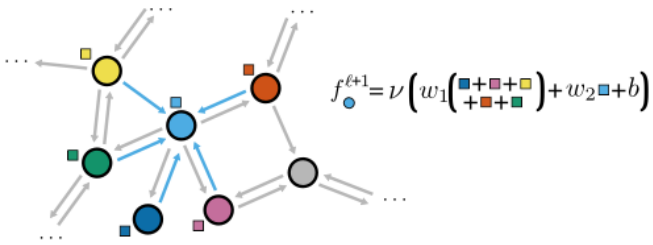


Figure 1: Illustration of message passing when the AGGREGATE$(\cdot)$ is just summation and COMBINE$(\cdot)$ is a linear tranform followed by a non-linearity. Feature updated of the central blue node is shown.

of nodes and is a k-dimensional tensor, in for example [13, 14, 15, 21, 22]. While [13, 14] didn't incorporate message passing operation, the other four works combine *higher-order* representation and local message passing and gain improvements in performance (such as prediction accuracy). In our paper, we view the message passing as a smart and natural way to leverage the graph struture, thus being so popular since the development of GNNs.

3.2 Invariance and Equivariance w.r.t S_m

A fundamental requirement for GNNs and any other graph representation learning methods are the learned representation should be invariant (for graph) or equivariant (for node representation) w.r.t S_m. That is because, the numbering of nodes are usually arbitrarily chosen, for any permutation of the node numbering, the underlying graph is the same, thus the resulting representation should either be the same or change corresponding to the permutation. Formally, the action of a permutation $\sigma \in S_m$ on a k-th order tensor $T \in R^{m \times \cdots \times m}$ is a linear representation of S_m on R^{m^k}, and the linear map corresponding to $\sigma : T \mapsto T^\sigma = \sigma \circ T$ is given by:

$$(T^\sigma)_{i_1,i_2,\ldots,i_k} = T_{\sigma^{-1}(i_1),\sigma^{-1}(i_2),\ldots,\sigma^{-1}(i_k)} \tag{3}$$

For example, when $k = 2$ and $T = A$ is the adjacency matrix, we have:

$$(A^\sigma)_{i,j} = A_{\sigma^{-1}(i),\sigma^{-1}(j)}$$

where $A^\sigma = \sigma \circ A$. From now on, we'll use the terms "linear representation" and "group action" on space R^{m^k} interchangeably.

For a transform $\phi : T^{\text{in}} \to T^{\text{out}}$, and denote $T^\sigma = \sigma \circ T$, the *invariance* w.r.t S_m is:

$$\phi(T^{\text{in}}) = \phi(\sigma \circ T^{\text{in}}) \quad \text{for all } \sigma \in S_m \tag{4}$$

the *equivariance* is:

$$\phi(\sigma \circ T^{\text{in}}) = \sigma \circ \phi(T^{\text{in}}) = \sigma \circ T^{\text{out}} \quad \text{for all } \sigma \in S_m \tag{5}$$

One can also replace S_m with any subgroup of S_m like D_m to define invariance or equivariance w.r.t. that group. The key concept of *equivariance* is the output is permuted as the manner of the input's permutation, or the map ϕ and action σ is commutable, i.e., $\phi \circ \sigma = \sigma \circ \phi$ (recall this from the schur lemma 1). Also note that invariance is a special case of equivariance when T^{out} is 0-th order tensor (just a scalar) with the trivial group action on it. And in reality, the GNNs mostly build equivariant maps first followed by a order-invariant *readout* function to get an invariant map. Therefore, start from now we only need to discuss *equivariance* and equivariant maps. We'll start by the discussion of equivariance w.r.t. S_m and then move to equivariance w.r.t. automorphism group of the graph. For future references, we give the definition of the automorphism group here:

Definition 9 (Automorphism group of a graph)**.** Let $G = (V, E)$ with $|V| = m$ and adjacency matrix $A \in R^{m*m}$, the *automorphism* group of G is a subgroup of S_m with all $\sigma \in S_m$ such that:

$$A^\sigma = A$$

That is, after renumbering the vertices with σ, the edge set is still the same, i.e., $E^\sigma = \{(\sigma(e_1^i), \sigma(e_2^i))\} = E$.

4. The space of equivariant maps

Suppose $\phi : T^{\text{in}} \mapsto T^{\text{out}}$ is equivariant w.r.t. S_m (or some subgroup of S_m), where T^{in} and T^{out} are k_{in} and k_{out} dimensional tensors respectively, we'd like to characterize all such equivariant maps to use in GNNs. To start with, we derive the matrix form of the permutation actions.

Definition 10. Let $\sigma \in S_m$ act on a k-th order tensor T by 3, defining the permutation matrix corresponding to σ as the orthongonal matrix:

$$P_\sigma \in \mathcal{R}^{m \times m} \qquad\qquad [P_\sigma]_{i,j} = \begin{cases} 1 & \text{if } \sigma(j) = i, \\ 0 & \text{otherwise,} \end{cases}$$

Then the matrix form of $\sigma \circ T$ is:

$$\text{vec}(\sigma \circ T) = P_\sigma^{(k)} \text{vec}(T) \qquad\qquad P_\sigma^{(k)} = \underbrace{P_\sigma \otimes P_\sigma \otimes \ldots \otimes P_\sigma}_{k \text{ times}}, \tag{6}$$

where $\text{vec}(\cdot)$ vectorize the tensor into R^{m^k} vector and $\otimes$ is the tensor product or Kronecker product of matrices. In particular, $P_\sigma^{(k)}$ is $n^k \times n^k$ dimension matrix.

For now we'll assume $k_{\text{in}} = k_{\text{out}}$ and only consider equivariance w.r.t. S_m. Equivariance w.r.t. subgroups of S_m, e.g., the automorphism group of the graph will be discussed in the following section. The case $k_{\text{in}} \neq k_{\text{out}}$ can be easily generalized, just

by having a different decomposition of action on T^{in} and T^{out}. Since the action of σ on T is a linear representation of S_m on the vector space R^{m^k}, by the complete reducibility 1, we have:

Proposition 3. *Let $\rho_\lambda, \lambda \vdash m$ be the* irreps *of S_m, $U = R^{m^k}$, $\sigma \in S_m$ act on U in the manner of equation 3 and is given in matrix form by equation 6, and suppose this action contains* irreps $\rho_{\lambda_1}, \ldots, \rho_{\lambda_p}$ *with multiplicities $\kappa_1, \ldots, \kappa_p$, we have:*

$$U = U_1 \oplus U_2 \oplus \cdots \oplus U_p, \quad U_i = \bigoplus_j^{\kappa_i} V_i^j$$

where the first part gives the canonical decomposition of U (cf 8) and the second part further decompose each U_i into irreducible subspaces, where V_i^j correspond to ρ_{λ_i}. In matrix form,

$$P_\sigma^{(k)} = M \begin{pmatrix} \rho_{\lambda_1}(\sigma) & 0 & \cdots & 0 \\ 0 & \rho_{\lambda_2}(\sigma) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \rho_{\lambda_p}(\sigma) \end{pmatrix} M^T, \quad \textit{for all } \sigma \in S_m$$

where M is the orthogonal transformation for basis and we abuse $\rho_{\lambda_i}(\sigma)$ to denote also the matrix of i-th irrep *in the decomposition.*

If we take a closer look at M, we can find that since $P_\sigma^{(k)}$ is under the standard basis of R^{m^k}, thus M is just the orthonormal basis of each U_i (thus each V_i^j) combined together, we denote $M = (M_1, \ldots, M_p)$ with M_i a $m^k \times (d_{\lambda_i} * \kappa_i)$ dimensional matrix, where d_{λ_i} is the degree of ρ_{λ_i}. Therefore, $P_\sigma^{(k)} T$ becomes:

$$P_\sigma^{(k)} T = [M_1, M_2, \ldots, M_p] \underbrace{\begin{pmatrix} \rho_{\lambda_1}(\sigma) & 0 & \cdots & 0 \\ 0 & \rho_{\lambda_2}(\sigma) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \rho_{\lambda_p}(\sigma) \end{pmatrix} \underbrace{\begin{pmatrix} M_1^T \\ M_2^T \\ \vdots \\ M_p^T \end{pmatrix}}_{(1)} T}_{2} \tag{7}$$

The part (1) is a generalized Fourier transform of T to its Fourier components (coordinates under the orthonormal basis) $\hat{T} = \begin{pmatrix} M_1^T T \\ M_2^T T \\ \vdots \\ M_p^T T \end{pmatrix} \triangleq \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_p \end{pmatrix}$, and the part (2) is the *irreps* $\rho_{\lambda_1}, \ldots, \rho_{\lambda_p}$ act independently on $\hat{T}$ with each component $B_i \mapsto$

$\begin{pmatrix} \rho_{\lambda_i}(\sigma) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \rho_{\lambda_i}(\sigma) \end{pmatrix} B_i$. Note that there're κ_i multiple of ρ_{λ_i} act the same on components of B_i correspond to V_i^j, with a slight abuse of notation, we can think $B_i \in R^{d_{\lambda_i} \times \kappa_i}$ (instead of $R^{(d_{\lambda_i} * \kappa_i) \times 1}$) and write this map as $\rho_{\lambda_i}(\sigma) B_i$, the fact that the map by *irrep* ρ_{λ_i} act independently on each column of B_i allow us to identify directly a set of equivariant maps by multiplying B_i with matrix $W_i \in R^{\kappa_i \times \kappa_i}$ to the right, and using associativity of matrix multiplication: $\rho_{\lambda_i}(\sigma)(B_i W_i) = (\rho_{\lambda_i}(\sigma) B_i) W_i$. This gives in total of $\sum_i \kappa_i^2$ independent equivariant maps, and we'll see later this is indeed the whole space of possible equivariant maps. The last part of the above equation maps Fourier components to their original space. In short, we can write:

$$P_\sigma^{(k)} T = \sum_i M_i \rho_{\lambda_i}(\sigma) \underbrace{M_i^T T}_{B_i} \tag{8}$$

with the abuse of notation to rearrange elements in B_i mentioned above.

4.1 Equivariance criteria

For now, we'd like to derive the necessary and sufficient condition for general equivariance w.r.t. any group action, then we'll return to the specific case of S_m.

Theorem 2. *Let G be a finite group acting on a vector space U by the linear action $\{g : U \to U\}_{g \in G}$ (In other words, $\rho : G \to \boldsymbol{GL}(U)$ is a linear representation of G and write in short $g = \rho(g)$), assume that we have a decomposition of U into **stable** subspaces:*

$$U = U_1 \oplus \ldots \oplus U_p$$

Let $\phi : U \to U$ be a linear map that is a homothety on each U_i, i.e., $\phi(w) = \alpha_i w$ for some fixed scalar α_i and for $W \in U_i$. Then ϕ is equivariant to the action of G on U in the sense that $\phi(g(u)) = g(\phi(u))$ for any $u \in U$ and any $g \in G$.

Proof. Suppose $u_i \in U_i$ and for any $g \in G$, then $g(u_i) \in U_i$, so $\phi(g(u_i)) = \alpha_i g(u_i) = g(\alpha_i u_i) = g(\phi(u))$.

By linearity, the equation holds for any $u \in U$ by the decomposition $u = u_1 + \cdots + u_p$, where $u_i \in U_i$. □

Theorem 3 (Necessary and sufficient condition for equivariant map)**.** *Let G be a finite group acting on a vector space U by the linear action $\{g : U \to U\}_{g \in G}$ and assume the action can be decomposed into* irreps *$\rho_1, \ldots, \rho_p$ with multiplicities $\kappa_1, \ldots, \kappa_p$ and degree d_i:*

$$U = U_1 \oplus U_2 \oplus \cdots \oplus U_p, \quad U_i = \bigoplus_{j=1}^{\kappa_i} V_i^j$$

Then $\phi : U \to U$ is an equivariant map w.r.t. this group action if and only if ϕ is of the form:

$$\phi(v) = \sum_{j'} \alpha^i_{j,j'} \tau^i_{j \to j'}(v) \quad \textit{for } v \in V^i_j \tag{9}$$

for some fixed set of coefficient $\{\alpha^i_{j,j'} : i \in [1, \dots, p], j, j' \in [1, \dots, \kappa_i]\}$.
In matrix form, suppose the matrix of g is R_g under basis (e_i) and $dim(U) = n$, and it decompose into:

$$R_g = M \begin{pmatrix} \rho_1(g) & 0 & \cdots & 0 \\ 0 & \rho_2(g) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \rho_p(g) \end{pmatrix} M^T \tag{10}$$

$$= \sum_i M_i \rho_i(g) M_i^T \tag{11}$$

Then $\phi : R^n \to R^n$ is equivariant if and only if it is of the form:

$$\phi(T) = M M^T T \begin{pmatrix} W_1 & 0 & \cdots & 0 \\ 0 & W_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & W_p \end{pmatrix} \tag{12}$$

$$= \sum_i M_i \underbrace{M_i^T T}_{B_i} W_i \tag{13}$$

where $T \in R^n$, $W_i \in R^{\kappa_i \times \kappa_i}$ is the fixed coefficients and B_i rearrange to $R^{d_i \times \kappa_i}$ when needed. The coefficients $\alpha^i_{j,j'}$ corresponds to $(W_i)_{j,j'}$. [2]

Proof. **Sufficiency:** firstly, $\alpha^i_{j,j'} \tau^i_{j \to j'}(v)$ is equivariant by definition of isomorphism map. And the equivariance of ϕ follows from the fact that sum of equivariant maps are equivariant.

Necessity: consider ϕ on V^i_j and let $W = Im(\phi|_{V^i_j})$ the image of ϕ on V^i_j. Since $\phi \circ g = g \circ \phi$ and $g(v) \in V^i_j$ for any $v \in V^i_j$, we have $g(\phi(v)) = \phi(g(v)) \in W$, so W is stable under the action. Thus we can decompose W into irreducible spaces $W = W_1 \oplus \dots W_p$. Apply this decomposition to $\phi(v)$ for $v \in V^i_j$, we get:

$$\phi(v) = \phi_1(v) + \dots + \phi_p(v) \quad \text{where } \phi_i(v) \in W_i$$

In other words $\phi_k = Proj_k \circ \phi$ where Proj_k is the projection of W onto W_k (uniquely defined by the direct sum decomposition). Then $\phi_k : V^i_j \to W_k$ and $\phi_k \circ g = g \circ \phi_k$. By Schur's lemma 1, for $\phi_k \neq 0$, we must have W_k isomorphic to V^i_j and

2. Similar discussions could be found in Section 4 of [23], but without a proof for necessity.

$\phi_k(v) = \theta_k \tau_{V_j^i, W_k}(v)$. Since only $V_{j'}^i$ is isomorphic to V_j^i, we have $W_k = V_k^i$ and $\phi_k(v) = \alpha_{j,k}^i \tau_{j,k}(v)$, where $\tau_{j,k}(v)$ is the isomorphic map sending V_j^i to V_k^i. Thus $\phi(v) = \sum_k \phi_k(v) = \sum_k \alpha_{j,k}^i \tau_{j,k}(v)$ for $v \in V_j^i$.

Finally, by linearity and $U = \bigoplus_{i,j} V_j^i$, the only possible equivariant function $\phi : U \to U$ is given by 9.

Connection to matrix form: first note that the isomorphism $\tau_{j \to j'}^i : V_j^i \to V_{j'}^i$ is given by:

$$\tau_{j \to j'}^i(u_{j,l}^i) = u_{j',l}^i$$

where $\{u_{j,l}^i, l = 1, \ldots, d_i\}$ and $\{u_{j',l}^i, l = 1, \ldots, d_i\}$ are orthonormal basis for V_j^i and $V_{j'}^i$ respectively such that the action of g is associated with matrix $\rho_i(g)$. Therefore,

$$\begin{aligned} \phi(u_{j,l}^i) &= \sum_{j'} \alpha_{j,j'}^i \tau_{j \to j'}^i(u_{j,l}^i) \\ &= \sum_{j'} \alpha_{j,j'}^i (u_{j',l}^i) \end{aligned}$$

Thus the matrix of ϕ under basis $(u_{j,l}^i, i = 1, \ldots, p, j = 1, \ldots, \kappa_i, l = 1, \ldots, d_i)$ is

$$\begin{pmatrix} W_1 & & & & & & \\ & \ddots & & & & & \\ & & W_2 & & & & \\ & & & \ddots & & & \\ & & & & W_p & & \\ & & & & & \ddots & \end{pmatrix}$$ (which is the same as $\begin{pmatrix} W_1 & 0 & \cdots & 0 \\ 0 & W_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & W_p \end{pmatrix}$ with B_i rearrange to $R^{d_i \times \kappa_i}$) and M is just the matrix to transform standard basis (e_i) to basis $(u_{j,l}^i)$.

□

Applying the result to the case of S_m, we get the following corollary.

Corollary 1. *Let $U = R^{m^k}$, $\sigma \in S_m$ act on U in the manner of equation 3 and is decomposed as in proposition 3, then the only possible equivariant maps $\phi : U \to U$ are:*

$$\phi(v) = \sum_{j'} \alpha_{j,j'}^i \tau_{j \to j'}^i(v) \quad \text{for } v \in V_j^i \tag{14}$$

with matrix form $\phi : R^n \to R^n$ where $n = dim(U) = m^k$

$$\phi(T) = \sum_i M_i M_i^T T W_i \tag{15}$$

for any $T \in R^n$ and fixed weight $W_i \in R^{\kappa_i \times \kappa_i}$

Proof. Since proposition 3 already gives the decomposition of the action of S_m on U, we can directly apply the previous theorem to get the result. □

Example 2. (i) For first order tensor $T \in R^m$, the action of S_m on R^m decomposes to two *irreps*: type (m) and $(m-1,1)$, so the basis of equivariant maps consists of two maps. Actually, the first is the identity map and the second is $T \mapsto \sum_i T_i$.

(ii) For second order tensor $T \in R^{m^2}$, the action of S_m on R^m decompose to: 2 isomorphic copies of type (m), 3 copies of $(m-1,1)$, 1 copy of $(m-2,2)$ and $(m-2,1,1)$, thus in total 15 equivariant maps in the basis. They're given by the following:

$$\begin{aligned} T^{\text{out}}_{i,j} = w_0 T^{\text{in}}_{i,j} + w_1 T^{\text{in}}_{j,i} + w_2 T^{\text{in}}_{i,*} + w_3 T^{\text{in}}_{*,i} + w_4 T^{\text{in}}_{*,j} + \\ w_5 T^{\text{in}}_{j,*} + w_6 T^{\text{in}}_{*,*} + w_7 \overline{T^{\text{in}}_{*}} + w_8 T^{\text{in}}_{i,i} + w_9 T^{\text{in}}_{j,j} + \\ \delta_{i,j}(w_{10}\overline{T^{\text{in}}_{i}} + w_{11}\overline{T^{\text{in}}_{*}} + w_{12} T^{\text{in}}_{*,*} + w_{13} T^{\text{in}}_{i,*} + w_{14} T^{\text{in}}_{*,i}) \end{aligned}$$

where $T^{\text{in}}_{p,*} = \sum_k T^{\text{in}}_{p,k}, T^{\text{in}}_{*,p} = \sum_k T^{\text{in}}_{k,p}, T^{\text{in}}_{*,*} = \sum_{k,l} T^{\text{in}}_{k,l}, \overline{T^{\text{in}}_{p}} = T^{\text{in}}_{p,p}$ and $\overline{T^{\text{in}}_{*}} = \sum_k T^{\text{in}}_{k,k}$. And $w_0, \ldots, w_{14}$ are coefficients that combine the basis together.

Note that although the theorem gives the most general form of equivariant maps, it's hard to carry it out explicitly. One needs to first find the *irreps* of G and carry the decomposition of group action explicitly (finding orthogonal matrix M), which is very specific to each group G and space U and hard to generalize. For the case of S_m, in the example above, we actually brute forcely enumerate the possible equivariant maps by elementary computation and check the number matches. Another linear algebraic method to get the equivariant maps explicitly could be found in [13].

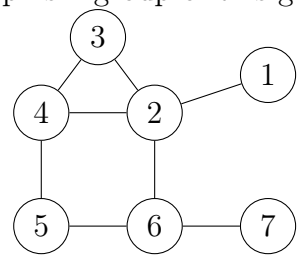
Applying the result for S_m to be used in GNNs, one might ask, where does the graph topology come into play? The maps are exactly the same with any kth-order tensor being input and output, regardless of whether it's a tensor on subgraph or it's representing data on other order-invariant objects like high-order relation on sets. Can we further extend the maps by taking into account the side information edges of the graph give us? The answer is yes and we'll discuss it in the next section.

5. Equivariance w.r.t automorphism group: *Schur* Net

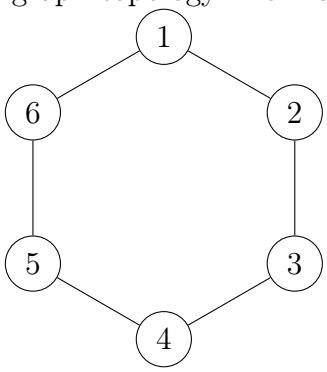
5.1 We only need equivariance w.r.t automorphism group

There are two motivations of *Schur* Net. The first is the observation that though the maps on a graph need to be equivariant or invariant w.r.t. S_m in an end-to-end point of view, the true ambiguity of the node labeling is only the automorphism group of the graph. For example, in the following graph, there's no ambiguity of node labeling: no matter what initial label is given, one can always relabel it by identifying the houselike structure (node 2-6), and label the node adjacent to the roof as 1, the node adjacent

to the bottom as 7. That is because the automorphism group of this graph is trivial, so there exists a canonical ordering of the graph.



The real ambiguity happens when the automorphism group is non-trivial, for example, in the six cycle shown below, there's no way to distinguish it if someone decides to relabel 1 to 2, 2 to 3, ..., 6 to 1. This relabeling is a cyclic rotation which lies in D_6, the automorphism group of six cycle. Therefore, the real challenge the map of a graph needs to face is the invariance/equivariant w.r.t. the automorphism group. This is the minimum constraint of feature maps on a graph and takes into account the graph topology information.



Indeed, [18] proposed to design equivariant maps w.r.t. the automorphism group on chosen subgraphs. They empirically implemented all such equivariant maps for cycles of length five and six and paths of length three to six. However, either using the aforementioned group theoretic approach or crafting the equivariant maps by hand, it's hard for the users to come up with the explicit form of equivariant maps. The fact that this calculation needs to be done for every automorphism group of each subgraph chosen further makes things more complicated and not very scalable. To mimic this, we designed an algorithm that could directly get most of the equivariant maps from the eigenvalue decomposition (EVD) of graph Laplacian and appliable to any subgraphs without any change. Our algorithm is motivated by another aspect of the adjacency matrix.

5.2 Rethinking MPNNs

The simple MPNN proposed by [1] (see equation 2) is not included in the characterization of first-order equivariant maps w.r.t. S_n, yet still being equivariant. In particular, $\phi(H) = AHW$ for $H \in R^n$ and for any $\sigma \in S_n$, we have:

$$\phi(\sigma \circ H) = A^\sigma H^\sigma W = P_\sigma A P_\sigma^T P_\sigma H W = P_\sigma A H W = \sigma \circ \phi(H)$$

The reason for this is the map ϕ itself depends on adjacency matrix A, which permutes with σ. Indeed, the equivariance condition becomes:

$$\phi_{\sigma(A)}(\sigma \circ T^{\text{in}}) = \sigma \circ \phi_A(T^{\text{in}})$$

This enables ϕ to use information about graph topology (filter out the neighboring nodes) and as a large number of papers [2, 7, 24] shows, this indeed helps with finding good graph embeddings. **The key idea here is to provide the map ϕ with some objects of the graph (like adjacency matrix) that transform together with the input when relabeling occurs, thus the relative order being unchanged and providing meaningful information about the graph such as neighborhood information.** One may ask, can we use this clever idea to leverage graph topology information and extend the set of equivariant maps for kth-order tensor? The answer is again yes, and this brings us to use graph Laplacian and spectral theory in the designing of equivariant maps.

5.3 *Schur* Net

Building from the two motivations, we're seeking a way to construct equivariant maps w.r.t. local automorphism group by leveraging adjacency matrix or graph Laplacian. This can be done as simply as an EVD on the graph Laplacian.

Definition 11 (Combinatorial graph Laplacian)**.** Let S be a graph with adjacency matrix A. $L = D - A$ its *combinatorial graph Laplacian*, where D is the diagonal matrix with degree of ith node at (i, i)-th entry. Note that, L transform with permutation $\sigma \in S_m$ as the same way A do:

$$L^\sigma = \sigma \circ L = P_\sigma L P_\sigma^T$$

where P_σ permutation matrix of σ. Also note for $\sigma \in Aut_S$,

$$L^\sigma = D^\sigma - A^\sigma = D - A = L.$$

Lemma 2. *Let S be an undirected graph with m vectices, Aut_S its automorphism group, and $L = D - A$ its* combinatorial graph Laplacian*. Assume L has t distinct eigenvalues $\lambda_1, \ldots, \lambda_t$ and corresponding subspaces $U_1, \ldots, U_t$. Then each U_i is invariant under the first-order action of Aut_S on R^m and $R^m = U_1 \oplus \ldots \oplus U_t$.*

Proof. Since L is real symmetric, the eigenvectors spans R^m, thus $R^m = U_1 \oplus \ldots \oplus U_t$.

Let $\sigma \in Aut_S$, by definition $P_\sigma A P_\sigma^T = A$, and thus $P_\sigma L P_\sigma^T = L$. Furthermore, $\mathbf{v} \in U_i$ if and only if $L\mathbf{v} = \lambda_i \mathbf{v}$. Therefore, for any $\mathbf{v} \in U_i$ and any $\sigma \in Aut_S$,

$$L \underbrace{P_\sigma \mathbf{v}}_{\sigma(\mathbf{v})} = P_\sigma L P_\sigma^T P_\sigma \mathbf{v} = P_\sigma L \mathbf{v} = \lambda_i \underbrace{P_\sigma \mathbf{v}}_{\sigma(\mathbf{v})}$$

showing that $\sigma(\mathbf{v}) \in U_i$ and hence that U_i is an invariant subspace. □

Corollary 2. *Given a multi-index* $\mathbf{i} = (i_1, \ldots, i_k) \in \{1, \ldots, p\}^k$*, define* $U_{\mathbf{i}} = U_{i_1} \otimes \ldots \otimes U_{i_k}$ *with* $U_1, \ldots, U_t$ *the same in the lemma, then* $U_{\mathbf{i}}$ *is an invariant subspace of* R^{m^k} *under kth-order action of* Aut_S *on* R^{m^k} *(see equation 3).*

Proof. Since $U_1, \ldots, U_t$ are invariant subspaces, and the action of σ on $U_{\mathbf{i}}$ is given by:

$$g(e_{i_1} \otimes \ldots \otimes e_{i_k}) = (g(e_{i_1})) \otimes \ldots \otimes (g(e_{i_k})) \in U_{\mathbf{i}}$$

where $e_{i_j} \in U_{i_j}$ indicates $g(e_{i_j}) \in U_{i_j}$. By linearity, the invariance of $U_{\mathbf{i}}$ follows. □

Corollary 3. *Let S,L be as in the lemma, and let* $\phi : R^m \to R^m$ *is defined by:*

$$\phi(v) = \alpha_i v \quad \textit{for } v \in U_i$$

Then ϕ *is equivariant w.r.t.* Aut_S*. In matrix form,* ϕ *is given by:*

$$\phi(T) = \sum_i M_i M_i^T T W_i$$

where $M = (M_1, \ldots, M_t)$ *is orthonormal basis of* $R^m = U_1 \oplus \ldots \oplus U_t$ *(M_i corresponds to U_i),* $T \in R^m$ *and* W_i *is a scalar coefficient.*

Proof. Directly apply lemma 2 and theorem 2, we get the desired result. □

This conveniently gives equivariant maps just by looking at the eigenspaces. The result also generalize to higher-order case.

Theorem 4. *Let* $S, L, M_i's$ *and* $\mathbf{i}$ *defined as above. Define the type of* $\mathbf{i}$ *as the tuple* $\mathbf{n} = (n_1, \ldots, n_t)$*, where* n_j *is the number of occurrences of index* j *in* $\mathbf{i}$ *and we define* $\mathcal{I}_{\mathbf{n}}$ *as the set of all multi-indices of type* $\mathbf{n}$*. Assume we're working on kth-order tensor* $T \in R^{m^k}$*, and define the kth-order eigen-projector*

$$\Pi_{\mathbf{i}} = \mathcal{P}_i \left(M_{i_1}^T \otimes M_{i_2}^T \otimes \cdots \otimes M_{i_k}^T \right) : \mathbb{R}^{m \times \cdots \times m} \to \mathbb{R}^{m \times \cdots \times m}$$

which projects $T \in R^{m^k}$ *to* $U_{\mathbf{i}}$*. Here* $\mathcal{P}_i$ *is a permutation map that canonicalizes the form of the projection (maps those dimensions that correspond to* $i_j = 1$ *to first* n_1 *slots, those dimensions with* $i_j = 2$ *to the second* n_2 *slots and so on). Then for any collection of coefficients* $\{W_{\mathbf{i},\mathbf{i}'} \in R\}$*, the map*

$$\phi : T \mapsto \sum_{\mathbf{n}} \sum_{\mathbf{i}' \in \mathcal{I}_{\mathbf{n}}} \sum_{\mathbf{i} \in \mathcal{I}_{\mathbf{n}}} \Pi_{\mathbf{i}'}^T \Pi_{\mathbf{i}} (T W_{\mathbf{i},\mathbf{i}'})$$

is equivariant w.r.t. the kth-order action of Aut_S *on* R^{m^k}*. This gives in total of* $\sum_{\mathbf{n}} |\mathcal{I}_{\mathbf{n}}|^2 = t^k$ *parameters*

Proof. Note that $R^{m^k} = \bigoplus_{\mathbf{n}} \bigoplus_{\mathbf{i}} U_{\mathbf{i}}$ and all $U_{\mathbf{i}}, \mathbf{i} \in \mathbf{n}$ are isomorphic. Use theorem 3 to get the result. □

Building a *Schur* Layer: The previous result gives the set of equivariant maps on a subgraph S in graph G. Following the standard procedure [18, 15] to extend this to an equivariant map across the graph G, one needs to find all occurrences of S in G, and apply ϕ to each of them with shared weight W. We can also add feature dimensions to kth-order tensor on a subgraph S. We summarize this into the following theorem:

Theorem 5. *A* Schur *layer corresponds to a neuron $\mathfrak{n}_S$ on subgraph S with m vertices, which act the same on all occurrences of S in graph G with n vertices. Assume the maps on each single S is given by 3 (or 4), then $\mathfrak{n}_S$ is equivariant map on G w.r.t. S_n that maps first-order (or kth-order) tensor to first-order (or kth-order) tensor on all occurrences on S. Furthermore, we could add feature dimensions (the dimension that does not permute by $\iota \in S_n$) to the input and output tensor and extend the weight matrix W accordingly. To be specific, in first-order case, $T^{in} \in R^{m \times c_{in}}, T^{out} \in R^{m \times c_{out}}$ and $\mathfrak{n}_S : T^{in} \mapsto T^{out}$ is given by:*

$$\mathfrak{n}_S(T^{in}) = \sum_i M_i M_i^T T^{in} W_i$$

where $W_i \in R^{c_{in} \times c_{out}}$. In kth-order case, $T^{in} \in R^{\underbrace{m \times \ldots m}_{k} \times c_{in}}, T^{out} \in R^{\underbrace{m \times \ldots m}_{k} \times c_{out}}$ and $\mathfrak{n}_S : T^{in} \mapsto T^{out}$ is given by:

$$\mathfrak{n}_S(T^{in}) = \sum_{\mathbf{n}} \sum_{\mathbf{i}' \in \mathcal{I}_{\mathbf{n}}} \sum_{\mathbf{i} \in \mathcal{I}_{\mathbf{n}}} \Pi_{\mathbf{i}'}^T \Pi_{\mathbf{i}} (T \odot W_{\mathbf{i},\mathbf{i}'})$$

where $W_{\mathbf{i},\mathbf{i}'} \in R^{c_{in} \times c_{out}}$ and $\odot$ is the multiplication along the last dimension. We call the resulting neuron Schur *layer since the equivariant maps it constructs are based on the group theoretical idea we described in section 4.1, which is fundamentally based on Schur's lemma 1 of equivariant linear maps between two* irreps.

Proof. First we note that $\mathfrak{n}_S$ is actually equivariant w.r.t. S_m acting on a single instance of S. This is because the local graph Laplacian also permutes with $\sigma \in S_m$. Given $\sigma \in S_m$, the corresponding graph Laplacian L^σ corresponding to $\sigma \circ S$ is $P_\sigma L P_\sigma^T$, thus eigenspaces M_i become $P_\sigma M_i$ after permutation. so

$$\begin{aligned}
\mathfrak{n}_S^\sigma(\sigma \circ T^{\text{in}}) &= \sum_i P_\sigma M_i M_i^T P_\sigma^T P_\sigma T^{\text{in}} W_i \\
&= P_\sigma \sum_i M_i M_i^T T^{\text{in}} W_i \\
&= P_\sigma \mathfrak{n}_S(T^{\text{in}})
\end{aligned}$$

For $\iota \in S_n$, it can be decomposed to two parts: (1) the part that permute the orders of S in G (2) the part that permutes inside each instance of S. And we can consider

the selection process of subgraph S to be (1) find all occurrences of S in G (2) sort them according to alphabetic order comparing the set of their node numbers. So the selection process is also equivariant to ι. Therefore, the $\mathfrak{n}_S$ operates on all occurrences of S in G is equivariant w.r.t. S_n globally.

Lastly, the case of kth-order tensor can be proved the same way.

□

We refer *Schur* net as the graph neural network formed by stacking a set of *Schur* layers together as in standard GNN literature. An order invariant *readout* function (such as summation across nodes) is added at the end for graph property prediction task, and use MLPs to make the final prediction. More details can be found in the experiment section.

5.4 Analysis of *Schur* layer

First of all, we notice that the equivariant map characterized in 3 (or 4) may not be the full set of equivariant maps w.r.t. Aut_S. Because (1) In the decomposition of $U = R^{m^k}$ to eigenspaces $U = U_1 \oplus \dots U_t$, while U_i is stable subspaces, it might not be *irreducible* and finer decomposition may be possible. In other words, there might be two irreps (isomorphic or not) corresponding to the same eigenvalue. (2) The maps defined by 3 didn't take into account the isomorphic subspaces corresponding to the same type of *irrep*, thus ignored the possible equivariant maps between V_j^i and $V_{j'}^i$. So, it is of interest to find out how much the gap would be between our approach and the group theoretical approach. We first look at some examples in the first-order case (see table 1).

We note that for cycles in the graph case, there's no gap. However, when we add branches to the cycle to make the automorphism group smaller, the multiplicities of the *irreps* increase, e.g., in 5-cycle with one branch case, S_2 only have two 1-dimensional *irreps*: trivial and sign representation of permutation, and the first-order action decompose to 4 copies of trivial and 2 copies of sign representation, gives in total $4 * 4 + 2 * 2 = 20$ possible equivariant maps. However, note that # of *irreps* (counting multiplicities given by $\sum_i \kappa_i$) is equal to # of distinct eigenvalues, meaning that each eigenspace corresponds to an irreducible subspace and the decomposition of R^m provided by eigenspaces is indeed a decomposition to the *irreps* in all of the cases listed in the table. Therefore, the multiplicities are the key reason for the gap between our EVD approach and the group theoretical approach, since our EVD approach can't capture the isomorphic property between subspaces. In principle, it is possible to find out which eigenspace is isomorphic to which, but in our current implementation, we didn't take this into account because we found out that only considering cycles is enough for a very good performance in the datasets we used.

Generally, it's complicated to determine the gap between our EVD approach and the group theoretic approach, especially in higher-order cases (where merely determining the κ_i's is tricky), so we leave this into feature exploration.

Graph	Aut_S	**# of distinct Eigenvalues (*Schur* Layer)**	$\sum_i (\kappa_i)^2$ ***irreps* approach**	$\sum_i \kappa_i$
6-cycle	D_6	4	4	4
5-cycle	D_5	3	3	3
4-cycle	D_4	3	3	3
3-cycle	D_3	2	2	2
5-star	S_4	3	5	3
4-star	S_3	3	5	3
3-path	S_2	3	5	3
n-cliques	S_n	2	2	2
5-cycle with one branch	S_2	6	20	6
6-cycle with one branch	S_2	7	29	7

Table 1: Examples on EVD approach vs group theoretical approach towards # of equivariant maps w.r.t. Aut_S. To calculate the decomposition of R^m into *irreps*, one can use 2: $\kappa_i = (\phi|\chi_i)$ where ϕ and χ_i is the character of the first-order action and ith *irrep*, respectively. Another quick approach is to use $\phi(\sigma) = \sum_k \kappa_k \chi_k(\sigma)$ and look at some examples of σ to determine what the κ_k should be.

6. Related works

Higher order MPNNs have become a popular research area in graph representation learning since the seminal work by [13], which characterized the space of equivariant map w.r.t. S_n for kth order tensors. The following works have mainly been divided into two streams, the one is based on k-Weisfeiler Leman test (k-WL test) [16, 17], for instances, [14] proposed k-order graph neural networks that is as powerful as k-WL test; [8] proposed k-GNN to approximate (or simulate) k-WL test; [22] further designed a (k, t)-FWL+ hierarchy that extends the k-WL test and implemented the corresponding neuron version. A common feature of these approaches is they all work on all k-tuples of vertices for k-th order neural networks and thus make their space and time complexity at least $\theta(n^k)$.

The other line of work is seeking ways to choose subgraphs in the graph, that are representative or contain important information, such as functional groups in a molecule. [25] chooses simplicial complex and [21] further extends this to any cell complex such as cycles. [15] incorporates any subgraph template and uses the equivariant maps defined [13] for local feature transform and devised a high order message passing scheme between those subgraphs, called P-tensor. This is arguably the most general framework in this line of work. This line of work is kind of independent of

the k-WL test since the k-WL test is aimed to distinguish any pair of non-isomorphic graphs while chosen subgraphs are meant to work in some specific regions with domain prior. It's possible to still compare this kind of network with the distinguishing power of k-WL test, for example, [21] shows if they include cycles of size at most k (including nodes and edges), their network are as powerful as WL test. However, such comparisons are not very meaningful to indicate the expressive power of such a domain-specific approach. In our opinion, the ability to learn certain features (such as count cycles of certain sizes or learn some function related to specific functional groups) might be a better criterion. It'll be an interesting research direction to design suitable criteria for the expressive power of such higher order MPNNs in general.

Our work follows this line of work since we're choosing specific subgraphs to learn representation on. But we utilize the subgraph's automorphism group to devise more possible equivariant maps (none of the aforementioned methods are built on the automorphism group). The only other work to our knowledge that uses the automorphism group is Autobahn [18], which is the first to introduce equivariance to the automorphism group to GNNs. The difference between our work and it lines twofold: (1) Autobahn theoretically introduces equivariant maps w.r.t. automorphism group via generalized convolution and Cosets, which is another group theoretical approach to construct equivariant maps, while we're using the decomposition to *irreps* (2) More importantly, Autobahn didn't mention a practical approach to construct those equivariant maps explicitly, hindering the utilization of more diverse subgraphs. In their experiments, they only use cycles of length five and six and paths of length three to six and construct the equivariant maps potentially by hand. In contrast, we give a simple and efficient way to construct equivariant maps w.r.t. automorphism group by EVD, which is very general to be used by any subgraph. Though our approach doesn't give the full set of equivariant maps, experiments show improvement over the traditional approach where only equivariant maps w.r.t. S_n are used. We believe this algorithm, together with the group theoretical idea about decomposition R^{m^k} into stable subspaces and *irreps* are beneficial to the research community.

There is a "third" type of higher order GNNs that is called Subgraph GNN, which deviates from the original definition of higher order MPNNs but can still be considered as higher order network. In particular, node-based GNNs such as [26, 12, 27, 28, 29] associate each node with a subgraph (by deleting the node, marking the node or extracting the EGO-network) and do MPNN on each subgraph, where they get the resulting representation $X \in R^{n\times n}$ that can be considered as a 2nd order tensor representation on the original graph.

7. Experiments

7.1 Implementation of *Schur* net

To empirically evaluate *Schur* net, we implement the first order case in theorem 5 and compare it with other higher order MPNNs. Note that, the theorem only gives the

equivariant maps to transform local representation on a subgraph, i.e., $\phi : T^{\text{in}} \mapsto T^{\text{out}}$ where $T^{\text{in}} \in R^{m*c_{\text{in}}}$ and $T^{\text{out}} \in R^{m*c_{\text{out}}}$ are representations of the subgraph. Another key component that is missing is the message passing between different subgraphs, which as mentioned in the previous section, is already given by [15] (P-tensor). In P-tensor framework, the most general linear equivariant message passing is defined based on an extension of equivariant maps by considering different domains. More details can be found in appendix 9. We implement our *Schur* net based on the P-tensor framework but use *Schur* layer to transform local representation.

There are several design details about the architecture that are worth mentioning: (1) We chose cycles of lengths three to eight in most of the experiments and also added branched cycles to show our algorithm's scalability. The reason to do so is in the chemical dataset we used, cycles are indeed the most important functional group to the property to be predicted, other subgraphs such as m-stars, and m-paths didn't help the performance. (2) While having first-order representation on the cycles, we also maintain 0th-order representation (just scalar) on node and edges, as in [21, 15]. Those node and edge representations captures more elementary information about the graph, such as k-hop neighborhood, and is still important in higher order MPNNs. The representations pass messages with each other by intersection as defined in P-tensor framework. (3) As discussed in appendix 11, we view *Schur* layer's operation as a spectral convolution filter [30] applied to the subgraph and the number of channels to indicate how many times it expands the input feature to the output feature. Complete details about the architecture and experiments can be found in appendix 12. We will also make our code publicly accessible after submitting the thesis.

7.2 *Schur* layer improves over *Linmaps*

First of all, we want to show that considering more possible equivariant maps on the subgraph can indeed boost the performance. To this end, we performed controlled experiments to compare *Schur* layer with the equivariant maps w.r.t. S_m (following [15] we called it *Linmaps*). To make a fair comparison, we use the same architecture including MLPs and message passing defined by P-tensor and only replace the equivariant maps used in *Linmaps* by what is defined in theorem 5. We didn't compare with Autobahn [18] because it didn't use the P-tensor framework in the original paper and it's hard for us to implement the convolution w.r.t. automorphism group for all the subgraphs we chose. But we note that for the case of the cycles (see table 1), the equivariant maps given by our approach are equal to that given by the group theoretical approach.

We first start with some molecular datasets in TUdataset [31], which is a small but commonly used benchmark for GNN. In table 2, we see that *Schur* layer consistently improves over *Linmaps* in various bioinformatics and molecule datasets.

Then we move on to the larger scale dataset ZINC-12K [32]. It is a commonly used molecular benchmark and contains 12K molecules, divided into training, validation, and test set with a ratio of 10:1:1. The task on it is to regress the $logP$ coefficient of

Dataset	*Linmaps*	*Schur* Layer
Proteins	74.7 ± 3.8	**75.4 ± 4.8**
MUTAG	89.9 ± 5.5	**90.94 ± 4.7**
PTC_MR	61.1 ± 6.9	**64.6 ± 5.9**
NCI1	82.1 ± 1.8	**82.7 ± 1.9**

Table 2: Comparison of *Linmaps* and *Schur* Layer performance on TUdatasets. Numbers are binary classification accuracy.

each molecule and the Mean absolute error (MAE) between the prediction and the ground truth is used for evaluation.

We design experiments to compare *Linmaps* and *Schur* layers in various scenarios, to showcase the robustness of the improvement. In table 3, we see that *Schur* layer outperforms *Linmaps* when certain cycle sizes are considered, especially when only cycles 5 and 6 are chosen as subgraphs in the neural network. Moreover, table 4 shows under various message passing schemes between edges and cycles, *Schur* Layer consistently outperforms *Linmaps*. Those are indications of the added expressive power of extra equivariant maps in *Schur* layer, and they're effective in various architectural design settings.

Layer	cycle size {5,6}	cycle size {3,4,5,6}	cycle size {3,4,5,6,7,8}
Linmaps (baseline)	0.149 ± 0005	0.128 ± 0.002	0.112 ± 0.001
Schur layer	$\mathbf{0.129 \pm 0.006}$	$\mathbf{0.120 \pm 0.006}$	$\mathbf{0.111 \pm 0.004}$

Table 3: Comparison between *Schur* Layer and *Linmaps* with different set of cycles chosen. Experiments on ZINC-12k dataset and all scores are validation MAE.

Layer	Pass message when overlap ≥ 2	Pass message when overlap ≥ 1
Linmaps (baseline)	0.085 ± 0.007	0.075 ± 0.003
Schur layer	$\mathbf{0.076 \pm 0.004}$	$\mathbf{0.072 \pm 0.002}$

Table 4: Comparison between *Schur* Layer and *Linmaps* with different message passing schemes. The message passing scheme is a design choice in P-tensor framework, where the user can set when two subgraph's representations communicate. The mostly common use case is to require at least k vertices in the intersection of two subgraphs for them to communicate. Experiments on ZINC-12k dataset and all scores are validation MAE. Cycle sizes of {3,4,5,6,7,8} are used

In table 5, we compare the runtime of our *Schur* layer and *Linmaps*, which shows the extra computational cost of *Schur* layer wasn't significant while being able to use more equivariant maps and achieving better accuracy.

Dataset	*Linmaps*	*Schur* Layer
Zinc-12k	25.4s	27.6s
NCI1	9.5s	11.5s

Table 5: Runtime per epoch with hyper-params num_layers = 4, rep_dim = 128, dropout = 0.0, batch_size = 256, num of channels = 4, cycle_sizes = 3,4,5,6,7,8. The implementation of *Linmaps* and *Schur* Layer is based on our internal software. We're devoted to optimizing the backend to make it faster now. But this also shows *Schur* Layer didn't add much computational costs to Linmaps while being more expressive.

7.3 Usage of *Schur* layer

Then we studied the possibilities of adding *Schur* layer in different places of higher-order message passing scheme.

In table 6, tried different ways to use *Schur* layer. We observed that the more condensed the higher-order feature is, the more improvement that the *Schur* Layer brings to us over *Linmaps*. We attribute the improvements of adding/replacing *Schur* layer in various scenarios over *Linmaps* the benefit gained from utilizing the subgraph structure and increased number of equivariant maps. We also tried other ways to use *Schur* layer, the result is summarized in 11.

7.4 Flexibility

The other advantage of *Schur* layer is it computes the feature transform only based on the subgraph's Laplacian, bypassing a difficult step of finding the automorphisms group and *irreps* of the subgraph it acts on.

As discussed in the theory, *Schur* layer constructs equivariant maps only based on the subgraph's Laplacian and is applicable directly to any subgraphs, making the implementation much easier when different subgraphs are chosen than the group theoretical approach. This allows it to easily extend to any subgraph templates that're favorable by the user. To demonstrate this, we augment the subgraphs in the model by all the five and six cycles with one to three branches (including in total 16 non-isomorphic subgraph templates), comparing with baseline model where only the cycle itself is considered. In table 7, we see that adding cycles with branches could provide the *Schur* Net with more detailed topological information, thus improving the performance. In appendix 12, we show the code for adding this, which actually

Model	Test MAE
Linmaps	0.071 ± 0.004
Simple *Schur*-Net	0.070 ± 0.005
Linmap *Schur*-Net	$\mathbf{0.068 \pm 0.002}$
Complete *Schur*-Net	$\mathbf{0.064 \pm 0.002}$

Table 6: An experiment demonstrating different ways of using *Schur* layer. "Complete *Schur* Layer" means that we apply *Schur* Layer on the incoming messages together with the original cycle representation. "Linmap SchuLayer" means that we just apply the *Schur* Layer on the aggregated subgraph representation feature. "Simple *Schur* Layer" means we directly apply *Schur* Layer on the subgraph features without any preprocessing. We can observe that as the subgraph information diversifies, *Schur* layer tends to decouple the dense information better and results in better performance. The test MAE of *Linmaps* in this table is taken from [15].

only requires a single definition of those templates without modification to the neural network.

Model	Validation MAE
Schur-Net on 5,6 cycles(baseline)	0.118
Schur-Net on 5,6 cycles with up to three branches	**0.106**

Table 7: Flexibility of *Schur* layer. Experiments on ZINC-12k dataset. All other settings are the same. A smaller network than previous experiments was used.

7.5 Benchmark results

Finally, and most importantly, we compare the *Schur*-Net to several other higher-order MPNNs on ZINC-12k dataset and OGB-HIV dataset [33] in table 8. We included baselines of (1) classical MPNNs: GCN[1], GIN [7], GINE [9], PNA[34], HIMP [35] (2) higher order MPNN of the first kind: N^2-GNN [22] [3], (3) higher order MPNN of the second kind: CIN [21], P-tensors [15] (4) Subgraph-GNNs: DS-GNN(EGO+) and DSS-GNN(EGO+) [27], GNN-AK+ [12], SUN(EGO+)[29] (5) Autobahn [18].

We find that *Schur* Net ranked second on ZINC-12K and outperformed all other baselines on OGB-HIV dataset. This shows the expressivity of adding more equivariant maps by leveraging the subgraph topology. Furthermore, note that while N^2-GNN outperforms *Schur* Net on ZINC-12K, it's a second-order model whereas in our

3. The works [14, 8] don't have results in those two dataset

Model	ZINC-12K MAE(↓)	OGB-HIV ROC-AUC(% ↑)
GCN	0.321 ± 0.009	76.07 ± 0.97
GIN	0.408 ± 0.008	75.58 ± 1.40
GINE	0.252 ± 0.014	75.58 ± 1.40
PNA	0.133 ± 0.011	79.05 ± 1.32
HIMP	0.151 ± 0.002	78.80 ± 0.82
N^2-GNN	$\mathbf{0.059 \pm 0.002}$	-
CIN	0.079 ± 0.006	80.94 ± 0.57
P-tensors	0.071 ± 0.004	80.76 ± 0.82
DS-GNN (EGO+)	0.105 ± 0.003	77.40 ± 2.19
DSS-GNN (EGO+)	0.097 ± 0.006	76.78 ± 1.66
GNN-AK+	0.091 ± 0.011	79.61 ± 1.19
SUN (EGO+)	0.084 ± 0.002	80.03 ± 0.55
Autobahn	0.106 ± 0.004	78.0 ± 0.30
Schur-Net	0.064 ± 0.002	$\mathbf{81.6 \pm 0.295}$

Table 8: Comparison of different models on the ZINC-12K and OGBG-MOLHIV datasets

experiment, we only used first-order activation. Also, the partial reason *Autobahn* didn't perform well is in the original implementation, the author didn't use P-tensor framework and used only a part of all possible linear message passing schemes. This shows to get the full power of equivariant maps w.r.t. subgraph automorphism group, we need to combine it with a general message passing framework between subgraphs as well.

8. Conclusion

In this thesis, we present a group theoretical approach towards constructing equivariant linear maps w.r.t. S_m (or its subgroups). We then point out the importance of considering the subgraph's automorphism group to learn the local structure. Towards this end, we come up with an algorithm based on spectral graph theory that could directly obtain a large number of equivariant maps just by an EVD of the graph Laplacian, which is much simpler than the group theoretical approach and thus is more scalable. Although in theory there are scenarios that our approach, *Schur* layer, didn't capture all the possible equivariant maps w.r.t. the automorphism group, we show in practice, it achieves strong performance in common chemical datasets: ZINC-12K and OGB-HIV.

Additionally, our approach exposes heretofore unexplored connections between permutation equivariance and spectral GNNs such as [36, 37]. It also highlights the fact that while permutation equivariance is a fundamental constraint on graph neural networks, the key to building high-performing GNNs is to use as much side information about the graph topology as possible, including graph adjacency matrix, vertex label or degrees and etc, to reduce the size of the group that the network needs to be equivariant to.

8.1 Limitations and future directions

As discussed in section 5.4, *Schur* layer didn't capture all possible equivariant maps w.r.t. the local automorphism group, especially when the multiplicities of *irreps* in the decomposition of group action are high. To theoretically explore what are the gaps in all scenarios and to find a way to incorporate the maps enabled by isomorphic eigenspaces under group action are interesting future directions to explore.

For experiments, we only used first-order activations and chose subgraph as cycles and branched cycles in this thesis. Potentially, a more diverse set of subgraphs and higher-order activations could be beneficial in various application settings. Also, leveraging the side information provided by nodes' label to further reduce the automorphism group would also increase the number of equivariant maps and thus increase expressivity.

Furthermore, applying the powerful GNN with *Schur* layer to other graph learning tasks (like node classification, link prediction, etc) and graph generation tasks is also interesting. For example, we can explore how much improvement a more expressive architecture like *Schur* layer could bring us over the traditional GNNs in molecule generation, and what are the most suitable type of GNN for the generation task.

References

[1] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. 5, 2017.

[2] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *ICML*, 2017.

[3] Pau Gainza, Felix Sverrisson, Federico Monti, Emanuele Rodola, Davide Boscaini, Michael M Bronstein, and Bruno E Correia. Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning. *Nature Methods*, 17(2):184–192, 2020.

[4] Chendi Qian, Andrei Manolache, Kareem Ahmed, Zhe Zeng, Guy Van den Broeck, Mathias Niepert, and Christopher Morris. Probabilistically rewired message-passing neural networks, 2024. URL `https://arxiv.org/abs/2310.02156`.

[5] Vikraman Arvind, Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky. On weisfeiler-leman invariance: Subgraph counts and related graph properties. *Journal of Computer and System Sciences*, 113:42–59, 2020.

[6] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? In *Advances in Neural Information Processing Systems (NIPS)*, 2020.

[7] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks. *Int. Conf. on Learning Representations*, pages 1–17, 2019.

[8] Christopher Morris, Gaurav Rattan, and Petra Mutzel. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. *arXiv preprint arXiv:1904.01543*, 2019.

[9] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay S. Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2020.

[10] Emily Alsentzer, Samuel Finlayson, Michelle Li, and Marinka Zitnik. Subgraph neural networks. *Advances in Neural Information Processing Systems*, 33:8017–8029, 2020.

[11] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. *33rd AAAI Conference on Artificial Intelligence*, 2019. ISSN 2159-5399.

[12] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. From stars to subgraphs: Uplifting any gnn with local structure awareness, 2022. URL `https://arxiv.org/abs/2110.03753`.

[13] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2019.

[14] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/bb04af0f7ecaee4aae62035497da1387-Paper.pdf`.

[15] Andrew R. Hands, Tianyi Sun, and Risi Kondor. P-tensors: a general framework for higher order message passing in subgraph neural networks. In Sanjoy Dasgupta,

Stephan Mandt, and Yingzhen Li, editors, *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, pages 424–432. PMLR, 02–04 May 2024. URL https://proceedings.mlr.press/v238/hands24a.html.

[16] Boris Weisfeiler and Andrei Leman. The reduction of a graph to a canonical form and the algebra which appears therein. *Nauchno-Technicheskaya Informatsiya*, 9 (2):12–16, 1968.

[17] Martin Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Cambridge University Press, 2017.

[18] Erik Henning Thiede, Wenda Zhou, and Risi Kondor. Autobahn: Automorphism-based graph neural nets, 2022. URL https://arxiv.org/abs/2103.01710.

[19] Jean-Pierre Serre. *Linear representations of finite groups*. Springer-Verlag, New York, 1977. ISBN 0-387-90190-6. Translated from the second French edition by Leonard L. Scott, Graduate Texts in Mathematics, Vol. 42.

[20] B.E. Sagan. *The Symmetric Group: Representations, Combinatorial Algorithms, and Symmetric Functions*. Graduate Texts in Mathematics. Springer New York, 2013. ISBN 9781475768046. URL https://books.google.com/books?id=Y6vTBwAAQBAJ.

[21] Cristian Bodnar, Fabrizio Frasca, Yu Guang Wang, Nina Otter, Guido Montúfar, Pietro Liò, and Michael Bronstein. Weisfeiler and Lehman Go Topological: Message Passing Simplicial Networks. 2021. URL http://arxiv.org/abs/2103.03212.

[22] Jiarui Feng, Lecheng Kong, Hao Liu, Dacheng Tao, Fuhai Li, Muhan Zhang, and Yixin Chen. Extending the design space of graph neural networks by rethinking folklore weisfeiler-lehman. In *Advances in Neural Information Processing Systems*, 2023.

[23] Erik Henning Thiede, Truong-Son Hy, and Risi Kondor. The general theory of permutation equivarant neural networks and higher order graph variational encoders. *CoRR*, abs/2004.03990, 2020. URL https://arxiv.org/abs/2004.03990.

[24] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive

biases, deep learning, and graph networks, 2018. URL https://arxiv.org/abs/1806.01261.

[25] Cristian Bodnar, Fabrizio Frasca, Yu Guang Wang, Nina Otter, Guido Montúfar, Pietro Liò, and Michael Bronstein. Weisfeiler and lehman go topological: Message passing simplicial networks, 2021. URL https://arxiv.org/abs/2103.03212.

[26] Muhan Zhang and Pan Li. Nested graph neural networks, 2021. URL https://arxiv.org/abs/2110.13197.

[27] Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M. Bronstein, and Haggai Maron. Equivariant subgraph aggregation networks, 2022. URL https://arxiv.org/abs/2110.02910.

[28] Jiaxuan You, Jonathan Gomes-Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks, 2021. URL https://arxiv.org/abs/2101.10320.

[29] Fabrizio Frasca, Beatrice Bevilacqua, Michael M. Bronstein, and Haggai Maron. Understanding and extending subgraph gnns by rethinking their symmetries, 2022. URL https://arxiv.org/abs/2206.11140.

[30] Deyu Bo, Xiao Wang, Yang Liu, Yuan Fang, Yawen Li, and Chuan Shi. A survey on spectral graph neural networks, 2023.

[31] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. TUDataset: A collection of benchmark datasets for learning with graphs. URL http://arxiv.org/abs/2007.08663.

[32] John J Irwin, Khanh G Tang, Jennifer Young, Chinzorig Dandarchuluun, Benjamin R Wong, Munkhzul Khurelbaatar, Yurii S Moroz, John Mayfield, and Roger A Sayle. Zinc20—a free ultralarge-scale chemical database for ligand discovery. *Journal of chemical information and modeling*, 60(12):6065–6073, 2020.

[33] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in neural information processing systems*, volume 33, pages 22118–22133, 2020.

[34] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Velickovic. Principal neighbourhood aggregation for graph nets. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

[35] Matthias Fey, Jan-Gin Yuen, and Frank Weichert. Hierarchical inter-message passing for learning on molecular graphs, 2020.

[36] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral Networks and Locally Connected Networks on Graphs. *International Conference on Learning Representations (ICLR)*, page 14, 2014. URL `http://arxiv.org/abs/1312.6203`.

[37] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep Convolutional Networks on Graph-Structured Data. *arXiv*, 2015.

[38] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. URL `https://arxiv.org/abs/1502.03167`.

[39] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

9. A brief overview about P-tensor framework

The P-tensor framework is a framework for linear equivariant maps w.r.t. S_n both between the same subgraph and across different subgraphs. It is built on the equivariant maps characterized by [13].

Definition 12 (P-tensors)**.** Let U be a finite set of atoms and $D = (x_1, \dots, x_d)$ an ordered subset of U. We say that a k-th order tensor $T \in \mathbb{R}^{d \times d \times \cdots \times d}$ is a k-th order permutation covariant tensor (or P-tensor for short) with reference domain D if under reordering by $\tau \in S_d$ D it transforms to

$$[\tau \circ T]_{i_1, i_2, \dots, i_k} = T_{\tau^{-1}(i_1), \dots, \tau^{-1}(i_k)}.$$

9.1 Message passing between P-tensors with the same reference domain

Consider the equivariant maps sending T on D to T^{out} on D. The space of equivariant maps w.r.t S_m is characterized by [13]:

Proposition 4. *The space of linear maps $\phi : \mathbb{R}^{dk_1} \to \mathbb{R}^{dk_2}$ that is equivariant to permutations $\tau \in S_d$ is spanned by a basis indexed by the partitions of the set $\{1, 2, \dots, k_1 + k_2\}$.*

Then the authors designed a straightforward way to write the maps explicitly. Specifically, for each partition of the set $\{1, 2, \dots, k_1 + k_2\}$, there are three parts that determines the equivariant map: (1) summing over specific dimensions or diagonals of T^{in} (2) transferring T^{in} to T^{out} by identifying indices of T^{in} with indices of T^{out} (3) broadcasting the result along certain dimensions of T^{out}. These three operations correspond to the three different types of sets that can occur in a given partition $\mathcal{P}$: (1) those that only involve the second $k2$ numbers, (2) those that involve a mixture of the first k_1 and k_2 and (3) those only involve the first k_1 numbers. The type (1) - (3) corresponds to type (1) - (3) of operations with the dimensions in the sets.

For example, in the case $k_1 = k_2 = 3$, the $\mathcal{P} = \{\{1, 3\}, \{2, 5, 6\}, \{4\}\}$ partition corresponds to (a) summing T^{in} along its first dimension (corresponding to $\{4\}$) (b) transferring the diagonal along the second and third dimension of T^{in} to the second dimension of T^{out} (corresponding to $\{2, 5, 6\}$) (c) broadcasting the result along the diagonal of the first and third dimensions (corresponding to $\{1, 3\}$). Explicitly, this gives the equivariant map:

$$T^{\text{out}}_{a,b,a} = \sum_c T^{\text{in}}_{c,b,b}$$

See table 9 for another example for $k_1 = k_2 = 2$ case.

9.2 Message passing between P-tensors with the different reference domains

If T^{in} has reference domain D_1 and T^{out} has D_2, with $D_1 \neq D_2$ and $D_1 \cap D_2 \neq \emptyset$. We could have more options corresponding to summing either over the intersection or

$\mathcal{P}$	ϕ	$\mathcal{P}$	ϕ
{{1}, {2}, {3}, {4}}	$T^{\text{out}}_{a,b} = \sum_{c,d} T^{\text{in}}_{c,d}$	{{2}, {1,3,4}}	$T^{\text{out}}_{b,a} = T^{\text{in}}_{b,b}$
{{1}, {2}, {3,4}}	$T^{\text{out}}_{a,b} = \sum_{c} T^{\text{in}}_{c,c}$	{{1,2,3}, {4}}	$T^{\text{out}}_{a,a} = \sum_{b} T^{\text{in}}_{a,b}$
{{1}, {2,4}, {3}}	$T^{\text{out}}_{a,b} = \sum_{c} T^{\text{in}}_{c,b}$	{{1,2,4}, {3}}	$T^{\text{out}}_{a,a} = \sum_{b} T^{\text{in}}_{b,a}$
{{1}, {2,3}, {4}}	$T^{\text{out}}_{a,b} = \sum_{c} T^{\text{in}}_{b,c}$	{{1,2}, {3,4}}	$T^{\text{out}}_{a,a} = \sum_{c} T^{\text{in}}_{c,c}$
{{2}, {1,4,3}}	$T^{\text{out}}_{b,a} = T^{\text{in}}_{b,b}$	{{1,3}, {2,4}}	$T^{\text{out}}_{a,b} = T^{\text{in}}_{a,b}$
{{1,3}, {2,4}}	$T^{\text{out}}_{b,a} = \sum_{c} T^{\text{in}}_{c,b}$	{{1,4}, {2,3}}	$T^{\text{out}}_{b,a} = T^{\text{in}}_{b,a}$
{{1,2,3,4}}	$T^{\text{out}}_{a,a} = T^{\text{in}}_{a,a}$	{{1,2,3,4}}	$T^{\text{out}}_{a,a} = T^{\text{in}}_{a,a}$

Table 9: The $\mathcal{B}(4) = 15$ possible partitions of the set $\{1, 2, 3, 4\}$ and the corresponding permutation equivariant linear maps $\phi : \mathbb{R}^{k\times k} \to \mathbb{R}^{k\times k}$.

over D_1, and broadcasting either over the intersection or over D_2. So the number of maps corresponging to partition of type (p_1, p_2, p_3) is $2^{(p_1+p_3)}$. Let $D_1 \cap D_2 = d^{\cap}$, the previous example would have the following maps in $D_1 \neq D_2$ case:

$$T^{\text{out}}_{a,b,a} = \begin{cases} \sum_{c=1}^{d^{\cap}} T^{\text{in}}_{c,b,b} & a, b \leq d^{\cap} \\ 0 & \text{otherwise,} \end{cases} \quad (a \in \{1, \dots, d^{\cap}\},\ c \in \{1, \dots, d^{\cap}\})$$

$$T^{\text{out}}_{a,b,a} = \begin{cases} \sum_{c=1}^{d^{\cap}} T^{\text{in}}_{c,b,b} & b \leq d^{\cap} \\ 0 & \text{otherwise,} \end{cases} \quad (a \in \{1, \dots, d_2\},\ c \in \{1, \dots, d^{\cap}\})$$

$$T^{\text{out}}_{a,b,a} = \begin{cases} \sum_{c=1}^{d_1} T^{\text{in}}_{c,b,b} & a, b \leq d^{\cap} \\ 0 & \text{otherwise,} \end{cases} \quad (a \in \{1, \dots, d^{\cap}\},\ c \in \{1, \dots, d_1\})$$

$$T^{\text{out}}_{a,b,a} = \begin{cases} \sum_{c=1}^{d_1} T^{\text{in}}_{c,b,b} & b \leq d^{\cap} \\ 0 & \text{otherwise,} \end{cases} \quad (a \in \{1, \dots, d_2\},\ c \in \{1, \dots, d_1\})$$

10. Alternative Proof for Corollary 3

Here we give a more direct proof for Corollary 3.

Proof. Since the M_i's and λ_i's are eigenspaces and eigenvectors of the Laplacian L, $L = M\Lambda M^T$, where $M = [M_1, \cdots, M_p]$ and $\Lambda = \text{diag}(\lambda_1, \cdots, \lambda_m)$.

The neuron $\mathfrak{n}_S$ is a linear transform $\phi : R^m \to R^m$, with $\phi(T) = \Sigma_i^p M_i M_i^T T W_i = MVM^T T$, where $V = \text{diag}(W_1 I_{n_1}, \cdots, W_p I_{n_p})$. Here the W_i's are just scalars and I_{n_i} is an identity matrix of the same size as the corresponding eigenspace.

Given a permutation $\sigma \in S_m$, we want to show that $\phi_\sigma(\sigma \circ T) = \sigma \circ \phi_e(T)$, which is the definition of equivariance. Note that ϕ_σ actually depends on σ, since we're doing eigenvalue decomposition on the Laplacian of the subgraph transformed by σ. This is the key to proving equivariance. We use e to denote the identity permutation.

Now $L^\sigma = P_\sigma L P_\sigma^T = M_\sigma \Lambda M_\sigma^T$, where $M_\sigma = P_\sigma M$, while the eigenvalues remain invariant after permutation and eigenspaces are transformed in the same manner as L. Thus,

$$\begin{aligned}\phi_\sigma(\sigma \circ T) &= M_\sigma V M_\sigma^\top P_\sigma T \\ &= P_\sigma M V M_\sigma^\top P_\sigma^\top P_\sigma T \\ &= P_\sigma \, \phi_e(T)\end{aligned}$$

which is the desired result.

□

Remark 3. Note that in the proof we actually get equivariance w.r.t. S_m, not just w.r.t. to the local automorphism group. So we don't need to find a canonical ordering of the subgraph. This takes the advantage that L is permuted by $\sigma \in S_m$, so it kind of "remembers" the renumbering.

The key to the proof is that the transform ϕ depends on an object transformed with permutation σ, specifically. graph Laplacian L. And all we need is that the object L is transformed equivariantly with σ. We can also add side information such as node labels or degrees to further constrain the automorphism group and increase the number of distinct eigenvalues. One easy way to do that is to set $L' = L + V$, where $V = \mathrm{diag}(v_1, \cdots, v_m)$ captures the node labels or degrees, and build the Schur neuron $\mathfrak{n}_S$ from L'.

11. Ways to use *Schur* layer

While *Schur* layer can be viewed as a generic transform on any subgraph's kth order activation, we provide some thoughts and possibilities to instantiate it.

In our experiment, we primarily view *Schur* layer as a way to expand the feature of a subgraph. It can be viewed as an analogue to CNN's convolution filter and it's indeed a constrained version of spectral convolution filter [30] on the subgraph. In light of this, we call *number of channels* of *Schur* layer as how many times it expand the output feature versus the input feature.

Besides, we observe that the linear equivariant map described [13] is a special case of *Schur* layer's map. In first order case, the two possible linear maps in [13] is (1) identity map, corresponding to take all weights $W_i = 1$ (2) $T \to \Sigma_i T_i \mathbb{1}$, where $\mathbb{1}$ is all one vector, corresponding to set $W_1 = 1$, and $W_i = 0$ for $i \neq 1$ since $\mathbb{1}$ is eigenvector of any graph Laplacian with eigenvalue 0. Consequently, we suggest to keep the two basic but important linear maps and augment them with *Schur* layer.

Moreover, we suggest that number of channels of *Schur* layer be proportional or approximately equal to number of distinct eigenvalues of the subgraph it attach to. This is because the later is the number of independent linear maps for *Schur* layer. Empirically we verified this by an experiment that only varied the number of channels. In figure 2, we see that further increase the number of channels beyond 4 wouldn't

give us performance gain since cycle 5 and 6 only has 3 and 4 distinct eigenvalues respectively.

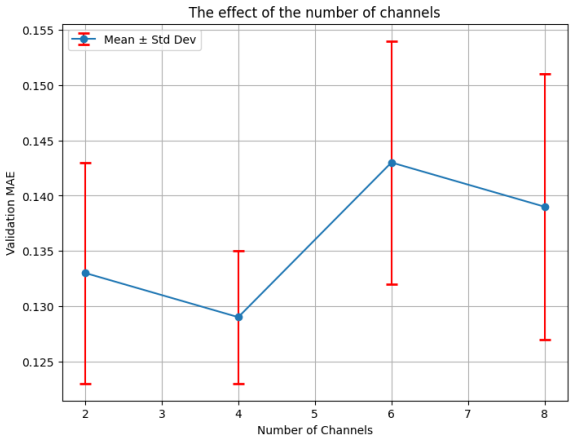


Figure 2: A study on num of channels in Schur layer. Cycle 5 and 6 are included.

Finally, we're wondering if we can use *Schur* layer in place of MLP's to provide a local structure aware transform to subgraph activations. Therefore, we tried to replace the 2-layer MLP with 2-layer *Schur* layer in the original *Linmaps* architecture. But it turns out this use case wasn't as effective as the use as learning new feature. Possibly because *Schur* layer learns feature itself while MLP transform the learned feature to some desired space.

Methods	Validation MAE
Linmaps	0.087
Schur layer (in place of MLP)	0.081
Schur layer (as learning new feature)	0.076

Table 10: Comparison about two use cases of *Schur* layer. Experiment on ZINC-12K.

12. Architecture and experiment details

In designing of our architecture, we follow the high order message passing scheme proposed by P-tensor (see 9 for detail) and add *Schur* layer to each of the convolution layers. A visualization of our architecture can be found in figure 3 In each convolution layer, we maintain 0th-order node representation ($h_v \in R^{|V|*dim}$) and edge representation ($h_e \in R^{|E|*dim}$) where the input graph is $G = (V, E)$. We further maintain 1st-order representation on cycles ($h_{D_k} \in R^{(k*(c_k)*dim)}$) where D_k is cycle of length

k and c_k is the number of such cycle in G. We did message passing between node and edge representations the same as CIN [21]. The edge and cycle pass messages with each other by transfer maps described by [15] and then the incoming message to cycles as well as the original cycle representation is transformed by *Schur* layer. When updating the edge representation and cycle representation, we combine the original representation with the incoming message by either concatenation or ε-add described in GIN [7] and feed it into an MLP to get new representations.

Hyperparameters For experiments on ZINC-12K and OGB-HIV, we tune representation dimension in $\{32, 64, 128\}$, and experiment on cycles up to length 18. In the best-performing model, we use representation dimension 128 and cycle lengths from 3 to 8 and number-of-layers is always 4. For SchuLayer, we tuned number of channels from 2 to 8 and found $2, 4$ are a suitable choice when it is used as an augmentation to *Linmaps*. For MLPs, we either use 2 to 3 layers depending on how dense the input's information is. We always use a batchnorm [38] after the Linear layer and then do ReLu activation. For training hyperparameters, we use an init learning rate of 0.01 and use ReduceLROnPlateau scheduler in PyTorch with patience of 10 or 30. We use Adam optimizer [39] for all trainings. We train for 500 or 1000 epochs depending on model size. In particular, in table 3, the models have representation dimension 32, so it is trained to only 500 epochs. All other models are trained 1000 epochs. A batch size of 256 is used for all models. All results are run at least twice (most of them at least three times) to calculate the standard deviation.

For experiments on TUdataset (table 2), we chose a set of hyper-params by intuition (we didn't tune them because this experiment is to demonstrate the effectiveness of *Schur* layer and compare *Schur* layer with *Linmaps* under the same experiment setting. we don't aim to compare with Sota on this experiment). Hyper-params for both *Schur* Layer and *Linmaps*: num_layers = 4, rep_dim = 32,64, dropout = 0.5, batch_size = 32,128, lr = 0.01, num of channels = {2,4}, cycle sizes = 3,4,5,6. We're training with StepLR scheduler where reduce lr to 1/2 after every 50 epochs, with a total of 300 epochs. The hyperparams weren't tuned, we just chose a smaller value for a smaller dataset and a bigger value for bigger datasets by rule-of-thumb. Linmaps is implemented on our own according to the description of P-tensor, then *Schur* Layer replaces Linmaps operation by *Schur* operation. We follow the evaluation strategy specified in [7].

Used Compute Resources Experiments are all run on one Tesla L4 from PyTorch Lightning Workspace and one NVIDIA RTX 4090. The code is implemented in PyTorch. For the running time, on the ZINC-12k dataset, it takes around 8.53 ± 1.2 hours to finish training for 1000 epochs with 4 layers and 128 dimension representation on an NVIDIA RTX 4090. We're running on a desktop with a Ryzen 7900 CPU and 64GB of RAM.

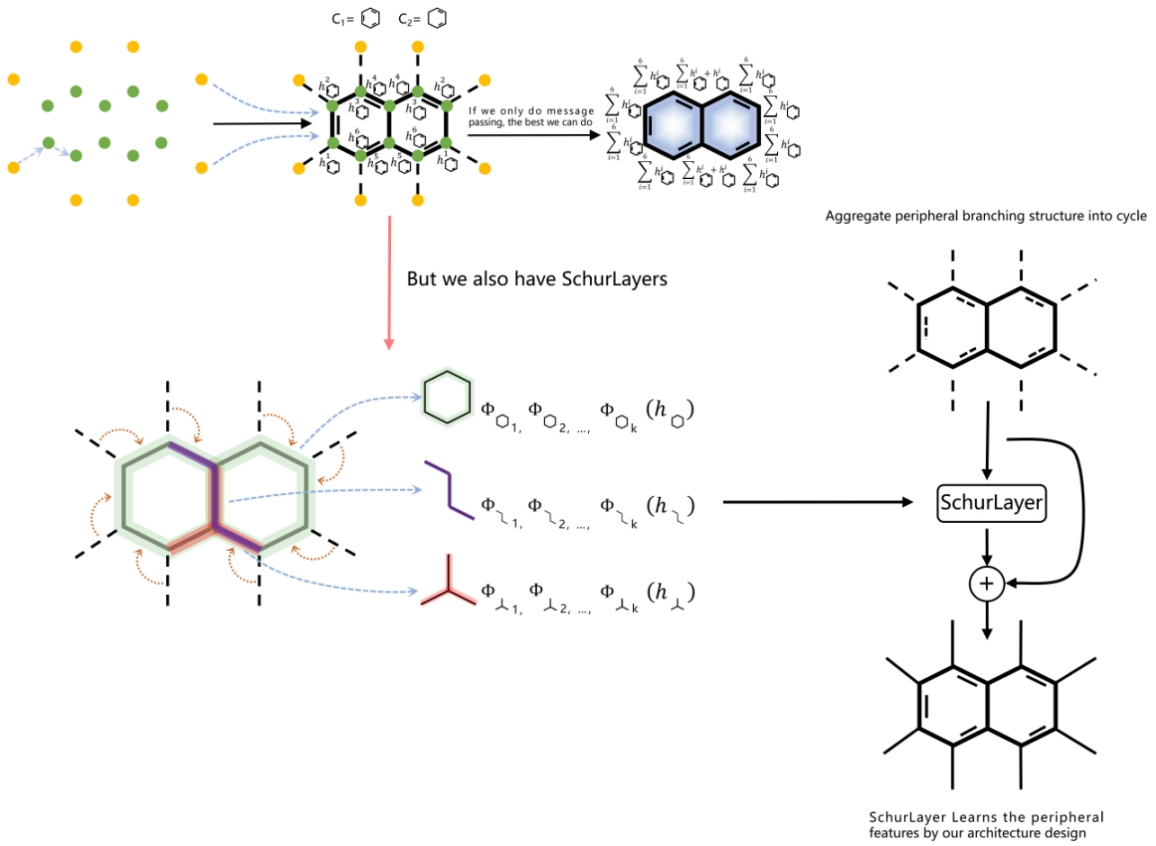


Figure 3: An example illustration of one layer of our model architecture. For an input graph, we first execute the standard node and edge level message passing and lift up the model to higher-order by forming higher-order representations on specific template graphs. In this example, we learned the representation on a Naphthalene. To further decouple its information, we may apply the SchurLayer on the template graphs that would potentially provide us insight about the graph. In this example, the 6 cycle, star graph, and path graph are all applying SchuLayer to learn the features locally. Through our architecture design, we can also incorporate the peripheral information. Automorphism group then helps us discover distinct key features such as non-symmetries, and we pass these aggregated features into the next layer of the architecture.

12.1 Adding branched cycles

In experiment 7.4, We augment the subgraphs in the model by all the five and six cycles with one to three branches (including in total 16 non-isomorphic subgraph templates). Here we show the code of doing this, which just includes defining the

subgraph templates and finding them in the graph. All other calculations follow the same pipeline as if there were only cycles of size fix and six, showing that *Schur* layer is easy to extend to any user-defined subgraphs.

```
import torch
import ptens as p

#----------------cycle5------------
#cycle5_one_branch
cycle5_one_branch = torch.tensor([
    [0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5],
    [1, 5, 0, 2, 1, 3, 2, 4, 3, 0, 0]
])
degree = torch.tensor([3, 2, 2, 2, 2, -1], dtype=torch.float)
cycle5_one_branch = p.subgraph.from_edge_index(cycle5_one_branch.
   float(), degrees=degree)

#cycle5_two_branch_00
cycle5_two_branch_00 = torch.tensor([
    [0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 6],
    [1, 5, 6, 0, 2, 1, 3, 2, 4, 3, 0, 0, 0]
])
degree = torch.tensor([4, -1, -1, -1, -1, -1, -1], dtype=torch.float
   )
cycle5_two_branch_00 = p.subgraph.from_edge_index(
   cycle5_two_branch_00.float(), degrees=degree)

#cycle5_two_branch_01
cycle5_two_branch_01 = torch.tensor([
    [0, 0, 1, 1, 1, 2, 2, 3, 3, 4, 4, 5, 6],
    [1, 5, 0, 2, 6, 1, 3, 2, 4, 3, 0, 0, 1]
])
degree = torch.tensor([3, 3, 2, 2, 2, -1, -1], dtype=torch.float)
cycle5_two_branch_01 = p.subgraph.from_edge_index(
   cycle5_two_branch_01.float(), degrees=degree)

#cycle5_two_branch_02
cycle5_two_branch_02 = torch.tensor([
    [0, 0, 1, 1, 2, 2, 2, 3, 3, 4, 4, 5, 6],
    [1, 5, 0, 2, 1, 3, 6, 2, 4, 3, 0, 0, 2]
])
degree = torch.tensor([3, 2, 3, 2, 2, -1, -1], dtype=torch.float)
cycle5_two_branch_02 = p.subgraph.from_edge_index(
   cycle5_two_branch_02.float(), degrees=degree)

#cycle5_three_branch_012
cycle5_three_branch_012 = torch.tensor([
    [0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 5],
    [1, 5, 0, 2, 6, 1, 3, 7, 2, 4, 3, 0, 0]
])
```

```
degree = torch.tensor([3, 3, 3, 2, 2, -1, -1, -1], dtype=torch.float)
cycle5_three_branch_012 = p.subgraph.from_edge_index(cycle5_three_branch_012.float(), degrees=degree)

#cycle5_three_branch_013
cycle5_three_branch_013 = torch.tensor([
    [0, 0, 1, 1, 1, 2, 2, 3, 3, 3, 4, 4, 5],
    [1, 5, 0, 2, 6, 1, 3, 2, 4, 7, 3, 0, 0]
])
degree = torch.tensor([3, 3, 2, 3, 2, -1, -1, -1], dtype=torch.float)
cycle5_three_branch_013 = p.subgraph.from_edge_index(cycle5_three_branch_013.float(), degrees=degree)


#---------------cycle6-----------

#define and find 6 cycles with different branches
cycle6_one_branch = torch.tensor([[0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6],
        [1, 5, 6, 0, 2, 1, 3, 2, 4, 3, 5, 0, 4, 0]])
degree = torch.tensor([3,2,2,2,2,2,-1],dtype=torch.float)
cycle6_one_branch = p.subgraph.from_edge_index(cycle6_one_branch.float(),degrees=degree)

#cycle6_two_branch_00
cycle6_two_branch_00 = torch.tensor([
    [0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 7],
    [1, 5, 6, 0, 2, 1, 3, 2, 4, 3, 5, 0, 4, 0, 0]])
# degree = torch.tensor([4,2,-1,2,2,2,-1,-1],dtype=torch.float)
degree = torch.tensor([4,-1,-1,-1,-1,-1,-1,-1],dtype=torch.float) # include all template s.t. one vertex has two branches
cycle6_two_branch_00 = p.subgraph.from_edge_index(cycle6_two_branch_00.float(),degrees=degree)

#cycle6_two_branch_01
cycle6_two_branch_01 = torch.tensor([
     [0, 0, 0, 1, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 7],
    [1, 5, 6, 0, 2, 7, 1, 3, 2, 4, 3, 5, 0, 4, 0, 1]])
degree = torch.tensor([3,3,2,2,2,2,-1,-1],dtype=torch.float)
cycle6_two_branch_01 = p.subgraph.from_edge_index(cycle6_two_branch_01.float(),degrees=degree)

#cycle6_two_branch_14
cycle6_two_branch_14 = torch.tensor([
    [0, 0, 1, 1, 1, 2, 2, 3, 3, 4, 4, 4, 5, 5, 6, 7],
    [1, 5, 0, 2, 6, 1, 3, 2, 4, 3, 5, 7, 0, 4, 1, 4]])
degree = torch.tensor([2,3,2,2,3,2,-1,-1],dtype=torch.float)
cycle6_two_branch_14 = p.subgraph.from_edge_index(cycle6_two_branch_14.float(),degrees=degree)
```

```
83
84  #cycle6_two_branch_13
85  cycle6_two_branch_13 = torch.tensor([
86      [0, 0, 1, 1, 1, 2, 2, 3, 3, 3, 4, 4, 5, 5, 6, 7],
87      [1, 5, 0, 2, 6, 1, 3, 2, 4, 7, 3, 5, 0, 4, 1, 3]])
88  degree = torch.tensor([2,3,2,3,2,2,-1,-1],dtype=torch.float)
89  cycle6_two_branch_13 = p.subgraph.from_edge_index(
        cycle6_two_branch_13.float(),degrees=degree)
90
91  #cycle6_three_branch_012
92  cycle6_three_branch_012 = torch.tensor([
93      [0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 5, 5, 6, 7, 8],
94      [1, 5, 6, 0, 2, 7, 1, 3, 8, 2, 4, 3, 5, 0, 4, 0, 1, 2]
95  ])
96  degree = torch.tensor([3, 3, 3, 2, 2, 2, -1, -1, -1], dtype=torch.
        float)
97  cycle6_three_branch_012 = p.subgraph.from_edge_index(
        cycle6_three_branch_012.float(), degrees=degree)
98
99  #cycle6_three_branch_013
100 cycle6_three_branch_013 = torch.tensor([
101     [0, 0, 0, 1, 1, 1, 2, 2, 3, 3, 3, 4, 4, 5, 5, 6, 7, 8],
102     [1, 5, 6, 0, 2, 7, 1, 3, 2, 4, 8, 3, 5, 0, 4, 0, 1, 3]
103 ])
104 degree = torch.tensor([3, 3, 2, 3, 2, 2, -1, -1, -1], dtype=torch.
        float)
105 cycle6_three_branch_013 = p.subgraph.from_edge_index(
        cycle6_three_branch_013.float(), degrees=degree)
106
107 #cycle6_three_branch_135
108 cycle6_three_branch_135 = torch.tensor([
109     [0, 0, 1, 1, 1, 2, 2, 3, 3, 3, 4, 4, 5, 5, 5, 6, 7, 8],
110     [1, 5, 0, 2, 6, 1, 3, 2, 4, 7, 3, 5, 0, 4, 8, 1, 3, 5]
111 ])
112 degree = torch.tensor([2, 3, 2, 3, 2, 3, -1, -1, -1], dtype=torch.
        float)
113 cycle6_three_branch_135 = p.subgraph.from_edge_index(
        cycle6_three_branch_135.float(), degrees=degree)
```

Listing 1: Code for branched 5 and 6 Cycles