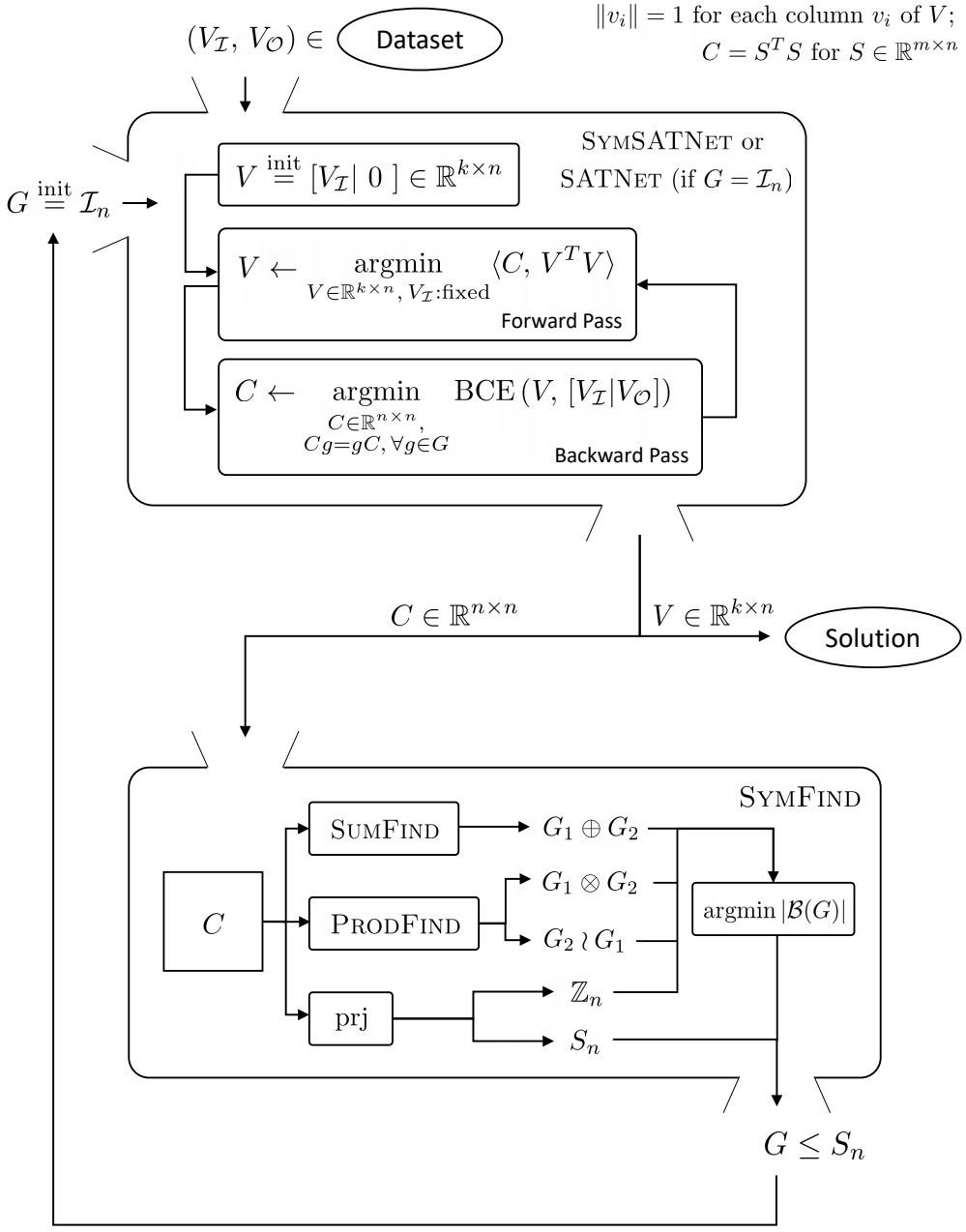


Figure 4: The overall procedure of SATNet, SymSATNet, and SymFind. The original SATNet takes as input the assignment pairs in the dataset, and learns the parameter matrix C which describes the logical rules to solve. Our SymFind algorithm receives the learnt parameter C of SATNet, and uses the subroutine algorithms SumFind and ProdFind and the Reynolds operator prj to find the candidate groups, and returns the strongest group G among the candidates. Finally, SymSATNet exploits the groups symmetries in logical rules provided by domain experts, or automatically detected group G by SymFind.

A Notation

For natural numbers n_1 and n_2 with $n_1 \leq n_2$, we write $[n_1 : n_2]$ for the set $\{n_1, n_1 + 1, \ldots, n_2\}$. For an $m \times m$ matrix M, $1 \leq a \leq b \leq m$, and $1 \leq c \leq d \leq m$, we define $M[a : b, c : d]$ by a $(b - a + 1) \times (d - c + 1)$ matrix M:

$$M[a : b, c : d]_{i,j} = M_{a+i-1,c+j-1} \quad \text{for } i \in [b - a + 1] \text{ and } j \in [d - c + 1].$$

Also, if H is a subgroup of G, we denote it by $H \leq G$.

B Proof of Theorem 2.3

In this section, we prove Theorem 2.3. Here we prove the direct sum, direct product, and wreath product cases by an argument similar to the previous work [27]. For the wreath product case, we slightly generalise the previous results [27], which considered only transitive group actions. We will use the notation from Theorem 2.3. Let G and H be permutation groups over $[p]$ and $[q]$, respectively. Also, let $A \in \mathcal{B}(G)$, $B \in \mathcal{B}(H)$, $O \in \mathcal{O}(G)$, and $O', O'' \in \mathcal{O}(H)$.

Claim B.1. *The matrices $A \oplus \mathbf{0}_q$, $\mathbf{0}_p \oplus B$, $\mathbf{1}_{O \times (p+O')}$, and $\mathbf{1}_{(p+O) \times O'}$ are $(G \oplus H)$-equivariant. Also, the matrices of these types for all possible choices of A, B, O, and O' form a linearly independent set.*

Proof. Pick arbitrary group elements $g \in G$ and $h \in H$. Let P_g and P_h be the permutation matrices corresponding to g and h, respectively.

We prove the claimed equivariance below:

$$\begin{aligned}
(P_g \oplus P_h)(A \oplus \mathbf{0}_p) &= (P_g A) \oplus \mathbf{0}_p = (AP_g) \oplus \mathbf{0}_p = (A \oplus \mathbf{0}_p)(P_g \oplus P_h), \\
(P_g \oplus P_h)(\mathbf{0}_q \oplus B) &= \mathbf{0}_q \oplus (P_h B) = \mathbf{0}_q \oplus (BP_h) = (\mathbf{0}_q \oplus B)(P_g \oplus P_h), \\
(P_g \oplus P_h)\mathbf{1}_{O \times (p+O')} &= \mathbf{1}_{O \times (p+O')} = \mathbf{1}_{O \times (p+O')}(P_g \oplus P_h), \\
(P_g \oplus P_h)\mathbf{1}_{(p+O') \times O} &= \mathbf{1}_{(p+O') \times O} = \mathbf{1}_{(p+O') \times O}(P_g \oplus P_h).
\end{aligned}$$

The third and fourth lines use the fact that $\mathbf{1}_{O \times (p+O')}$ and $\mathbf{1}_{(p+O') \times O}$ are invariant under the left or right multiplication of the permutation matrix $P_g \oplus P_h$. This holds because the orbits of G are preserved by any permutation in G, and those of H are preserved by all permutations in H, so that

$$P_g \mathbf{1}_{O \times O'} = \mathbf{1}_{O \times O'} = \mathbf{1}_{O \times O'} P_h, \qquad P_h \mathbf{1}_{O' \times O} = \mathbf{1}_{O' \times O} = \mathbf{1}_{O' \times O} P_g. \tag{6}$$

Next, we show the claimed linear independence by analysing the indices of the nonzero entries of the four types of matrices in the claim.

1. The matrices of the type $A \oplus \mathbf{0}_q$ for some $A \in \mathcal{B}(G)$ are linearly independent, since their A parts are linearly independent.
2. The matrices of the type $\mathbf{0}_p \oplus B$ for some $B \in \mathcal{B}(H)$ are also linearly independent by similar reason.
3. Different matrices of the type $\mathbf{1}_{O \times (p+O')}$ for some O and O' do not share an index of a nonzero entry, since different orbits of a group are disjoint. Thus, the matrices of this type are linearly independent.
4. Different matrices of the type $\mathbf{1}_{(p+O) \times O'}$ for some O and O' do not share an index of a nonzeron entry. Thus, the matrices of this form are linearly independent.

Also, any matrices of the above four types form a linearly independent set because those linear combinations do not share any indices of nonzero entries. From this and the above reasoning for each of the four types of matrices it follows that the matrices of those four types are linearly independent, as claimed. □

Claim B.2. *The matrix $A \otimes B$ is $(G \otimes H)$-equivariant. Also, the matrices of this shape for all possible choices of A and B form a linearly independent set.*

Proof. Pick arbitrary group elements $g \in G$ and $h \in H$. Let P_g and P_h be the permutation matrices corresponding to g and h, respectively. Then,

$$(P_g \otimes P_h)(A \otimes B) = (P_g A) \otimes (P_h B) = (AP_g) \otimes (BP_h) = (A \otimes B)(P_g \otimes P_h).$$

For linear independence, we can prove it using the fact $\langle A \otimes B,\ A' \otimes B' \rangle = \langle A,\ A' \rangle \cdot \langle B,\ B' \rangle$. □

Claim B.3. *Recall $A \in \mathcal{B}(G)$ and $B \in \mathcal{B}(H)$. The matrices $A \otimes \mathbf{1}_{O' \times O''}$ such that $A_{i,i} = 0$ for $i \in [p]$ and $I_O \otimes B$ are $(H \wr G)$-equivariant. Also, the matrices of these types for all possible choices of A, B, O, O', and O'' form a linearly independent set.*

Proof. Pick arbitrary group elements $g \in G$ and $\vec{h} = (h_1, \ldots, h_p) \in H^p$. Let $P_{g^{-1}}$ and P_{h_i} be the permutation matrices corresponding to g^{-1} and h_i for $i \in [p]$. We can express $\mathrm{wr}(\vec{h}, g)$ and $P_{g^{-1}}$ by

$$\mathrm{wr}(\vec{h}, g) = \sum_{i=1}^{p} \mathbf{1}_{\{i\} \times \{g(i)\}} \otimes P_{h_i}, \qquad P_{g^{-1}} = \sum_{i=1}^{p} \mathbf{1}_{\{i\} \times \{g(i)\}}.$$

Then, we prove the following equivariance:

$$\begin{aligned}
\mathrm{wr}(\vec{h}, g)\,(A \otimes \mathbf{1}_{O' \times O''}) &= \left(\sum_{i=1}^{p} \mathbf{1}_{\{i\} \times \{g(i)\}} \otimes P_{h_i} \right) (A \otimes \mathbf{1}_{O' \times O''}) \\
&= \sum_{i=1}^{p} \left(\mathbf{1}_{\{i\} \times \{g(i)\}} A \otimes P_{h_i} \mathbf{1}_{O' \times O''} \right) \\
&= \left(\sum_{i=1}^{p} \mathbf{1}_{\{i\} \times \{g(i)\}} \right) A \otimes \mathbf{1}_{O' \times O''} \qquad (7) \\
&= P_{g^{-1}} A \otimes \mathbf{1}_{O' \times O''} = A P_{g^{-1}} \otimes \mathbf{1}_{O' \times O''} \\
&= A \left(\sum_{i=1}^{p} \mathbf{1}_{\{i\} \times \{g(i)\}} \right) \otimes \mathbf{1}_{O' \times O''} \\
&= \sum_{i=1}^{p} \left(A\, \mathbf{1}_{\{i\} \times \{g(i)\}} \otimes \mathbf{1}_{O' \times O''} P_{h_i} \right) \qquad (8) \\
&= (A \otimes \mathbf{1}_{O' \times O''}) \left(\sum_{i=1}^{p} \mathbf{1}_{\{i\} \times \{g(i)\}} \otimes P_{h_i} \right) \\
&= (A \otimes \mathbf{1}_{O' \times O''})\, \mathrm{wr}(\vec{h}, g),
\end{aligned}$$

$$\begin{aligned}
\mathrm{wr}(\vec{h}, g)\,(I_O \otimes B) &= \left(\sum_{i=1}^{p} \mathbf{1}_{\{i\} \times \{g(i)\}} \otimes P_{h_i} \right) (I_O \otimes B) \\
&= \sum_{i=1}^{p} \left(\mathbf{1}_{\{i\} \times \{g(i)\}} I_O \otimes P_{h_i} B \right) \\
&= \sum_{i=1}^{p} \left(\mathbf{1}_{\{i\} \times (\{g(i)\} \cap O)} \otimes P_{h_i} B \right) \\
&= \sum_{i=1}^{p} \left(\mathbf{1}_{(\{i\} \cap O) \times \{g(i)\}} \otimes B P_{h_i} \right) \qquad (9) \\
&= \sum_{i=1}^{p} \left(I_O \mathbf{1}_{\{i\} \times \{g(i)\}} \otimes B P_{h_i} \right) \\
&= (I_O \otimes B) \left(\sum_{i=1}^{p} \mathbf{1}_{\{i\} \times \{g(i)\}} \otimes P_{h_i} \right) \\
&= (I_O \otimes B)\, \mathrm{wr}(\vec{h}, g).
\end{aligned}$$

Here, (7) and (8) use the same argument in (6), and (9) uses the fact that $i \in O \iff g(i) \in O$ for any $g \in G$ and $O \in \mathcal{O}(G)$. For linear independence, we again use the fact $\langle A \otimes B,\, A' \otimes B' \rangle = \langle A,\, A' \rangle \cdot \langle B,\, B' \rangle$ to show the orthogonality of all possible matrices of types $A \otimes \mathbf{1}_{O' \times O''}$ and $I_O \otimes B$. □

Claim B.4. *The number of basis elements of $G \oplus H$, $G \otimes H$, and $H \wr G$ can be computed as follows:*

$$\begin{aligned}
|\mathcal{B}(G \oplus H)| &= |\mathcal{B}(G)| + |\mathcal{B}(H)| + 2|\mathcal{O}(G)||\mathcal{O}(H)|, \\
|\mathcal{B}(G \otimes H)| &= |\mathcal{B}(G)||\mathcal{B}(H)|, \\
|\mathcal{B}(H \wr G)| &= |\mathcal{O}(G)||\mathcal{B}(H)| + (|\mathcal{B}(G)| - |\mathcal{O}(G)|)\,|\mathcal{O}(H)|^2.
\end{aligned}$$

Proof. First, we derive some general fact on a finite permutation group. Let $\mathbf{G}$ be a permutation group on $[r]$ for some r. Consider the action of $\mathbf{G}$ on the space of r-dimensional vectors $\mathcal{X} = \mathbb{R}^r$ by the row permutation action $g \cdot v = P_g v$ for any $g \in \mathbf{G}$. Define $\mathcal{F}(\mathbf{G}) = \{v \in \mathcal{X} : g \cdot v = v,\ \forall g \in \mathbf{G}\}$. Consider the following linear operator

$$\phi_{\mathbf{G}} : \mathcal{X} \to \mathcal{X}, \qquad \phi_{\mathbf{G}}(v) = \frac{1}{|\mathbf{G}|} \sum_{g \in \mathbf{G}} g \cdot v,$$

which can also be represented as the following matrix:

$$\phi_{\mathbf{G}} = \frac{1}{|\mathbf{G}|} \sum_{g \in \mathbf{G}} P_g.$$

The operator $\phi_{\mathbf{G}}$ is a projection map, since

$$\begin{aligned}
\phi_{\mathbf{G}}(\phi_{\mathbf{G}}(v)) &= \frac{1}{|\mathbf{G}|^2} \sum_{g_1 \in \mathbf{G}} \sum_{g_2 \in \mathbf{G}} g_1 \cdot (g_2 \cdot v) \\
&= \frac{1}{|\mathbf{G}|^2} \sum_{g_1 \in \mathbf{G}} \sum_{g \in \mathbf{G}} g \cdot v \\
&= \frac{1}{|\mathbf{G}|} \sum_{g \in G} g \cdot v.
\end{aligned}$$

Also, we have $\mathrm{im}(\phi_{\mathbf{G}}) = \mathcal{F}(\mathbf{G})$ where $\mathrm{im}(f)$ is the image of the function f. Now, by noting that a linear projection map has only eigenvalues 0 and 1, the dimension of $\mathcal{F}(\mathbf{G})$ can be computed by the sum of eigenvalues of $\phi_{\mathbf{G}}$, i.e., $\mathrm{tr}(\phi_{\mathbf{G}})$. Also, using Burnside's lemma, we can count the orbits of $\mathbf{G}$ (acting on $[r]$) by

$$\begin{aligned}
|\mathcal{O}(\mathbf{G})| &= \frac{1}{|\mathbf{G}|} \sum_{g \in \mathbf{G}} \mathrm{tr}(P_g) \\
&= \mathrm{tr}\left(\frac{1}{|\mathbf{G}|} \sum_{g \in \mathbf{G}} P_g \right) \\
&= \mathrm{tr}(\phi_{\mathbf{G}}).
\end{aligned}$$

Putting together, we get

$$\dim \mathcal{F}(\mathbf{G}) = \mathrm{tr}(\phi_{\mathbf{G}}) = |\mathcal{O}(\mathbf{G})|. \tag{10}$$

Next, we instantiate what we have just shown above for the following case that $\mathbf{G}$ is the following group:

$$\mathbf{G}_0^{\otimes 2} = \{g \otimes g : g \in \mathbf{G}_0\}$$

for some permutation group $\mathbf{G}_0$ on $[n]$. Note that $\mathbf{G}$ is a permutation group on $[n^2]$. As explained above, $\mathbf{G}_0^{\otimes 2}$ can act on the space of n^2-dimensional vectors $\mathbb{R}^{n^2}$. By vectorizing matrices, we can express the space $\mathcal{E}(\mathbf{G}_0)$ of $\mathbf{G}_0$-equivariant linear maps on $\mathbb{R}^n$ by

$$\begin{aligned}
\mathrm{vec}(\mathcal{E}(\mathbf{G}_0)) &= \left\{\mathrm{vec}(M) : P_g M P_g^T = M,\ \forall g \in \mathbf{G}_0\right\} \\
&= \{\mathrm{vec}(M) : (P_g \otimes P_g)\,\mathrm{vec}(M) = \mathrm{vec}(M),\ \forall g \in \mathbf{G}_0\} \\
&= \{\mathrm{vec}(M) : (g \otimes g) \cdot \mathrm{vec}(M) = \mathrm{vec}(M),\ \forall g \in \mathbf{G}_0\} \\
&= \mathcal{F}(\mathbf{G}_0^{\otimes 2}).
\end{aligned}$$

Thus, $|\mathcal{B}(\mathbf{G}_0)| = \dim \operatorname{vec}(\mathcal{E}(\mathbf{G}_0)) = \dim \mathcal{F}(\mathbf{G}_0^{\otimes 2})$. We now calculate the dimension of $\mathcal{F}(\mathbf{G}_0^{\otimes 2})$ using the relationship in (10):

$$\begin{aligned}|\mathcal{B}(\mathbf{G}_0)| &= \dim \mathcal{F}(\mathbf{G}_0^{\otimes 2}) = \operatorname{tr}(\phi_{\mathbf{G}_0^{\otimes 2}}) \\ &= \frac{1}{|\mathbf{G}_0^{\otimes 2}|} \sum_{g\otimes g \in \mathbf{G}_0^{\otimes 2}} \operatorname{tr}(P_{g\otimes g}) = \frac{1}{|\mathbf{G}_0|} \sum_{g\in \mathbf{G}_0} \operatorname{tr}(P_g)^2. \end{aligned} \tag{11}$$

Finally, we complete the proof by calculating (11) for $\mathbf{G}_0 = G \oplus H$, $\mathbf{G}_0 = G \otimes H$, and $\mathbf{G}_0 = H \wr G$:

$$\begin{aligned}|\mathcal{B}(G \oplus H)| &= \frac{1}{|G \oplus H|} \sum_{g\oplus h \in G \oplus H} \operatorname{tr}(P_{g\oplus h})^2 \\ &= \frac{1}{|G||H|} \sum_{g\in G} \sum_{h\in H} (\operatorname{tr}(P_g) + \operatorname{tr}(P_h))^2 \\ &= \frac{1}{|G|} \sum_{g\in G} \operatorname{tr}(P_g)^2 + \frac{1}{|H|} \sum_{h\in H} \operatorname{tr}(P_h)^2 + \frac{2}{|G||H|} \left(\sum_{g\in G} \sum_{h\in H} \operatorname{tr}(P_g) \operatorname{tr}(P_h) \right) \\ &= |\mathcal{B}(G)| + |\mathcal{B}(H)| + 2|\mathcal{O}(G)||\mathcal{O}(H)|,\end{aligned}$$

$$\begin{aligned}|\mathcal{B}(G \otimes H)| &= \frac{1}{|G \otimes H|} \sum_{g\otimes h \in G \otimes H} \operatorname{tr}(P_{g\otimes h})^2 \\ &= \frac{1}{|G||H|} \sum_{g\in G} \sum_{h\in H} \operatorname{tr}(P_g)^2 \operatorname{tr}(P_h)^2 \\ &= |\mathcal{B}(G)||\mathcal{B}(H)|,\end{aligned}$$

$$\begin{aligned}|\mathcal{B}(H \wr G)| &= \frac{1}{|H \wr G|} \sum_{\operatorname{wr}(\vec{h},g) \in H \wr G} \operatorname{tr}(\operatorname{wr}(\vec{h}, g))^2 \\ &= \frac{1}{|G||H|^p} \sum_{g\in G, (h_1,\ldots,h_p)\in H^p} \operatorname{tr}\left(\sum_{i=1}^p \mathbf{1}_{\{i\}\times\{g(i)\}} \otimes P_{h_i} \right)^2 \\ &= \frac{1}{|G||H|^p} \sum_{g\in G, (h_1,\ldots,h_p)\in H^p} \left(\sum_{i=1}^p \operatorname{tr}\left(\mathbf{1}_{\{i\}\times\{g(i)\}} \otimes P_{h_i} \right) \right)^2 \\ &= \frac{1}{|G||H|^p} \sum_{g\in G, (h_1,\ldots,h_p)\in H^p} \left(\sum_{i=1}^p \mathbb{1}_{\{i=g(i)\}} \operatorname{tr}(P_{h_i}) \right)^2 \\ &= \frac{1}{|G||H|^p} \sum_{g\in G, (h_1,\ldots,h_p)\in H^p} \left(\sum_{i=1}^p \mathbb{1}_{\{i=g(i)\}} \operatorname{tr}(P_{h_i})^2 \right. \\ &\qquad \left. + \sum_{i\neq j} \mathbb{1}_{\{i=g(i), j=g(j)\}} \operatorname{tr}(P_{h_i}) \operatorname{tr}(P_{h_j}) \right) \\ &= \frac{1}{|G||H|^p} \sum_{g\in G} \left\{ \sum_{i=1}^p \mathbb{1}_{\{i=g(i)\}} \left(\sum_{(h_1,\ldots,h_p)\in H^p} \operatorname{tr}(P_{h_i})^2 \right) \right. \\ &\qquad \left. + \sum_{i\neq j} \mathbb{1}_{\{i=g(i), j=g(j)\}} \left(\sum_{(h_1,\ldots,h_p)\in H^p} \operatorname{tr}(P_{h_i}) \operatorname{tr}(P_{h_j}) \right) \right\}\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{|G||H|^p} \sum_{g \in G} \left\{ \left(\sum_{i=1}^{p} \mathbb{1}_{\{i=g(i)\}} \right) |H|^p |\mathcal{B}(H)| \right. \\
&\qquad \left. + \left(\sum_{i \neq j} \mathbb{1}_{\{i=g(i), j=g(j)\}} \right) |H|^p |\mathcal{O}(H)|^2 \right\} \\
&= \frac{1}{|G|} \sum_{g \in G} \left\{ \left(\sum_{i=1}^{p} \mathbb{1}_{\{i=g(i)\}} \right) |\mathcal{B}(H)| \right. \\
&\qquad \left. + \left(\sum_{i=1}^{p} \mathbb{1}_{\{i=g(i)\}} \sum_{j=1}^{p} \mathbb{1}_{\{j=g(j)\}} - \sum_{i=1}^{p} \mathbb{1}_{\{i=g(i)\}} \right) |\mathcal{O}(H)|^2 \right\} \\
&= \frac{1}{|G|} \sum_{g \in G} \left(\mathrm{tr}(P_g) |\mathcal{B}(H)| + \left(\mathrm{tr}(P_g)^2 - \mathrm{tr}(P_g) \right) |\mathcal{O}(H)|^2 \right) \\
&= |\mathcal{O}(G)||\mathcal{B}(H)| + \left(|\mathcal{B}(G)| - |\mathcal{O}(G)| \right) |\mathcal{O}(H)|^2 .
\end{aligned}
$$

□

Proof of Theorem 2.3 Claims B.1, B.2, and B.3 show that each set of matrices on the right-hand side of the equations in Theorem 2.3 consists of linearly independent matrices, and it is contained in the corresponding space of equivariant linear maps. Furthermore, Claim B.4 shows that the matrices in the set span the whole of the space of equivariant linear maps, since their number coincides with the dimension of the space. Hence, these three claims complete the proof of the theorem.

C Proofs of Theorem 3.2 and Lemma 3.3

Proof of Theorem 3.2 Assume that C is G-equivariant. Pick any $g \in G$. Then, C preserves the standard action of g on $\mathbb{R}^n$ via its permutation matrix. Thus, $Cg = gC$, where g denotes the permutation matrix P_g. This equation is equivalent to

$$
g^T C g = C, \tag{12}
$$

since $g^T = g^{-1}$. Using this equality, we show that the optimisation objective is invariant with respect to g's action:

$$
\begin{aligned}
\langle C, (Vg^{-1})^T (Vg^{-1}) \rangle &= \langle C, (g^{-1})^T (V^T V) g^{-1} \rangle \\
&= \mathrm{tr}(C^T g (V^T V) g^T)) \\
&= \mathrm{tr}((g^T C^T g)(V^T V)) \\
&= \mathrm{tr}((g^T C g)^T (V^T V)) \\
&= \langle g^T C g, V^T V \rangle \\
&= \langle C, V^T V \rangle .
\end{aligned}
$$

Here tr is the usual trace operator on matrices, the third equality uses the cyclic property of tr, and the last equality uses (12).

We move on to the proof of the converse. Assume $k = n$ and the equation (3) holds for every $V \in \mathbb{R}^{k \times n}$ and $g \in G$. Pick any $g \in G$. Then, for all $V \in \mathbb{R}^{k \times n}$,

$$
\langle C, V^T V \rangle = \langle C, (Vg^{-1})^T (Vg^{-1}) \rangle = \langle g^T C g, V^T V \rangle .
$$

The first equality follows from (3), and the second from our derivation above. But the space of matrices of the form $V^T V$ is precisely that of $n \times n$ symmetric positive semi-definite matrices, which is known to contain $n(n+1)/2$ independent elements. Since $n(n+1)/2$ is precisely the dimension of the space of $n \times n$ symmetric matrices, the above equality for every $V \in R^{k \times n}$ implies that $C = g^T C g$, that is, $Cg = gC$. This gives the G-equivariance of C, as desired.

Proof of Lemma 3.3 Pick arbitrary $V \in R^{k\times n}$ and $g \in G$. Let $v_1, \ldots, v_n$ be the columns of V in order, and $v'_1, \ldots, v'_n$ be those of Vg^{-1} in order. Then, $\|v_i\| = 1$ for every $i \in [n]$ if and only if $\|v_{g^{-1}(i)}\| = 1$ for all $i \in [n]$. The latter is equivalent to the statement that $\|v'_i\| = 1$ for every $i \in [n]$.

D Subroutines of SYMFIND

In this section, we describe the subroutines SUMFIND and PRODFIND of our SYMFIND algorithm.

D.1 Detection of direct sum

We recall the distributivity law of group constructors [13, 17]:

$$G \otimes (H \oplus H') = (G \otimes H) \oplus (G \otimes H'), \qquad (G \oplus G') \otimes H = (G \otimes H) \oplus (G' \otimes H),$$
$$G \wr (H \oplus H') = (G \wr H) \oplus (G \wr H'), \qquad (G \oplus G') \wr H = (G \wr H) \oplus (G' \wr H).$$

By this law and the fact that $\mathcal{I}_m = \bigoplus_{i=1}^{m} \mathbb{Z}_1$ for all m, any permutation group G definable in our grammar can be expressed as a direct sum

$$G = \bigoplus_{i=1}^{N} H_{(i,1)} \diamond \cdots \diamond H_{(i,n_i)}. \tag{13}$$

where each $H_{(i,j)} \in \{\mathbb{Z}_{m(i,j)}, \mathcal{S}_{m(i,j)}\}$ and $\diamond$ is either $\otimes$ or $\wr$. Let

$$H_i = H_{(i,1)} \diamond \cdots \diamond H_{(i,n_i)},$$

and p_i be the natural number such that H_i is a permutation group on $[p_i]$. Since both $\mathbb{Z}_m$ and $\mathcal{S}_m$ have only one orbit for all m, each H_i also has only one orbit. Hence, by Theorem 2.3,

$$\mathcal{B}(G) = \bigcup_{i,j\in[N]} B_{ij},$$

where

$$B_{ii} = \{\mathbf{0}_{p_1+\cdots+p_{i-1}} \oplus A \oplus \mathbf{0}_{p_{i+1}+\cdots+p_N} : A \in \mathcal{B}(H_i)\} \text{ for } i \in [N], \text{ and}$$
$$B_{ij} = \{\mathbf{1}_{(p_1+\cdots+p_{i-1}+[p_i])\times(p_1+\cdots+p_{j-1}+[p_j])}\} \text{ for } i, j \in [N] \text{ with } i \neq j.$$

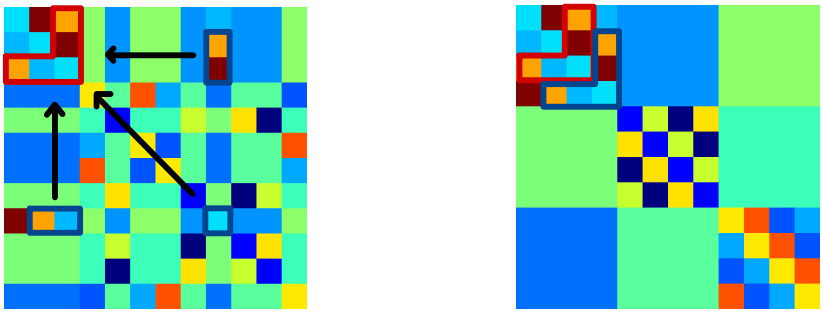
This means that all off-diagonal blocks of any G-equivariant matrices are constant matrices, just like the matrix shown in (b) of Figure 5.

Given a matrix $M \in \mathbb{R}^{m\times m}$, the subroutine SUMFIND aims at finding a permutation $\sigma : [m] \to [m]$ and the split $m = p_1 + \ldots + p_N$ for $p_1, \ldots, p_N > 0$ such that $P_\sigma^T M P_\sigma$ is approximately G-equivariant for some permutation group G on $[m]$ and this group G has the form in (13) where $H_i = H_{(i,1)} \diamond \cdots \diamond H_{(i,n_i)}$ is a permutation group on $[p_i]$. The result of the subroutine is (G, σ).

SUMFIND achieves its aim using two key observations. The first is an important implication of the G-equivariance condition on $M_{\sigma^{-1}} = P_\sigma^T M P_\sigma$ that we mentioned above: when the split $m = p_1 + \ldots + p_N$ is used to group entries of $M_{\sigma^{-1}}$ into blocks, the G-equivalence of $M_{\sigma^{-1}}$ implies that the off-diagonal blocks of $M_{\sigma^{-1}}$ are constant matrices. This implication suggests one approach: for every permutation σ on $[m]$, construct the matrix $M_{\sigma^{-1}}$, and find a split $m = p_1 + \ldots + p_N$ for some N such that off-diagonal blocks of $M_{\sigma^{-1}}$ from this split are constant matrices. Note that this approach is not a feasible option in practice, though, since there are exponentially many permutations σ. The second observation suggests a way to overcome this exponential blow-up issue of the approach. It is that when $M_{\sigma^{-1}}$ is G-equivariant, in many cases its diagonal blocks are cyclic in the sense that the (i, j)-th entry of a block is the same as the $(i + 1, j + 1)$-th entry of the block. This pattern can be used to search for a good permutation σ efficiently.

SUMFIND works as follows. It first finds a permutation $\sigma : [m] \to [m]$ and a split $m = p_1 + \ldots + p_N$ using the process that we will explain shortly. Figure 5 illustrates the input matrix M, and its permuted $M_{\sigma^{-1}}$ that has nine blocks induced by the split of $m = p_1 + p_2 + p_3$. Then, SUMFIND calls SYMFIND recursively on each diagonal block of $M_{\sigma^{-1}}$ and gets, for every $i \in [N]$,

$$(H_i, \sigma_i) = \text{SYMFIND}(M_{\sigma^{-1}}[p_{i-1} + 1 : p_i, p_{i-1} + 1 : p_i])$$



(a) Original matrix M (b) Permuted matrix $M_{\sigma^{-1}} = P_\sigma^T M P_\sigma$

Figure 5: Block pattern which commonly appears in the equivariant matrices under a direct-sum group. In this example, $M_{\sigma^{-1}} \in \mathcal{E}(\mathbb{Z}_4 \oplus \mathbb{Z}_4 \oplus \mathbb{Z}_4)$. The red and blue "L-shaped" clusters have the same values.

where $p_0 = 0$. Finally, SUMFIND returns

$$\left(\bigoplus_{i=1}^{N} H_i,\ \sigma \circ \left(\bigoplus_{i=1}^{N} \sigma_i \right) \right)$$

where

$$\sigma_i \oplus \sigma_j : [p_i + p_j] \to [p_i + p_j],$$
$$(\sigma_i \oplus \sigma_j)(a) = \begin{cases} \sigma_i(a) & \text{if } a \in [p_i], \\ \sigma_j(a - p_i) + p_i & \text{otherwise.} \end{cases}$$

We now explain the first part of SUMFIND that finds a permutation σ and a split $m = p_1 + \ldots + p_N$. For simplicity, we ignore the issue of noise, and present a simpler version that uses equality instead of approximate equality (i.e., being close enough). We start by initialising $\sigma = \sigma_I$, the identity permutation, and $M_{\sigma^{-1}} = M$. Then, we pick an index k_1 from the set $K = \{k_1 \in [2 : m] : (M_{\sigma^{-1}})_{k_1,k_1} = (M_{\sigma^{-1}})_{1,1}\}$. Then, we locate $(M_{\sigma^{-1}})_{1,k_1}$ in the $(1, 2)$-th entry by swapping the indices 2 and k_1, which can be done by updating

$$M_{\sigma^{-1}} \leftarrow P_{\sigma_{(2,k_1)}} M_{\sigma^{-1}} P^T_{\sigma_{(2,k_1)}},$$
$$\sigma \leftarrow \sigma \circ \sigma^{-1}_{(2,k_1)}.$$

Here $\sigma_{(a,b)}(a) = b$, $\sigma_{(a,b)}(b) = a$, and $\sigma_{(a,b)}(c) = c$ for $c \neq a, b$. Next, we find a new index $k_2 \in [3 : m]$ which is suitable to swap with the index 3 in the updated $M_{\sigma^{-1}}$. As Figure 5 shows, we need to find k_2 such that $(M_{\sigma^{-1}})_{2,k_2} = (M_{\sigma^{-1}})_{1,2}$, $(M_{\sigma^{-1}})_{k_2,k_2} = (M_{\sigma^{-1}})_{2,2}$, and $(M_{\sigma^{-1}})_{k_2,2} = (M_{\sigma^{-1}})_{2,1}$. Once we find such k_2, we swap the indices 3 and k_2 by updating

$$M_{\sigma^{-1}} \leftarrow P_{\sigma_{(3,k_2)}} M_{\sigma^{-1}} P^T_{\sigma_{(3,k_2)}},$$
$$\sigma \leftarrow \sigma \circ \sigma^{-1}_{(3,k_2)}.$$

We repeat this process to find an index $k_l \in [l + 1 : m]$ and the "L-shaped" entries which preserve the cyclic pattern. If we cannot find such k_l, we stop the process, and check (i) whether all rows in $(M_{\sigma^{-1}})[1 : l, l + 1 : m]$ are the same and also (ii) whether all columns in $(M_{\sigma^{-1}})[l + 1 : n, 1 : l]$ are the same. If the answers for both (i) and (ii) are yes, we move on to find the next diagonal block in $M_{\sigma^{-1}}$ in the same manner.

If (i) or (ii) has a negative answer, we go back to the step right before choosing k_1 from K and resetting σ and M to the values at that step. Then, we repeat the above process with a new choice of k_1 from K. If no choice of $k_1 \in K$ leads to the situation where both (i) and (ii) have positive answers, then we conclude that $p_1 = 1$, and move on to find the next diagonal block of M starting from the index 2.

D.2 Detection of direct product

Let M be a $m \times m$ matrix. Assume that we are given $p, q \in \mathbb{N}$ with $pq = m$. If for some permutation groups $H \leq \mathcal{S}_p$ and $K \leq \mathcal{S}_q$, the matrix M lies in $\mathcal{E}(H \otimes K)$ (i.e., M is $H \otimes K$-equivariant), then M can be represented as a linear combination of Kronecker products:

$$M = \sum_{i=1}^{\gamma} (X_i \otimes Y_i) \tag{14}$$

where $X_i \in \mathcal{E}(H)$, $Y_i \in \mathcal{E}(K)$, and γ is a natural number.

The representation suggests the following strategy of finding a group G of symmetries of M that is the direct product of two permutation groups on $[p]$ and $[q]$. First, we express M as a linear combination of Kronecker products $X_i \otimes Y_i$ for $X_i \in \mathbb{R}^{p \times p}$ and $Y_i \in \mathbb{R}^{q \times q}$. Second, we pick a pair X_i and Y_i in the linear combination. Third, we call SYMFIND recursively on X_i and Y_i to get group-permutation pairs (H, σ_H) and (K, σ_K). Finally, we return $(H \otimes K, \sigma_H \otimes \sigma_K)$ where

$$\sigma_H \otimes \sigma_K : [m] \to [m],$$
$$(\sigma_H \otimes \sigma_K)((a-1)q + b) = (\sigma_H(a) - 1)q + \sigma_K(b) \quad \text{for all } a \in [p] \text{ and } b \in [q].$$

Our PRODFIND is the implementation of the strategy just described. The only non-trivial steps of the strategy are the first two, namely, to find the representation of M in (14), and to pick good X_i and Y_i. PRODFIND also has to account for the fact that the representation in (14) holds only approximately at best in our context.

For the representation finding, PRODFIND uses the technique [24] that solves the following optimisation problem for given γ:

$$\operatorname*{argmin}_{X_i \in \mathbb{R}^{p \times p}, Y_i \in \mathbb{R}^{q \times q}} \left\| M - \sum_{i=1}^{\gamma} (X_i \otimes Y_i) \right\|_F . \tag{15}$$

The technique performs the singular value decomposition of the rearranged matrix $\hat{M} \in \mathbb{R}^{p^2 \times q^2}$ of M defined by

$$\hat{M}_{(i'-1)p+i,(j'-1)q+j} = M_{(i-1)q+j,(i'-1)q+j'} \tag{16}$$

for $i, i' \in [p]$ and $j, j' \in [q]$. To understand why it does so, note that the optimisation problem in (15) is equivalent to the following problem:

$$\operatorname*{argmin}_{X_i \in \mathbb{R}^{p \times p}, Y_i \in \mathbb{R}^{q \times q}} \left\| \hat{M} - \sum_{i=1}^{\gamma} \operatorname{vec}(X_i) \operatorname{vec}(Y_i)^T \right\|_F . \tag{17}$$

This new optimisation problem can be solved using SVD. Concretely, if we have the SVD of M, namely,

$$\hat{M} = \sum_{i=1}^{\operatorname{rank}(\hat{M})} s_i (u_i v_i^T) \tag{18}$$

where $\operatorname{rank}(\hat{M})$ is the rank of $\hat{M}$, s_i is the i-th largest singular value, and u_i and v_i are the corresponding left and right singular vectors, we can solve the minimisation problem (17) by reshaping each term of (18) corresponding to the γ largest singular values, i.e.,

$$\hat{M} \approx \sum_{i=1}^{\gamma} \operatorname{vec}(X_i) \operatorname{vec}(Y_i)^T \quad \text{where } \operatorname{vec}(X_i) = \sqrt{s_i} u_i \text{ and } \operatorname{vec}(Y_i) = \sqrt{s_i} v_i \text{ for every } i \in [\gamma].$$

The first step of PRODFIND is to apply the above technique [24]. PRODFIND performs the SVD of the matrix $\hat{M}$ and gets s_i, u_i, v_i in (18) where $s_1 \geq s_2 \geq \ldots \geq s_l$ for some l. Then, it sets $L = s_1/5$, and picks γ so that all i's with $s_i \geq L$ are included when we approximate $\hat{M}$.

The second step of PRODFIND is to pick X_i and Y_i. PRODFIND simply picks X_1 and Y_1 that correspond to matrices (reshaped from u_1 and v_1) for the largest singular value s_1. We empirically

observed that the matrices X_1 and Y_1 are most informative about the symmetries of M, and are least polluted by noise from the training of SATNet.

The third and fourth steps of PRODFIND are precisely the last two steps of the strategy that we described above.

Our description of SYMFIND in the main text says that PRODFIND gets called for all the divisors p of n with q being n/p, since we do not know the best divisor p a priori, and that each invocation generates a new candidate in the candidate list $\mathcal{A}$. However, in practice, we ignore some divisors that violates our predefined criteria. In particular, we set an upper bound U on γ, and use a divisor p only when PRODFIND for p picks γ with $\gamma \leq U$, i.e., not too many terms are considered in the linear combination of Kronecker products. This criterion can be justified by the following lemma:

Lemma D.1. *Let $G = H \otimes K$ be the direct product of permutation groups H and K on $[p]$ and $[q]$, respectively. Consider a G-equivariant matrix $M \in \mathcal{E}(G)$ and its rearranged version $\hat{M}$ in* (16)*. Then,* $\operatorname{rank}(\hat{M}) \leq \min\left(|\mathcal{B}(H)|,\, |\mathcal{B}(K)|\right)$.

Proof. Let $M \in \mathcal{E}(G)$. The matrix M can be expressed as a linear combination of the basis elements in $\mathcal{B}(G)$:

$$
\begin{aligned}
M &= \sum_{i=1}^{|\mathcal{B}(H)|} \sum_{j=1}^{|\mathcal{B}(K)|} \alpha_{ij} (A_i \otimes B_j) \\
&= \sum_{j=1}^{|\mathcal{B}(K)|} \left(\sum_{i=1}^{|\mathcal{B}(H)|} \alpha_{ij} A_i \right) \otimes B_j \qquad (19) \\
&= \sum_{i=1}^{|\mathcal{B}(H)|} A_i \otimes \left(\sum_{j=1}^{|\mathcal{B}(K)|} \alpha_{ij} B_j \right) \qquad (20)
\end{aligned}
$$

where $A_i \in \mathcal{B}(H)$ and $B_j \in \mathcal{B}(K)$. We can also rewrite (19) and (20) with equations for $\hat{M}$:

$$
\hat{M} = \sum_{j=1}^{|\mathcal{B}(K)|} \operatorname{vec}(A'_j) \operatorname{vec}(B_j)^T = \sum_{i=1}^{|\mathcal{B}(H)|} \operatorname{vec}(A_i) \operatorname{vec}(B'_i)^T
$$

$$
\text{where } A'_j = \left(\sum_{i=1}^{|\mathcal{B}(H)|} \alpha_{ij} A_i \right) \text{ and } B'_i = \left(\sum_{j=1}^{|\mathcal{B}(K)|} \alpha_{ij} B_j \right).
$$

Since the summands $\operatorname{vec}(A'_j) \operatorname{vec}(B_j)^T$ and $\operatorname{vec}(A_i) \operatorname{vec}(B'_i)^T$ are rank-1 matrices, by the subadditivity of rank, i.e., $\operatorname{rank}(X + Y) \leq \operatorname{rank}(X) + \operatorname{rank}(Y)$, we have $\operatorname{rank}(\hat{M}) \leq \min(|\mathcal{B}(H)|,\, |\mathcal{B}(K)|)$. □

Having fewer basis elements is generally better because it leads to a small number of parameters to learn in SymSATNet. The above lemma says that a divisor p with large γ (which roughly corresponds to the large rank of $\hat{M}$ in the lemma) leads to large $|\mathcal{B}(H)|$ and $|\mathcal{B}(K)|$. Our criterion is designed to avoid such undesirable cases.

D.3 Detection of wreath product

Assume that the permutation groups G and H act transitively on $[p]$ and $[q]$ (i.e., for all $i, j \in [p]$ and $i', j' \in [q]$, there are $g \in G$ and $h \in H$ such that $g(i) = j$ and $h(i') = j'$). We recall that in this case, every equivariant matrix M under the wreath product of groups $H \wr G$ can be expressed as follows [27]:

$$
M = A \otimes \mathbf{1}_q + I_p \otimes B \qquad (21)
$$

for some $A \in \mathcal{E}(G)$ and $B \in \mathcal{E}(H)$, where $\mathbf{1}_m$, I_m are everywhere-one, identity matrices in $\mathbb{R}^{m\times m}$. We use this general form of $\mathcal{B}(H \wr G)$-equivariant matrices and make PRODFIND detect the case that symmetries are captured by a wreath product.

Our change in PRODFIND is based on a simple observation that the form in (21) is exactly the one in (14) with $\gamma = 2$. We change PRODFIND such that if we get $\gamma = 2$ while runnning PRODFIND(M, p), we check whether M can be written as the form in (21). This checking is as follows. Using given p and q, we create blocks

$$M^{(i,j)} = M[(i-1)\times q + 1 : i \times q, (j-1)\times q + 1 : j \times q]$$

of $q \times q$ submatrices in M for all $i, j \in [p]$. Then, we test whether all $M^{(i,i)} - M^{(j,j)}$ and $M^{(i,j)}$ for $i \neq j$ are constant matrices (i.e., matrices of the form $\alpha \mathbf{1}_q$ for some $\alpha \in \mathbb{R}$). If this test passes, M has the desired form. In that case, we compute A and B from M, and recursively call SYMFIND on A and B.

We can also make SUMFIND(M) detect wreath product. The required change is to perform a similar test on each diagonal block of the final $M_{\sigma^{-1}}$ computed by SUMFIND(M). The only difference is that when the test fails, we look for a rearrangement of the rows and columns of the diagonal block that makes the test succeed. If the test on a diagonal block succeeds (after an appropriate rearrangement), the group corresponding to this block, which is one summand of a direct sum, becomes a wreath product.

E Computational Complexity of SYMFIND

In this section, we analyse the computational complexity of our SYMFIND algorithm in terms of the dimension of the input matrix. Suppose that an $n \times n$ matrix is given to SymFind as an input. The complexity of SymFind shown in Algorithm 1 is $O(n^{3+\epsilon})$ for any arbitrarily small $\epsilon > 0$.

We can show the claimed complexity of SYMFIND using induction. When SumFind is called in the line 9 of Algorithm 1, it first spends $O(n^3)$ time for clustering, and then makes recursive calls to SymFind with $n_i \times n_i$ submatrices for $\sum n_i = n$. Each of these calls takes $O(n_i^{3+\epsilon})$ time by induction, and the total cost of all the calls is $\sum_i O(n_i^{3+\epsilon}) = O(n^{3+\epsilon})$. Also, when ProdFind is called in the lines 11-12 of Algorithm 1, for each divisor p of n, ProdFind performs SVD in $O(n^3)$ steps, and makes recursive calls to SymFind. The recursive calls here together cost $O(p^{3+\epsilon/2} + (n/p)^{3+\epsilon/2}) = O(n^{3+\epsilon/2})$ by induction. Thus, the loop in lines 11-14 costs $O(n^{3+\epsilon/2}d(n)) = O(n^{3+\epsilon})$, where $d(n)$ is the number of divisors of n, since $d(n) = O(n^{\epsilon/2})$ for any arbitrarily small $\epsilon > 0$. It remains to show that applying the Reynolds operator (in the lines 2 and 6) and counting the number of basis elements (in the line 15) take $O(n^{3+\epsilon})$ steps. The former costs $O(n^2)$ since it is actually an average pooling operation when we consider permutation groups. The latter costs at most $O(n)$ (when it is given $\mathcal{B}(\bigoplus_{i=1}^{n} \mathbb{Z}_1)$).

F Symmetries in Sudoku and Rubik's Cube Problems

In this section, we describe the group symmetries in Sudoku and the completion problem of Rubik's cube. In both problems, we can find certain group $G \leq \mathcal{S}_n$ acting on $\mathbb{R}^{k\times n}$ such that for any valid assignment $V \in \mathbb{R}^{k\times n}$ of problem, $g \cdot V$ is also valid for any $g \in G$.

In 9×9 Sudoku, we denote the first three rows of a Sudoku board by the first band, and the next three rows by the second band, and the last three rows by the third band. In the same manner, we denote each three columns by a stack. For Sudoku problem, one can observe that any row permutations within each band preserve the validity of the solutions. Also, the permutations of bands preserve the validity. We can represent this type of hierarchical actions by wreath product groups. In this case, 3 bands are permuted in a higher level, and 3 rows in each band are permuted in a lower level, which corresponds to the group action of $\mathcal{S}_3 \wr \mathcal{S}_3$. The same process can be applied to the stacks and the columns, which also corresponds to the group $\mathcal{S}_3 \wr \mathcal{S}_3$. Furthermore, the permutations of number occurences in Sudoku (e.g., switching all the occurences of 3's and 9's) also preserves the validity of the solutions. In this case, any permutations over $[9]$ are allowed, thus $\mathcal{S}_9$ represents this permutation action. Overall, we have three permutation groups, $G_1 = \mathcal{S}_3 \wr \mathcal{S}_3$, $G_2 = \mathcal{S}_3 \wr \mathcal{S}_3$, and $G_3 = \mathcal{S}_9$. These three groups are in different levels; G_1 is at the outermost level, and G_3 is at the innermost level.

Algorithm 2 Forward pass of SATNet

Input: $V_{\mathcal{I}}$
init random unit vectors v_o, $\forall o \in \mathcal{O}$
compute $\Omega = VS^T$
repeat
  for $o \in \mathcal{O}$ **do**
    compute $g_o = \Omega s_o - \|s_o\|^2 v_o$
    compute $v_o = -g_o/\|g_o\|$
    update $\Omega = \Omega + (v_o - v_o^{\text{prev}})s_o^T$
  end for
until not converged
Output: $V_{\mathcal{O}}$

Algorithm 3 Forward pass of SymSATNet

Input: $V_{\mathcal{I}}$
init random unit vectors v_o, $\forall o \in \mathcal{O}$
repeat
  for $o \in \mathcal{O}$ **do**
    compute $g_o = Vc_o - C_{o,o}v_o$
    compute $v_o = -g_o/\|g_o\|$
    update $V = V + (v_o - v_o^{\text{prev}})\mathbb{1}_o^T$
  end for
until not converged
Output: $V_{\mathcal{O}}$

Note that these three types of actions are commutative, i.e., the order does not matter if we apply the actions in different levels. This relation forms a direct product between each level, and we conclude $G = (\mathcal{S}_3 \wr \mathcal{S}_3) \otimes (\mathcal{S}_3 \wr \mathcal{S}_3) \otimes \mathcal{S}_9$.

For Rubik's cube, we first introduce some of the conventional notations. We are given a $3 \times 3 \times 3$ Rubik's cube composed of 6 faces and 9 facelets in each face. There are also 26 cubies in the Rubik's cube, and they are classified into 8 corners, 12 edges, and 6 centers. To change the color state of Rubik's cube, we can move the cubies with 9 types of rotations, which are denoted by U, D, F, B, L, R, M, E, S. We can represent each color state of the Rubik's cube by a function from $F = [6] \times [9]$ to $C = [6]$, where F encodes the face and facelets, and C encodes the colors. The initial state of Rubik's cube is $s : F \to C$ satisfying $s(i, j) = i$ for all $(i, j) \in F$. If there exists a sequence of rotations that transforms s to the initial state, then s is solvable. Our completion problem of Rubik's cube is to find a color assignment such that the assignment is solvable. For example, if a corner cubie contains 2 same colors, it is not solvable because these adjacent same colors cannot be separated by the 9 types of rotations.

We can observe the following group symmetries in the completion problem of Rubik's cube: any solvable color state is still solvable after being transformed by the 9 types of rotations. These 9 rotations generate a permutation group $\mathcal{R}_{54}$ acting on the 6×9 facelets of a Rubik's cube, which is called the Rubik's cube group. Furthermore, like Sudoku, we can also consider the permutations of color occurences. In this case, if we assume the colors 0 and 5 are initially on the opposite sides, then the solvability can be broken by switching the colors 0 and 1 since the colors 0 and 5 are then no longer on the opposite sides. Considering this, computing all possible permutations acting on the color occurences is not trivial. Instead, we can generate such group $\mathcal{R}_6$ by 3 elements, each of which corresponds to the 90° rotation in one of the three axes in 3-dimension. Finally, these two groups $\mathcal{R}_{54}$ and $\mathcal{R}_6$ form different levels of actions, and actions from different levels commute each other. Therefore, we conclude $G = \mathcal{R}_{54} \otimes \mathcal{R}_6$.

G Efficient Implementation of SymSATNet

Our SymSATNet includes forward-pass and backward-pass algorithms as described in Section 3. Here, we describe slight changes of the forward-pass and backward-pass algorithms in SymSATNet, and the improvement of efficiency obtained by these changes. Algorithms 2 and 3 are the forward pass algorithms of SATNet and SymSATNet, and Algorithms 4 and 5 are the backward pass algorithms of SATNet and SymSATNet. In those algorithms, we let c_o be the o-th column of the matrix C, and v_o^{prev} and u_o^{prev} be the previous o-th columns of V and U before the update. Also, let $P_o = I_k - v_o v_o^T$, and $\mathbb{1}_o$ be the n-dimensional one-hot vector whose only o-th element is 1. Note that our algorithms are only slightly different from the ones of original SATNet, and the only difference is that the inner products $s_i^T s_j$ are substituted by $C_{i,j}$ which can be directly derived from $C = S^T S$. This allows us to implement the forward-pass and the backward-pass algorithm of SymSATNet in the same manner as the original SATNet.

The above changes bring a small difference in the computational complexity. In SATNet, new matrices Ω and Ψ are required for the rank-1 update in each loop. Recall that $V, U \in \mathbb{R}^{k \times n}$ and

Algorithm 4 Backward pass of SATNet

1: **Input:** $\{\partial \ell / \partial v_o : o \in \mathcal{O}\}$
2: **init** $U_{\mathcal{O}} = 0$ and $\Psi = U_{\mathcal{O}} S_{\mathcal{O}}^T = 0$
3: **repeat**
4: **for** $o \in \mathcal{O}$ **do**
5: **compute** $dg_o = \Psi s_o - \|s_o\|^2 u_o - \partial \ell / \partial v_o$
6: **compute** $u_o = -P_o dg_o / \|g_o\|$
7: **update** $\Psi = \Psi + (u_o - u_o^{\text{prev}}) s_o^T$
8: **end for**
9: **until** not converged
10: **Output:** $U_{\mathcal{O}}$

Algorithm 5 Backward pass of SymSATNet

1: **Input:** $\{\partial \ell / \partial v_o : o \in \mathcal{O}\}$
2: **init** $U_{\mathcal{O}} = 0$
3: **repeat**
4: **for** $o \in \mathcal{O}$ **do**
5: **compute** $dg_o = U c_o - C_{o,o} u_o - \partial \ell / \partial v_o$
6: **compute** $u_o = -P_o dg_o / \|g_o\|$
7: **update** $U = U + (u_o - u_o^{\text{prev}}) \mathbb{1}_o^T$
8: **end for**
9: **until** not converged
10: **Output:** $U_{\mathcal{O}}$

$S \in \mathbb{R}^{m \times n}$ imply $\Omega, \Psi \in \mathbb{R}^{k \times m}$. SATNet costs $O(nmk)$ twice every iteration, in the lines 6, 8 of Algorithm 2 and in the lines 5, 7 of Algorithm 4. SymSATNet costs $O(n^2 k)$ once every iteration, in the line 5 of Algorithm 3 and in the line 5 of Algorithm 5. Another slight difference is in the first line of the body of each for loop, where SATNet computes $\|s_o\|^2$, but SymSATNet uses the constant $C_{o,o}$. If we denote the required number of iterations by t, then SATNet costs $O(nmk \cdot 2t)$, and SymSATNet costs $O(n^2 k \cdot t)$ inside the loop. These make one of the major differences between the runtime of SATNet and SymSATNet, since these are scaled by t, the number of iterations until convergence. Outside the loop, SATNet additionally costs $O(nmk)$ in the line 3 of Algorithm 2, and SymSATNet costs $O(n^2 d)$ in the forward computation of C in (4), and in the backward computation with the chain rule in (5). These costs are not significant because each of these occurs only once every gradient update in SATNet and SymSATNet.

We also inspected the values of t in SATNet and SymSATNet in a training run for Sudoku and Rubik's cube problem in the configuration we used in our experiments. For Sudoku, SATNet repeated the loop $t = 21.39$ times on average, in the range of $15 \leq t \leq 32$, while SymSATNet finished the loop in $t = 19.10$ on average, in the scope of $3 \leq t \leq 26$. For Rubik's cube, SATNet completes the loop in $t = 9.25$ on average, whose scope was $3 \leq t \leq 25$, while SymSATNet iterates $t = 7.30$ times on average, with the range $2 \leq t \leq 26$. These results show that the component t is reduced in SymSATNet for those two problems, and bring further improvement of efficiency in practice.

H Hyperparameters

In this section, we specify the hyperparameters to run SymFind algorithm and the validation steps in SymSATNet-Auto. Each validation step requires a threshold of improvement of validation accuracy to measure the usefulness of each smaller part in a given group, i.e., only the smaller parts showing improvement greater than the threshold were considered to be useful, and were combined to construct the whole group. We determined this threshold by the number of corruption in the dataset; 0.05 for the dataset with 0 and 1 corruption, 0.15 for 2 corruption, and 0.2 for 3 corruption. Also, our SymFind algorithm receives λ as an input, which determines the degree of sensitivity of SymFind. In practice, this tolerance was implemented by two hyperparameters, say λ_1 and λ_2. λ_1 was used to compute a threshold to decide whether a pair of entries have the same value within our tolerance so that they have to be clustered. We computed this threshold by the multiplication of λ_1 and the norm of input matrix, and λ_1 was fixed by 0.1 for both problems. λ_2 played a role of a threshold to conclude that the input matrix truly has the symmetries under the discovered group by SymFind. λ_2 was used in the last step of SymFind, where the largest group is selected among the candidates. With the Reynolds operator, we computed the distance between the input matrix and the equivariant space under each candidate group, and filtered out the groups whose distance is greater than λ_2. We determined λ_2 by the number of corruption in the dataset; 0.4 for the dataset with 0 corruption, 0.5 for 1 corruption, 0.55 for 2 corruption, and 0.6 for 3 corruption.

I GPU Usage

The implementation of SymSATNet is based on the original SATNet code, and thus the calculations in the forward-pass and the backward-pass algorithms of SymSATNet can be accelerated by GPU. We used GeForce RTX 2080 Ti for every running of SATNet and SymSATNet.

J Ablation Study of Validation Steps

In this section, we present the results of ablation study for the additional validation step in SymSATNet-Auto. All the configurations are the same, except that the discovered group symmetries by SYMFIND are directly exploited by SymSATNet without the subsequent validation step. We used noisy Sudoku and Rubik's cube datasets, and repeated our experiment with each dataset ten times. Here we report the average test accuracies with 95% confidence interval.

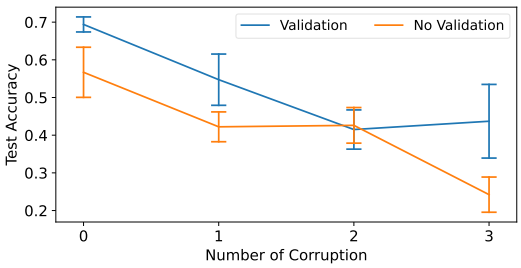


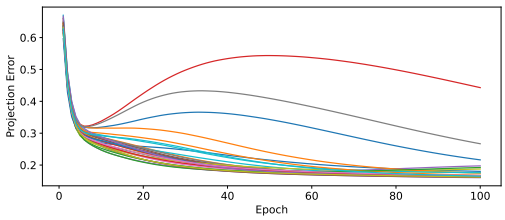
Figure 6: Best test accuracies of SymSATNet-Auto with or without the validation step using noisy Rubik's cube datasets.

In Sudoku, our SYMFIND algorithm always succeeded in finding the full group symmetries, and these correct symmetries were preserved by the validation step since they always significantly improved the validation accuracies after projecting the parameter C of SATNet. For this reason, omitting the validation step did not change the performance of SymSATNet-Auto. Thus, we do not plot the results for the Sudoku case. For Rubik's cube, the results are shown in Figure 6. Without the validation step, SymSATNet-Auto often overfit into the wrong group symmetries induced by the noise in the parameter matrix C of SATNet, resulting in lower improvement of performance. These results show that the additional validation step is useful not only when the training and the test examples share the same distribution, but also when the distribution of training examples is the perturbation of that of test examples by noise.

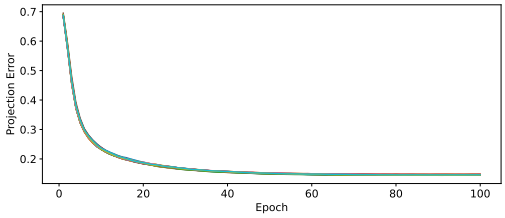
K Emergence of Symmetries in SATNet

In this section, we report our experiment for detecting the appearance and disappearance of symmetries in the SATNet's parameter matrix C throughout the training epochs. We measured how close C is to $\mathcal{E}(G)$, the space of equivariant matrices under G, by computing $\|\mathrm{prj}(G, C) - C\|_F$ (denoted by *projection error* below) where G is the ground-truth permutation group for a given learning problem (e.g., Sudoku or Rubik's cube). We evaluated the projection errors throughout 100 training epochs. Our experiment for each problem was repeated 30 times.

Figure 7 shows the results. For Sudoku, in multiple cases out of 30 trials, the projection error hit the lowest point around the 5-10th epochs, and after these epochs, the projection error started to increase until certain epochs, so that the training ended with a high projection error. We did not observe any clear difference in the training loss (or the test loss) between these cases with high projection errors and the other cases (which showed low projection errors). This result suggests possible overfitting from the perspective of symmetry discovery in the original SATNet. For Rubik's cube,



(a) Sudoku



(b) Rubik's cube

Figure 7: Projection errors of the SATNet's parameter C during 100 epochs. We repeated 30 training trials for each problem to report the projection errors.

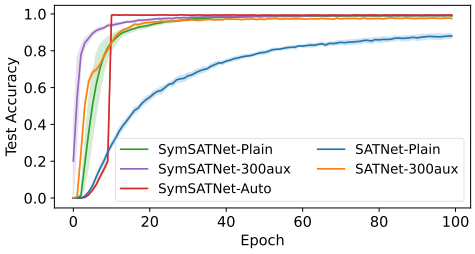
Table 3: Best train and test accuracies during 100 epochs and average total train times (10^2 sec). Each experiment was repeated 10 times and its average and 95% confidence interval are reported. Additional times for automatic symmetry detection (SYMFIND and validation) are reported after $+$.

MODEL	SUDOKU			CUBE		
	TRAIN ACC.	TEST ACC.	TIME	TRAIN ACC.	TEST ACC.	TIME
SATNET-PLAIN	93.5% ±1.2%	88.1% ±1.3%	48.0 ±0.17	99.4% ±0.1%	55.7% ±0.5%	1.8 ±0.01
SATNET-300AUX	99.8% ±0.0%	97.9% ±0.2%	90.3 ±0.68	99.8% ±0.0%	56.5% ±0.6%	14.0 ±0.12
SYMSATNET-PLAIN	98.8% ±0.1%	99.2% ±0.2%	25.6 ±0.14	67.1% ±0.2%	66.9% ±0.9%	1.1 ±0.00
SYMSATNET-300AUX	99.2% ±0.1%	99.3% ±0.1%	53.6 ±0.56	69.6% ±0.4%	67.6% ±0.5%	9.5 ±0.29
SYMSATNET-AUTO	99.3% ±0.1%	99.5% ±0.2%	22.7 +0.14 ±0.35	70.2% ±3.1%	68.1% ±2.0%	3.4 +0.66 ±0.19

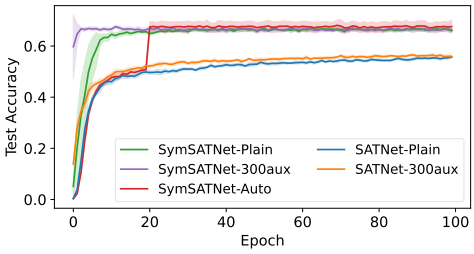
in all of our trials, the projection error always decreased throughout the epochs, and no sign of overfitting was detected. Answering why this is the case is an interesting topic for future research. Also, in the very beginning of the training for both problems, the projection errors were not sufficiently small. By choosing proper stopping epochs in SymSATNet-Auto, we can avoid high projection errors, as we did (i.e., we picked the 10th epoch for Sudoku, and the 20th epoch for Rubik's cube). Finally, we point out that there may be factors other than the projection error that influence the performance of our symmetry-discovery algorithm. If found, those factors would become useful tools for analysing the theoretical guarantees of SYMFIND algorithm, and we leave it to future work.

L SymSATNet with Auxiliary Variables

By introducing auxiliary variables, the original SATNet can achieve higher expressive power and better performance while trading-off the runtime efficiency. The similar process can be done in SymSATNet. If a permutation group G captures group symmetries of a problem to be solved without auxiliary variables, then $G \oplus \mathcal{I}_m$ represents the extension of the same symmetries with the m auxiliary variables. The singleton group $\mathcal{I}_m$ here acts trivially (i.e., permutes nothing) on those auxiliary variables.



(a) Sudoku



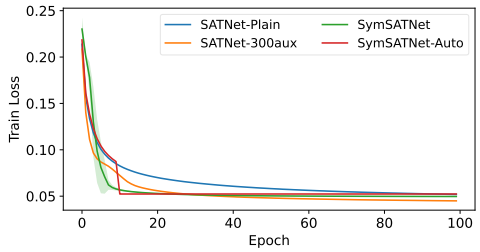
(b) Rubik's cube

Figure 8: Test accuracies throughout the first 100 training epochs. Each training run was repeated 10 times to compute the average and the 95% confidence interval.

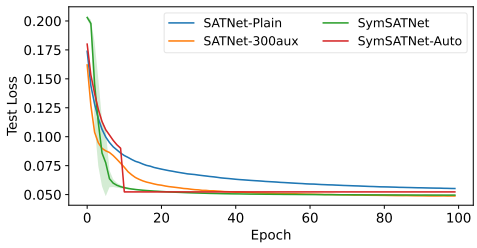
We now report the findings of our experiments with SymSATNet with 300 auxiliary variables (SymSATNet-300aux) that retains the above extension of group symmetries. We compared the performance of SymSATNet-300aux with the other four models that we used before, on the 0-corrupted Sudoku and Rubik's cube datasets. We denote SymSATNet without auxiliary variables by SymSATNet-Plain to distinguish it from SymSATNet-300aux. We repeated the training for each dataset 10 times. Here we report the average and best test accuracies, the average training times, and 95% confidence interval. We trained SymSATNet-300aux with the learning rate $\eta = 4 \times 10^{-2}$, which is the same as SymSATNet-Plain. All the experimental setups were the same as before.

Figure 8 and Table 3 show the results. In both tasks, SymSATNet-300aux outperformed all the other models except SymSATNet-Auto. Also, for Rubik's cube, SymSATNet-300aux showed the fastest convergence in epoch among all the models. Note that both the test accuracies and the training times of SymSATNet-300aux were remarkably improved over those of SATNet baselines. These results demonstrate that SATNet with auxiliary variables still enjoyed the benefits by exploiting a minor extension of group symmetries, although the benefits were not as significant as in the case of SymSATNet-Plain.

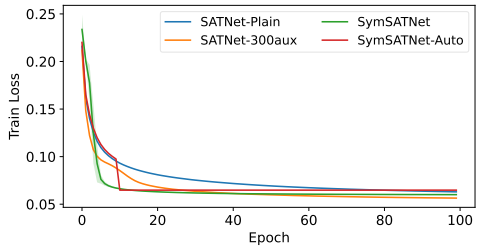
M Loss Curves



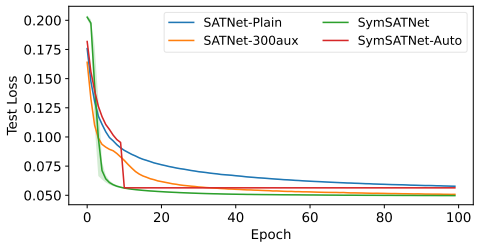
(a) Training losses in Sudoku with 0 corruption



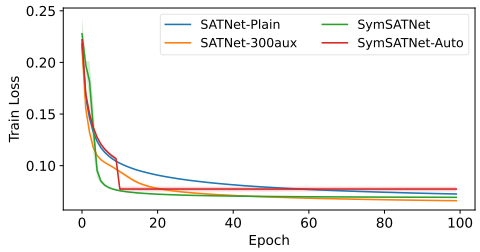
(b) Test losses in Sudoku with 0 corruption



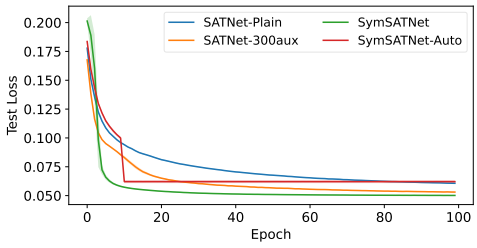
(c) Training losses in Sudoku with 1 corruption



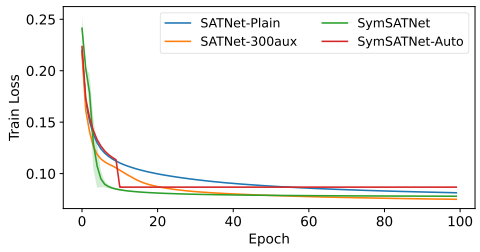
(d) Test losses in Sudoku with 1 corruption



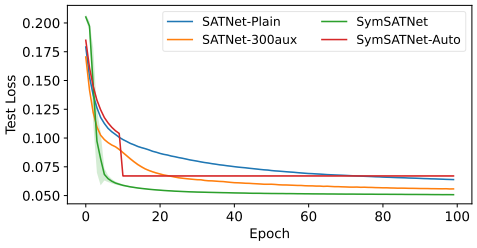
(e) Training losses in Sudoku with 2 corruptions



(f) Test losses in Sudoku with 2 corruptions

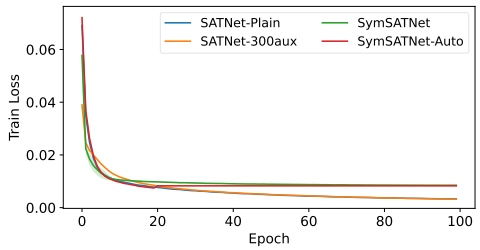


(g) Training losses in Sudoku with 3 corruptions

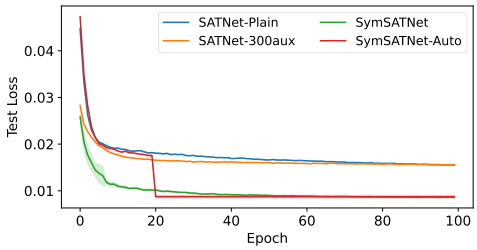


(h) Test losses in Sudoku with 3 corruptions

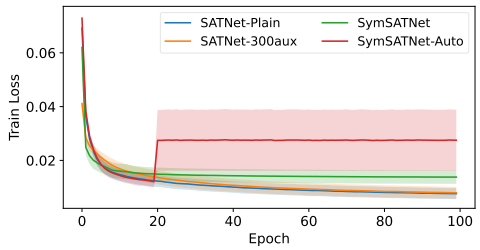
Figure 9: Training and test loss curves for Sudoku. Each loss is averaged over 10 trials and each 95% confidence interval is included.



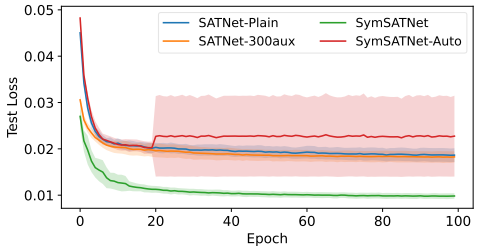
(a) Training losses in Rubik's cube with 0 corruption



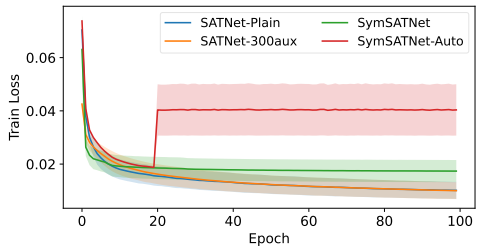
(b) Test losses in Rubik's cube with 0 corruption



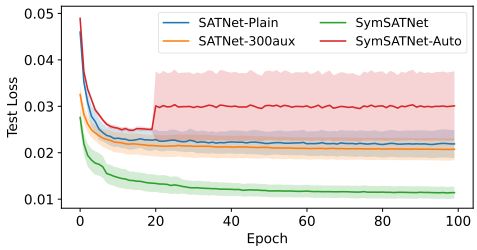
(c) Training losses in Rubik's cube with 1 corruption



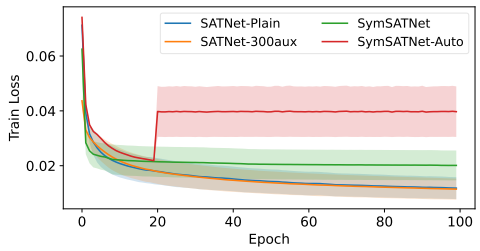
(d) Test losses in Rubik's cube with 1 corruption



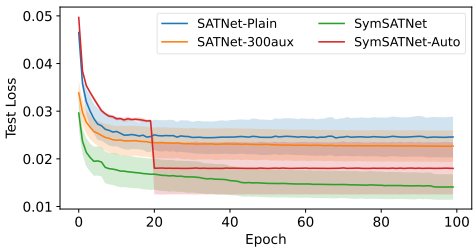
(e) Training losses in Rubik's cube with 2 corruptions



(f) Test losses in Rubik's cube with 2 corruptions



(g) Training losses in Rubik's cube with 3 corruptions



(h) Test losses in Rubik's cube with 3 corruptions

Figure 10: Training and test loss curves for the Rubik's cube problem. Each loss is averaged over 10 trials and each 95% confidence interval is included.