Fourier Bases for Solving Permutation Puzzles

Horace Pan
Department of Computer Science
University of Chicago
hopan@uchicago.edu

Risi Kondor
Department of Computer Science, Statistics
University of Chicago
Flatiron Institute
risi@uchicago.edu

Abstract

Traditionally, permutation puzzles such as the Rubik's Cube were often solved by heuristic search like A^*-search and value based reinforcement learning methods. Both heuristic search and Q-learning approaches to solving these puzzles can be reduced to learning a heuristic/value function to decide what puzzle move to make at each step. We propose learning a value function using the irreducible representations basis (which we will also call the Fourier basis) of the puzzle's underlying group. Classical Fourier analysis on real valued functions tells us we can approximate smooth functions with low frequency basis functions. Similarly, smooth functions on finite groups can be represented by the analogous low frequency Fourier basis functions. We demonstrate the effectiveness of learning a value function in the Fourier basis for solving various permutation puzzles and show that it outperforms standard deep learning methods.

1 Introduction

Most methods to solve permutation puzzles such as the Rubik's Cube, Pyraminx (fig: 2a), and the Sliding Number Tile Puzzle involve heuristic search such as A^*-search and Iterative Deepening A^*-search (IDA) (Korf, 1985). The runtime of these methods depends on how well the heuristic functions used estimate the distance of a given puzzle state to the goal state. The most effective heuristic functions for permutation puzzles like the Rubik's Cube and the Sliding Number Tile Puzzle use pattern databases (Korf and Taylor, 1996; Korf, 1997; Kociemba, n.d.; Felner et al., 2004), which effectively store the distance of various partially solved puzzles to the solution state. Korf (1997) famously solved instances of the Rubik's Cube using a pattern database of all possible corner cube configurations and various edge configurations and has since improved upon the result with an improved hybrid heuristic search (Bu and Korf, 2019).

Machine learning based methods have become popular alternatives to classical planning and search approaches for solving these puzzles. Brunetto and Trunda (2017) treat the problem of solving the Rubik's cube as a supervised learning task by training a feed forward neural network on 10 million cube samples, whose ground truth distances to the goal state were computed by brute force search. Value based reinforcement learning methods to solve the Rubik's Cube and Sliding Number Tile Puzzle (Bischoff et al., 2013) are also common. Most recently, McAleer et al. (2018) and Agostinelli et al. (2019) developed a method to solve the Rubik's cube and other puzzles by training a neural network value function through reinforcement learning . They then used this value network in conjunction with A^* search and Monte Carlo tree search.

In this paper, we focus on learning to solve permutation problems in the reinforcement learning setting, specifically using value functions. The crux of our approach to solving permutation puzzles is to exploit the underlying structure of the puzzle by using specialized basis functions for expressing this value function. There has been a wealth of research in reinforcement learning on constructing basis functions for value function approximation (Keller et al., 2006; Menache et al., 2005). Common choices for a set of basis functions include radial basis functions (Kretchmar and Anderson, 1997) and Fourier series (sinusoids) (Konidaris et al., 2011). Proto Value Functions (PVF) (Mahadevan and Maggioni, 2007, 2006; Mahadevan, 2005) try to solve Markov decision processes (MDP) by representing the

Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130. Copyright 2021 by the author(s).

value function in the eigenbasis of the MDP's graph Laplacian. Their method uses ideas from spectral graph theory which tell us that the smoothest functions on a graph correspond to the lowest eigenvalue eigenvectors of the graph laplacian. Though our work is conceptually similar to Proto Value Functions, computing the eigenvectors of the graph Laplacian of a permutation puzzle would be a prohibitively expensive $O(n^3)$ operation, where n is the size the puzzle. Our approach uses spectral ideas as well, but does not suffer from cubic run-time costs.

What is an appropriate basis for learning functions over our permutation puzzles? It turns out that the puzzles we are interested in solving are groups. Representation theory (Serre, 1977) tells us there is a natural basis, the Fourier Basis, for expressing functions over a given group. Using this Fourier basis, we show that we can learn linear value functions to permutation puzzles such as Pyraminx, and the 2-by-2 Rubik's cube with far fewer samples and orders of magnitude fewer parameters than a neural network parameterization. Our work is the first to demonstrate the efficacy of the Fourier Basis in reinforcement learning and the first to propose leveraging the representations of wreath product groups to solve the 2-by-2 cube.

2 Related Work

Permutation puzzles often exhibit symmetries that can be exploited to shrink their state space. In the traditional planning literature, many have proposed various procedures to handle symmetries in heuristic search. Fox and Long (1999, 2002), and Pochter et al. (2011) try to detect symmetric states during heuristic search to avoid searching states that are isomorphic to previously seen states. Domshlak et al. (2012) tries to detect symmetries in the transition graph of the puzzle to avoid redundant search as well. While our work deals with permutation puzzles that do have symmetries, we do not explicitly model any of their symmetries since our focus is on demonstrating the efficacy of the Fourier basis for value function approximation.

Though they are not commonly known in the machine learning community, the Fourier basis of the symmetric group has been used in machine learning and statistics (Diaconis, 1988) for a variety of applications including: learning rankings (Kondor and Barbosa, 2010; Kondor and Dempsey, 2012; Mania et al., 2018), and performing inference over probability distributions on permutations for object tracking (Kondor et al., 2007; Huang et al., 2009, 2008). These works deal with bandlimited representations of functions over the symmetric group, but do not address reinforcement learning over permutation puzzles.

Our work builds on Swan (2017)'s which used the irreducible representations of the symmetric group specifically for solving the 8-puzzle (the 3-by-3 sliding number tile puzzle). Swan demonstrates that commonly used A^* heuristics for solving the puzzle such as the Hamming Distance, and Manhattan distance, are bandlimited in the Fourier basis. Starting from these heuristics, Swan uses a derivative-free optimization technique to find the optimal Fourier coefficients to minimize the the number nodes explored during A^*-search. Our approach differs from Swan's in that we are learning value functions (which can eventually be used with heuristic search) in the reinforcement setting. We also go further than Swan by demonstrating that reinforcement learning using the Fourier basis works on wreath product groups as well the symmetric group.

3 Reinforcement Learning Background

It is natural to pose the problem of solving permutation puzzles under the reinforcement learning framework. We briefly review the key definitions on Markov Decision Processes. A (deterministic) Markov Decision Process is described by a tuple $\mathbb{M} = (S, A, r, T, \gamma)$. Here S is the state space, A is the action space, $r : S \times A \to \mathbb{R}$ is the reward function, $T : S \times A \to S$ is the transition operator of the process. γ is a discount parameter which determines how much future rewards are valued relative immediate rewards. The goal in reinforcement learning is to find a policy $\pi : S \to A$ that maximizes long term expected reward under actions drawn from π.

Permutation puzzles are deterministic Markov Decision Processes where the state space is the set of accessible puzzle states. The set of actions is the set of legal moves of the puzzle (for example, face twists for the Rubik's Cube). The reward function can be $+1$ for any action that transitions directly to the solved state and -1 otherwise.

3.1 Value Functions

Value based reinforcement learning methods learn a value function $V : S \to \mathbb{R}$ that estimates the long term reward of a given state, or a value function $Q : S \times A \to \mathbb{R}$ that measures the long term value of taking an action at a given state. Given a V or Q function, we naturally get the following greedy policy: $\pi(s) = \text{argmax}_{a\in A} V(T(s, a))$ or $\pi(s) = \text{argmax}_{a\in A} Q(s, a)$

Value iteration (Bellman, 1957) and Q-learning (Watkins and Dayan, 1992) methods iteratively update the current value estimate with one-step look-ahead

$$V(s) \leftarrow \text{argmax}_{a \in A} \left(r(s,a) + \gamma V(T(s,a)) \right)$$
$$Q(s,a) \leftarrow r(s,a) + \text{argmax}_{a' \in A} \gamma Q(T(s,a), a')$$

or by minimizing the squared difference between the two sides of the above update rules over a batch of states.

V or Q are often parameterized in some class of function approximators such as neural networks whose parameters are optimized through backpropagation.

4 Mathematical Preliminaries

Recall that a group G is a set of objects endowed with a multiplication operation $\cdot : G \times G \to G$ which satisfies the three properties:

1. the group multiplication is associative: $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ for all $a, b, c \in G$.
2. there exists an identity element of G, which we denote e, such that for every $g \in G$, $g \cdot e = e \cdot g = g$
3. for every $g \in G$, there exists an inverse element $g^{-1} \in G$.

The state space of a permutation puzzle naturally forms a group, since any legal move can be followed by any other legal move, and also any legal move has an "inverse" which just undoes it.

Definition 1 *The cyclic group of order m, denoted C_m, is the set of integers $\{0, 1, \ldots, m-1\}$ with the group operation being addition modulo m.*

Definition 2 *The Symmetric group of order n, denoted $\mathbb{S}_n$, is the set of permutations on n objects, or equivalently the set of bijections from the set $\{1, 2, \ldots, n\}$ to $\{1, 2, \ldots, n\}$.*

Definition 3 *Given a subset of group elements: S, we denote by $\langle S \rangle$ the subgroup of G* ***generated by*** *S, i.e., the set of all elements of G that can be expressed as a finite product of the elements of S and their inverses. We say G is generated by S if $G = \langle S \rangle$.*

Definition 4 *Given a group G and a set of generators $S \subseteq G$, the* ***Cayley Graph*** *$\Gamma(G, S)$ is a graph whose vertices correspond to elements of G (in a one-to-one fashion) and for any two vertices v_g and v_h, there is an edge between v_g and v_h if there is some $s \in S$ such that $s \cdot g = h$.*

Given a puzzle's underlying group G and legal puzzle moves (i.e., generators) S, solving the puzzle can be recast as finding a path on $\Gamma(G, S)$ from an arbitrary group element $g \in G$ to a solved state d. Without loss of generality, we take the identity element as the solved state.

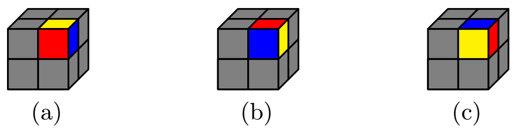


Figure 1: 2-by-2 Cube: three possible orientations of a corner cubie.

Example 1 *The set of legal positions of the Rubik's Cube forms a group. The inverse of any face move is the move twisting the same face in the opposite direction. The solved state is the identity element. Every legal cube position is the result of applying a sequence of* 90 *degree clockwise or counter-clockwise face moves (Front, Back, Left, Right, Up, or Down) to the solved state. Any legal cube state g which is produced by the sequence of face twists: $m_1 m_2 \ldots m_k$, has an inverse: $m_k^{-1} \ldots m_1^{-1}$.*

It turns out that in the case of all the permutation puzzles that we are interested in, the puzzle's group is a so-called **wreath product group**.

Definition 5 *Given n copies of a group G, the wreath product group $G \wr \mathbb{S}_n$ is defined to be the set of elements $(g, h) \in G^n \times \mathbb{S}_n$ with the group multiplication defined as:*

$$(g_1, h_1) \cdot (g_2, h_2) = (g_1 \cdot (h_2 \cdot g_2),\ h_2 h_1),$$

where the action of $h \in \mathbb{S}_n$ on $g \in G^n$ is defined as $[h \cdot g]_i = g_{h^{-1}(i)}$.

The permutation puzzles that we discuss will be wreath product groups where G is a cyclic group.

Example 2 *The group of the 2-by-2 Rubik's cube is a subgroup of the wreath product group: $C_3 \wr \mathbb{S}_8$. There are eight moveable "cubies" and each cubie has three visible colored facets. Each face twist permutes the eight cubies among themselves while cycling through the three possible orientations of a cubie (as shown in Fig 1).*

Example 3 *Pyraminx (Fig: 2a) is a three-layered tetrahedron shaped puzzle similar to the Rubik's Cube. The tips and middle layers of the puzzle can be rotated clockwise or counter-clockwise by $\frac{2\pi}{3}$. There are 4 tips and 6 edge facets that can be moved. The tips can cycle between three possible orientations and be moved independently of the rest of the layers of the puzzle. Similar to the cubies of the 2-by-2 cube, the six edge facets get permuted amongst themselves and cycle between their* *two possible orientations (determined by the two colors). The full group is a subgroup of: $(C_2 \wr \mathbb{S}_6) \times C_3^4$.*

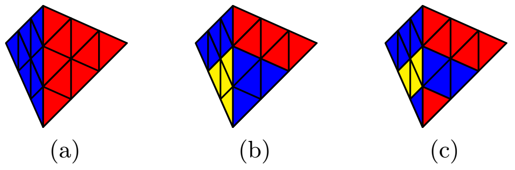


Figure 2: Suppose the bottom face of the Pyraminx puzzle in Fig 2a is yellow. Rotating the puzzle's front two layers clockwise by $\frac{2\pi}{3}$ results in Fig 2b. Subsequently, rotating just the front tip counterclockwise by $\frac{2\pi}{3}$ results in Fig 2c.

4.1 Fourier Analysis on Finite Groups

Classical Fourier analysis gives us the tools to decompose functions on the real line into linear combinations of sinusoidal basis functions. The harmonic analysis of functions on permutation groups is defined using **representations** and similarly gives us the tools to decompose functions on the group into the analogous basis functions. Representations of finite groups have been thoroughly studied in mathematics (Serre, 1977; Diaconis, 1988), so we will simply give a brief overview of the main concepts from the field that underpin our work.

Definition 6 *A **representation** ρ of a group G is a matrix valued function: $\rho : G \to \mathbb{C}^{d_\rho \times d_\rho}$ that preserves the group structure: $\rho(g \cdot h) = \rho(g)\rho(h)$. d_ρ is referred to as the dimensionality of the representation.*

Example 4 *The permutation matrices of size $n \times n$ comprise a representation for the Symmetric group $\mathbb{S}_n$: $[\rho(\sigma)]_{ij} = \mathbb{1}\{\sigma(j) = i\}$ for $\sigma \in \mathbb{S}_n$ and $1 \leq i, j \leq n$*

We say that two representations ρ_1 and ρ_2 are equivalent if there exists some invertible transformation T such that $\rho_1(g) = T\rho_2(g)T^{-1}$ for all $g \in G$. We can express this as $\rho_1 \equiv \rho_2$.

Given two representations $\rho_1 and \rho_2$ of G, we can construct larger representations by taking their direct sum:

$$(\rho_1 \oplus \rho_2)(g) = \left(\begin{array}{c|c} \rho_1(g) & 0 \\ \hline 0 & \rho_2(g) \end{array}\right).$$

We say that a representation ρ is **reducible** if there exists representations ρ_1, ρ_2 such that $\rho \equiv \rho_1 \oplus \rho_2$. A representation is **irreducible** if it is not reducible. For a given finite group G, we can construct infinitely many inequivalent reducible representations by taking any number of direct sums of an arbitrary representation ρ. However, G only has finitely many inequivalent irreducible representations.

Example 5 *The cyclic group C_m has m irreducible representations. They are the complex exponentials: $\chi_k(j) = e^{2\pi ijk/m}$ for $j \in C_m, k \in \{0, 1, ..., m-1\}$ regarded as "1×1 matrices".*

For notational convenience, we will also refer to irreducible representations as "irreps" going forward. The group **Fourier transform** of a function $f : G \to \mathbb{C}$ at a given irreducible representation ρ is defined as:

$$\widehat{f}_\rho = \sum_{g \in G} f(g)\rho(g). \tag{1}$$

The **inverse Fourier transform** is

$$f(g) = \frac{1}{|G|} \sum_{\rho \in \mathcal{R}} \text{Tr}[\widehat{f}_\rho^\top \rho(g)]. \tag{2}$$

One way of looking at the irreducible representations of a finite group is to view each irreducible ρ is a collection of $d_\rho \times d_\rho$ individual functions $\rho_{ij} : G \to \mathbb{C}$, with $\rho_{ij}(g) = [\rho(g)]_{ij}$. Recall the property of traces: $\text{Tr}[A^\top B] = \text{vec}(A)^\top \text{vec}(B)$. The inverse fourier transform of f is just an expansion of f into a linear combination of these ρ_{ij} functions.

Theorem 1 *Serre (1977) Given a complete set of unitary irreducible representations of a finite group G, the set of matrix entries of these irreducible representations form a complete orthonormal basis for $L(G) = \{f : G \to \mathbb{C}\}$, the space of functions on G:*

$$L(G) = \text{span}\{\sqrt{d_\rho}\,\rho_{ij} \mid 1 \leq i, j \leq d_\rho\}.$$

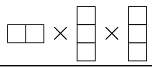
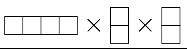
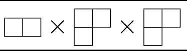
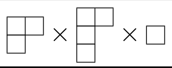
where the inner product on $L(G)$ is defined as:

$$\langle \phi, \psi \rangle_{L(G)} = \frac{1}{|G|} \sum_{g \in G} \phi(g)\psi(g)^*$$

Going forward, we will also refer to the irreducible representation basis as the **Fourier basis**, as is common in existing literature (Huang et al., 2009). Theorem 1 and the definition of the inverse Fourier transform tell us that we can express any function $f : G \to \mathbb{C}$ in fourier space using the matrix entries of the irreducible representations of G as a basis.

There exist well known constructions for the irreps of $\mathbb{S}_n$ such as Young's Orthogonal Representation (YOR). We leave the definition of YOR, which can also be found in Kerber (1971), to the Supplement. The irreducible representations of wreath product groups of the form $C_m \wr \mathbb{S}_n$ are constructed by inducing the irreducible representations of Young Subgroups of $\mathbb{S}_n$ up to $\mathbb{S}_n$ (Ceccherini-Silberstein et al., 2014). In Table 1, we show some examples of the irreps of $C_3 \wr \mathbb{S}_8$, the wreath product group associated with the 2-by-2 cube along

Table 1: Dimensionality of irreps of $C_3 \wr \mathbb{S}_8$

Young Subgroup	Young Subgroup Irrep	Dim
$\mathbb{S}_2 \times \mathbb{S}_3 \times \mathbb{S}_3$		560
$\mathbb{S}_4 \times \mathbb{S}_2 \times \mathbb{S}_2$		420
$\mathbb{S}_2 \times \mathbb{S}_3 \times \mathbb{S}_3$		2240
$\mathbb{S}_3 \times \mathbb{S}_4 \times \mathbb{S}_1$		1680

with their dimensionalities. We direct the reader to our supplement for full details on constructing YOR and the irreps of wreath product groups which follow the prescriptions given by Huang et al. (2009) and Rockmore (1995). For now we just assume that we can evaluate the irreps of the symmetric group and various wreath product groups relatively efficiently.

5 Learning a Value Function in the Fourier Basis

For a permutation puzzle whose states are elements of an underlying group G, we propose learning a value function $V : G \to \mathbb{C}$ in the Fourier basis. Once we pick a subset of irreducible representations of G to use, learning a value function amounts to learning their respective Fourier matrices. Let $\mathcal{R}$ be a complete set of inequivalent irreducible representations of G and $I \subseteq \mathcal{R}$ the subset of irreps that we pick. The value function is then approximated as:

$$V(g) = \sum_{\rho \in I} \mathrm{Tr}[\theta_\rho^\top \rho(g)], \tag{3}$$

where $\theta_\rho \in \mathbb{C}^{d_\rho \times d_\rho}$ for $\rho \in I$ are the learnable Fourier coefficient matrices. Note that the domain of the parameters may be $\mathbb{C}$ instead of $\mathbb{R}$ since some irreps of groups such as the 2-by-2 cube are defined over $\mathbb{C}$ (from its factor of C_3). Equation (3) can also be rewritten as:

$$V(g) = \theta^\top \Big(\bigoplus_{\rho \in I} \mathrm{vec}(\rho(g)) \Big), \tag{4}$$

$$\theta = \bigoplus_{\rho \in I} \mathrm{vec}(\theta_\rho) \in \mathbb{C}^{\sum_{\rho \in I} d_\rho \times d_\rho}. \tag{5}$$

where vec is the linear operator that converts a matrix into a column vector. The scaling factors of $\frac{1}{|G|}$ and d_ρ from the inverse Fourier transform (Eqn. 2) have been collapsed into the θ_ρ terms. Using this parameterization of the value function, we can use standard value based reinforcement learning algorithms such as Q-learning or value iteration to learn V. The number of parameters in V is: $\sum_{\rho \in I} d_\rho^2$.

5.1 Low Rank Fourier Matrices

Depending on the dimensionality of the irreducible representations that we use to parameterize V, it might not be feasible to learn dense θ_ρ matrices. We can circumvent this by parameterizing θ_ρ as a low rank matrix: $\theta_\rho = U_\rho W_\rho^\top$, where $U_\rho \in \mathbb{C}^{d_\rho \times k}$, $W_\rho \in \mathbb{C}^{d_\rho \times k}$, and $k \ll d_\rho$. The number of parameters in V is then: $2k \sum_{\rho \in I} d_\rho$.

5.2 Algorithm

Our algorithm for performing value iteration in Fourier space is Algorithm 1. We follow the general structure of deep-Q learning popularized by Mnih et al. (2013, 2015); Silver et al. (2016). In each iteration of the main loop, following Agostinelli et al. (2019), we sample a random walk starting from the goal state of our permutation puzzle, and store it in our cache/replay buffer. The contents of the cache are overwritten in a FIFO order when the cache is full. We then sample a batch of states S from the cache, compute the argmax neighbor values according to the auxiliary target network $V_{\tilde{\theta}}$ for each sampled state, and minimize the difference between the current value and the one-step look ahead value (line 10).

Algorithm 1: Value Iteration in Fourier Space

```
Result: $V_\theta$
Input: num epochs $T$, set of irreps $I$, batchsize $B$,
       max cache size $C$, target update interval
       $K$, random walk length $l$, discount $\gamma$
1  for $t \leftarrow 1$ to $T$ do
2      Sample a random walk of length $l$:
        $(s_1, s_2, \ldots s_l)$
3      Store $(s_1, s_2, ...., s_l)$ in circular cache
4      Sample a batch of states $S$ from cache
5      $S_t$ = stack [vec($\rho(s)$) for all $s \in S$, $\rho \in I$]
6      $S'$ = stack [vec($\rho(s')$) for all $s'$ adjacent to $s$
        for $s \in S, \rho \in I$]
7      Evaluate $V_\theta(S_t) = \theta^\top S_t$; $V_{\tilde{\theta}}(S') = \tilde{\theta}^\top S'$
8      Let $S_{t+1}$ be the argmax neighbors of each
        $s \in S_t$ from $V_{\tilde{\theta}}(S')$
9      $R_t$ = stack[1 if $s$ is solved $-1$ for all $s \in S_{t+1}$]
10     $L(\theta) = (V_\theta(S_t) - (R_t + \gamma V_{\tilde{\theta}}(S_{t+1})))^2$
11     Update $\theta$ with $\nabla_\theta L(\theta)$
12     if $t$ mod $K == 0$ then
13         $\tilde{\theta} \leftarrow \theta$
14 end
```

6 Experiments

We demonstrate the efficacy of the learning over the Fourier bases of the following three permutation puzzles using Algorithm 1: Pyraminx (Fig 2a), $\mathbb{S}_8$, and the 2-by-2 cube (Fig 1).

- Pyraminx (without the tips): a subgroup of $C_2 \wr S_6$ with 11520 states, and a diameter of 9. The tips can trivially be moved to the correct position, so we choose to ignore them.
- An $\mathbb{S}_8$ puzzle with 40320 states and a diameter of 9. The generators of this puzzle are the $\mathbb{S}_8$ permutations associated with the six valid face moves of the 2-by-2 cube.
- 2-by-2 Rubik's Cube: a subgroup of $C_3 \wr \mathbb{S}_8$ with 3.67×10^6 states, and a diameter of 14. Note that we are using a symmetrized version of the 2-by-2 cube, by modding out the 24 rotational symmetries of the cube (the full 2-by-2 cube has 88 million states).

The goal of our experiments is to show that we can learn effective value functions in the Fourier basis , where effectiveness is measured according to:

1. how often does our learned value function give a locally optimal move at a given state? When evaluating V_θ over all neighbors of a given state, is the argmax neighbor in fact closer to the goal state?
2. if we use the value network in a greedy/best first search manner, how often does this policy reach the goal state?
3. how well does the value function work when used in heuristic search?

We compare our models, which we will refer to as Fourier or irrep models going forward, against deep value networks (DVN) and an optimal solver, which simply uses the ground truth shortest path distance as the value function. These three puzzles are small enough that we can use Djikstra's algorithm to compute the shortest path distance of each state to the goal state which is also used to compute the proportion of locally optimal moves. The DVN uses onehot encoding representations of the puzzles while the Fourier models use the Fourier basis representations.

Baseline DVN For Pyraminx and the $\mathbb{S}_8$ puzzle, we parameterize the DVN as a multilayer perceptron with one hidden layer and rectified linear unit nonlinearities; the hidden layer sizes were 1024 and 2048 respectively.

For the 2-by-2 cube, the DVN architecture follows a similar architecture to the one used by Agostinelli et al. (2019): five fully connected layers with a rectified linear unit nonlinearity placed after every non-output layer, and a skip connection between the output of the 2nd and 4th fully connected layers. The layers have the following input and output dimensions: (88, 1024), (1024, 2048), (2048, 2048), (2048, 2048), (2048, 1), where 88 is the size of the one-hot encoding of a 2-by-2 cube state. For all puzzles, we decided the number of layers, hidden layer sizes, and residual layers by doing grid search. Similar to Agostinelli et al. (2019), we found that increasing the number of layers did not improve training while wider hidden layers did.

We trained the Pyraminx, $\mathbb{S}_8$, and 2-by-2 cube models for a maximum of 60k, 100k, and 300k epochs. For the 2-by-2 cube experiments, we terminated training for irrep models if they showed no improvement in solves or locally optimal moves within the last 10k epochs.

Hyperparameters During training, we sample a random walk of length 15 for Pyraminx, the $\mathbb{S}_8$ puzzles, and a random walk of length 25 for the 2-by-2 cube. All model parameters were optimized using Adam (Kingma and Ba, 2014). For all the Fourier models, we used a learning rate of 0.005 and a minibatch size of 128 for the two smaller puzzles and 32 for the 2-by-2 cube. For the DVN, we used a learning rate of 0.003 and a minibatch size of 128. The initial weights of the the DVN were drawn from a normal distribution with mean 0 and standard deviation of $0.03, 0.05$ or 0.1. The capacity of the circular cache was $100,000$ and the target network was updated every 100 epochs. In our experience, most of these hyperparameters (except the model architecture of the DVNs) can be slightly modified without changing the performance of the models too much. We picked the various hyperparameters by randomized grid search: after training DVNs for 10k for the smaller puzzles or 50k epochs for the 2-by-2 cube over various parameter settings, we picked the ones that lead to the highest proportion of locally optimal moves.

For the Fourier models, the only model parameter to choose is the set of irreps/Fourier basis functions to use for parameterizing V_θ. Finding the best k irreps to use would require performing a cross validation over $O(|R|^k)$ different combinations of irreps, which is not practical. Instead, for each puzzle, for each irrep ρ, we trained a Fourier model using only that single irrep ρ over 5000 epochs. We then ranked the irreps by the proportion of locally optimal moves made over a random sample of puzzle states. We suspect that a deeper understanding of the group structure of the puzzles will inform us on how to choose which irreps to use which we leave for future work. For further details on the range of values searched over for the network architectures, see our supplement.

The algorithm for learning the value function with the DVN is identical to Algorithm 1 except lines $5-6$ use the one-hot encoding of the puzzle states instead of the irreducible representations and V_θ would instead be a neural network parameterized with weights θ. For

$\mathbb{S}_n$ the one-hot encoding of $\sigma \in \mathbb{S}_n$ is its corresponding permutation matrix vectorized. For Pyraminx and the 2-by-2 cube, which are both wreath product groups, recall that an element of the wreath product group $C_m \wr \mathbb{S}_n$ can be represented as a tuple (τ, σ), where $\tau \in C_m^n$ and $\sigma \in \mathbb{S}_n$. The onehot encoding of τ is an $m \times n$ binary vector. A onehot encoding of (τ, σ) is just a concatenation of the individual onehot-encodings of τ and σ. So the one-hot encoding of a state of the Pyraminx puzzle $(C_2 \wr \mathbb{S}_6)$ has length $6 \times 6 + 2 \times 6 = 48$. The one-hot encoding of a state from the 2-by-2 cube $(C_3 \wr \mathbb{S}_8)$ has length $8 \times 8 + 3 \times 8 = 88$.

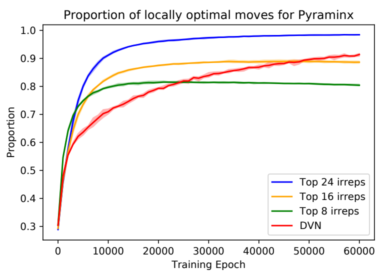
Evaluation We evaluated the proportion of local optimal moves made and proportion of random puzzles solved using a greedy policy in conjunction with the learned value function every 1000 epochs. We say that the value function produces a locally optimal move at state s if the argmax of V on the neighbors of s is in fact a state that is closer to the goal state than s. For the two smaller puzzles, we evaluated the proportion of locally optimal moves and greedy solves over all the legal puzzle states. The 2-by-2 cube is too large to do this for all for all cube states so we instead did this for a random sample of 1000 cubes. We repeated the training for each model over five random seeds. In Figures 3 and 4, we plot the median values of these metrics with the max and min shaded. All experiments were run on a GeForce GTX 1080 Ti GPU.

Heuristic search with the value function To get another measure of how effective these learned value functions were, we used the Fourier models and the DVN in A^* search and kept track of the number of states explored. We only do this for the 2-by-2 cube since a greedy/best first search policy is already quite effective on the smaller puzzles. A^* search results in an optimal path if the heuristic function used is admissable (never underestimates the true distance to the goal state). There are no such guarantees on the Fourier model or the DVN being admissable heuristics, but it is reasonable to use A^* search since the 2-by-2 is small enough that the search will eventually terminate after visiting all states if our value functions are particularly ineffective as heuristics. We generate 1000 sample test cubes uniformly and tested the Fourier models (including the low rank models) and the DVN. A puzzle state is considered "explored" after we evaluate the value function on each of its neighbors and add them on the priority queue of puzzle states to explore next. In general, it is preferable to have a heuristic function that requires fewer state explorations. Node statistics using an optimal heuristic function (Djikstra's shortest path) are also shown so that we have a sense of how suboptimal these value functions performed.

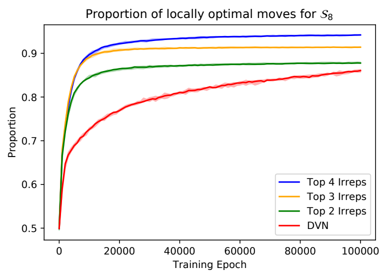
Learning low rank Fourier models One of the downsides of the Fourier approach is its memory footprint. The top 2 irreps of the 2-by-2 cube have dimensionality 560×560 and 420×420. So the learned Fourier matrices have: $2 \times (560^2 + 420^2) = 9.8 \times 10^5$ parameters (note: the factor of two comes from the fact that the irreps of the 2-by-2 cube are complex valued). However, as proposed in Section 5.1, we can address the scaling issues by learning low rank Fourier matrices. To demonstrate the feasibility of learning low rank Fourier matrices, we train rank $1, 10$ and 100 Fourier matrices for a Fourier model that uses the top 2 irreducible representation of the 2-by-2 cube. We use these resulting low rank Fourier models in A^*-search as well. Plots for the proportion of greedy solves and locally optimal moves of the low rank Fourier models are in the supplement.

Discussion Figures 3 and 4 show that our Fourier approach unequivocally outperforms DVN in terms of the number of states that need to be seen to learn a successful policy. Table 2 gives the final proportion of random puzzles solved using each of the learned value functions with greedy search and the number of unique puzzle states seen during training. For the smaller puzzles, the DVN is still a reasonable policy and with more training would likely outperform the Fourier models. For the 2-by-2 cube, the difference between the DVN and the Fourier models is substantial: the Fourier models learn can solve up to 86% of the sampled cubes while the DVN can only solve around 14% at most under a greedy policy. It is not entirely surprising that the Fourier models would outperform the DVN since the DVN must learn a representation from the one-hot encoding of puzzle states. As shown in Table 3, when the Fourier based value functions are used in heuristic search, we solve all the randomly sampled test cubes within the allotted state exploration budget. The irrep models do not find optimal (shortest) paths to the solved state (in terms of number of states explored), but considering the number of unique cube states they were trained on (especially the low rank models), we believe the performance is noteable.

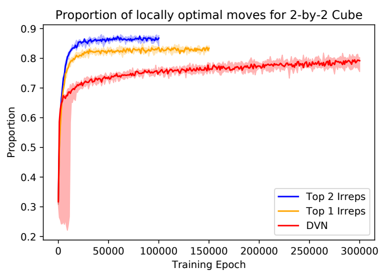
Our results lend support to the common complaint that deep reinforcement learning is quite sample inefficient. Using one-hot encodings and deep neural networks as function approximators for solving permutation puzzles is attractive because they avoid the need for any domain expertise; the dynamics of the puzzle can be learned by sufficiently sampling the state space. We show that there is a middle ground between using domain knowledge and learning from the sampled puzzle states through reinforcement learning by using the Fourier basis. We can still learn effective value functions through the same value iteration techniques



(a) Pyraminx

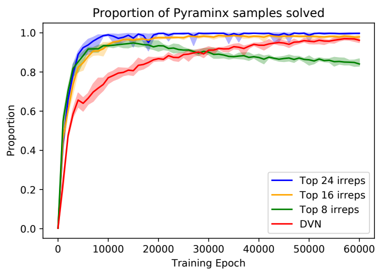


(b) $\mathbb{S}_8$

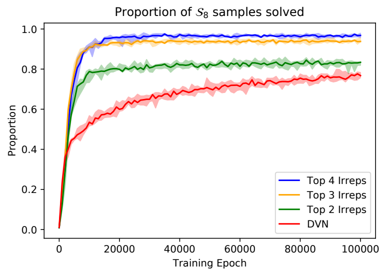


(c) 2-by-2 Cube

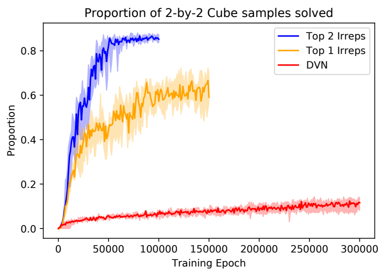
Figure 3: Proportion of locally optimal moves



(a) Pyraminx



(b) $\mathbb{S}_8$



(c) 2-by-2 Cube

Figure 4: Proportion of puzzles solved with greedy policy

and we can do so much more efficiently in the Fourier basis, which are intrinsic to each puzzle.

7 Limitations and Future Work

While there are concerns on how well this method can scale, the effectiveness of the low rank Fourier models gives us hope that we can extend this work to larger groups, including the 3-by-3 Rubik's Cube. The 3-by-3 Rubik's cube's group is a subgroup of $(C_3 \wr \mathbb{S}_8) \times (C_2 \wr \mathbb{S}_{12})$, where $C_2 \wr \mathbb{S}_{12}$ is the group that corresponds to how the twelve edge facets of the 3-by-3 cube permute amongst themselves. Its irreducible representations are the tensor products $\rho \otimes \phi$, where ρ is an irrep of $C_3 \wr \mathbb{S}_8$ and ϕ is an irrep of $C_2 \wr \mathbb{S}_{12}$. The resulting Fourier matrix has size $(d_\rho \times d_\phi) \times (d_\rho \times d_\phi)$. To learn a value function V for the Rubik's Cube group, we would likely parameterize the learnable Fourier matrices in the value function V as the tensor products of low rank matrices to make the problem tractable. Another potential saving grace is that these irreps of wreath product groups are quite sparse as we describe in detail in the supplement.

The main challenge of applying this Fourier basis approach to larger groups is that computing their basis functions/irreps gets progressively more difficult. For wreath product groups, we need to compute induced representations, which involve identifying how various cosets are permuted by a given group element. For a group as large as $C_2 \wr \mathbb{S}_{12}$ that has 4.9×10^{11} elements, we would likely need to employ tools from computational group theory such as the Schreier-Sims algorithm (Seress, 2003) as a subroutine for these coset related tasks. Luckily, Schreier-Sims and its variants do run in polynomial time (polynomial in the log size of the group and the order of the base symmetric group). The irreps also get much larger but remain sparse. The largest irrep of $C_2 \wr \mathbb{S}_{12}$ has dimension 202752, but also only has 202752 nonzero entries so they can be stored in sparse matrix format. While this all seems quite expensive, recall that the size of this group is already too large to even run a breadth first search over.

Finding the top irreps to use in the value function remains a nontrivial and open ended research question. Our method of ranking the irreps based on how well each of them performed on a short training run may be too computationally expensive for larger groups.

8 Conclusion

We developed a novel approach for solving permutation puzzles using the Fourier basis of each puzzle's under-

Table 2: Proportion of puzzles solved by greedy search

Puzzle	Model	Parameters	% Greedy Solves ($\pm$ std)	% States seen	Train time (hrs)
Pyraminx	DVN	1.1×10^6	96.2 ± 0.7	100	2.5
	Top 8 Irrep	9.5×10^3	65.4 ± 1.7	100	1.1
	Top 16 Irrep	1.4×10^4	84.1 ± 1.4	100	1.2
	Top 24 Irrep	1.7×10^4	96.9 ± 0.7	100	1.3
$\mathbb{S}_8$	DVN	4.3×10^6	77.0 ± 0.9	93.8	2.9
	Top 2 Irrep	8.0×10^3	81.6 ± 0.1	93.8	1.6
	Top 3 Irrep	9.2×10^3	93.1 ± 0.7	93.8	1.6
	Top 4 Irrep	1.2×10^4	95.2 ± 1.5	93.8	1.7
2-by-2 Cube	DVN	1.0×10^7	11.4 ± 2.1	47.6	14.5
	Top 1 Irrep	6.3×10^5	67.3 ± 2.2	19.2	9.2
	Top 2 Irrep	9.8×10^5	86.6 ± 0.7	13.2	12.3
	Top 2 Irrep - Rank 1	2.8×10^5	9.4 ± 1.1	4.5	2.9
	Top 2 Irrep - Rank 10	2.8×10^4	47.7 ± 1.4	4.5	3.0
	Top 2 Irrep - Rank 100	2.8×10^3	58.8 ± 1.3	4.5	3.5
All Puzzles	Opt Solver(Djikstras)	-	100	100%	-

Table 3: 2-by-2 cube A^* search solve statistics

		Number of states explored		
Model	Solved	Lower Quartile	Median	Upper Quartile
Opt Solver(Djikstras)	100%	10	11	11
DVN	100%	25	67.5	179.75
Top 1 Irrep - Full Rank	100%	12	13	16
Top 2 Irrep - Full Rank	100%	11	12	14
Top 2 Irrep - Rank 100	100%	11	13	16
Top 2 Irrep - Rank 10	100%	12	14	17
Top 2 Irrep - Rank 1	100%	13	18	25

lying group. Our method learns a more effective policy for solving the 2-by-2 cube using far fewer samples than the existing standard deep reinforcement learning method. As far as we are aware, this is also the first application of the representation theory of wreath product groups and the symmetric group in reinforcement learning.

9 Acknowledgements

This work was done using computational resources provided by the University of Chicago's Research Computing Center's Midway and GM4 clusters. We thank the Flatiron Institute for hosting H. Pan during parts of the preparation of this paper. We also acknowledge generous support from DARPA grant number HR00111890038 and NSF award number 1828629.

References

Forest Agostinelli, Stephen McAleer, Alexander Shmakov, and Pierre Baldi. Solving the Rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1(8):356–363, 2019.

Richard Bellman. A Markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

Bastian Bischoff, Duy Nguyen-Tuong, Heiner Markert, and Alois Knoll. Solving the 15-puzzle game using local value-iteration. In *SCAI*, pages 45–54, 2013.

Robert Brunetto and Otakar Trunda. Deep heuristic-learning in the rubik's cube domain: An experimental evaluation. In *ITAT*, pages 57–64, 2017.

Zhaoxing Bu and Richard E Korf. A*+ ida*: A simple hybrid search algorithm. In *IJCAI*, pages 1206–1212, 2019.

Tullio Ceccherini-Silberstein, Fabio Scarabotti, and Filippo Tolli. *Representation theory and harmonic analysis of wreath products of finite groups*, volume 410. Cambridge University Press, 2014.

Persi Diaconis. *Group Representations in Probability and Statistics*, volume 11 of *Lecture Notes-Monograph Series*. Institute of Mathematical Sciences, 1988.

Carmel Domshlak, Michael Katz, and Alexander Shleyfman. Enhanced symmetry breaking in cost-optimal planning as forward search. In *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.

Ariel Felner, Richard E Korf, and Sarit Hanan. Additive pattern database heuristics. *Journal of Artificial Intelligence Research*, 22:279–318, 2004.

M. Fox and D. Long. The detection and exploitation of symmetry in planning problems. In *IJCAI*, 1999.

M. Fox and D. Long. Extending the exploitation of symmetries in planning. In *AIPS*, 2002.

Jonathan Huang, Carlos Guestrin, and Leonidas J Guibas. Efficient inference for distributions on permutations. In *Advances in neural information processing systems*, pages 697–704, 2008.

Jonathan Huang, Carlos Guestrin, and Leonidas Guibas. Fourier theoretic probabilistic inference over permutations. *Journal of Machine Learning Research*, 10(5), 2009.

Philipp W Keller, Shie Mannor, and Doina Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *Pro-*

ceedings of the 23rd international conference on Machine learning, pages 449–456, 2006.

A. Kerber. *Representations of Permutation Groups I*, volume 240 of *Lecture Notes in Mathamatics*. Springer-Verlag Berlin Heidelberg, 1971.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Herbert Kociemba. Two-phase algorithm details. `http://kociemba.org/cube.html`, n.d. [Online: Accessed: 03/13/2020].

Risi Kondor and Marconi S Barbosa. Ranking with kernels in fourier space. In *COLT*, pages 451–463, 2010.

Risi Kondor and Walter Dempsey. Multiresolution analysis on the symmetric group. In *Advances in Neural Information Processing Systems*, pages 1637–1645, 2012.

Risi Kondor, Andrew Howard, and Tony Jebara. Multi-object tracking with representations of the symmetric group. In *Artificial Intelligence and Statistics*, pages 211–218, 2007.

George Konidaris, Sarah Osentoski, and Philip Thomas. Value function approximation in reinforcement learning using the fourier basis. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 380–385, 2011.

Richard E Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109, 1985.

Richard E Korf. Finding optimal solutions to rubik's cube using pattern databases. In *AAAI/IAAI*, pages 700–705, 1997.

Richard E Korf and Larry A Taylor. Finding optimal solutions to the twenty-four puzzle. In *Proceedings of the national conference on artificial intelligence*, pages 1202–1207, 1996.

R Matthew Kretchmar and Charles W Anderson. Comparison of cmacs and radial basis functions for local function approximators in reinforcement learning. In *Proceedings of International Conference on Neural Networks (ICNN'97)*, volume 2, pages 834–837. IEEE, 1997.

Sridhar Mahadevan. Proto-value functions: Developmental reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 553–560, 2005.

Sridhar Mahadevan and Mauro Maggioni. Value function approximation with diffusion wavelets and laplacian eigenfunctions. In *Advances in neural information processing systems*, pages 843–850, 2006.

Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 8(Oct):2169–2231, 2007.

Horia Mania, Aaditya Ramdas, Martin J Wainwright, Michael I Jordan, Benjamin Recht, et al. On kernel methods for covariates that are rankings. *Electronic Journal of Statistics*, 12(2):2537–2577, 2018.

Stephen McAleer, Forest Agostinelli, Alexander Shmakov, and Pierre Baldi. Solving the rubik's cube with approximate policy iteration. In *International Conference on Learning Representations*, 2018.

Ishai Menache, Shie Mannor, and Nahum Shimkin. Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research*, 134(1):215–238, 2005.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. 2013. URL `http://arxiv.org/abs/1312.5602`. cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Nir Pochter, Aviv Zohar, and Jeffrey S Rosenschein. Exploiting problem symmetries in state-based planners. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

Daniel N Rockmore. Fast Fourier transforms for wreath products. *Applied and Computational Harmonic Analysis*, 2(3):279–292, 1995.

Ákos Seress. *Permutation Group Algorithms*. Cambridge Tracts in Mathematics. Cambridge University Press, 2003. doi: 10.1017/CBO9780511546549.

Jean-Pierre Serre. *Linear representations of finite groups*, volume 42. Springer, 1977.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Jerry Swan. Harmonic analysis and resynthesis of sliding-tile puzzle heuristics. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 516–524. IEEE, 2017.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. In *Machine Learning*, pages 279–292, 1992.