Exploiting Symmetries for Single and Multi-Agent Partially Observable Stochastic Domains

Byung Kon Kang, Kee-Eung Kim*

KAIST,
373-1 Guseong-dong Yuseong-gu
Daejeon 305-701, Korea

Abstract

While Partially Observable Markov Decision Processes (POMDPs) and their multi-agent extension Partially Observable Stochastic Games (POSGs) provide a natural and systematic approach to modeling sequential decision making problems under uncertainty, the computational complexity with which the solutions are computed is known to be prohibitively expensive.

In this paper, we show how such high computational resource requirements can be alleviated through the use of symmetries present in the problem. The problem of finding the symmetries can be cast as a graph automorphism (GA) problem on a graphical representation of the problem. We demonstrate how such symmetries can be exploited in order to speed up the solution computation and provide computational complexity results.

Key words: POMDP, POSG, Symmetry, Graph Automorphism

1. Introduction

Markov Decision Processes (MDPs) have been a classical mathematical framework for sequential decision making problems, in which the agent must make action decisions based on environment states. The number of steps at which the agent can make decisions can either be finite or infinite, leading to finite-horizon and infinite horizon problems, respectively. However, although computationally tractable, MDPs have often been shown inadequate to successfully model the agent's noisy perception of the environment state. In order to incorporate the uncertainty about the state perception inherent in the agent, an extended formalism called partially observable MDPs (POMDPs) has emerged [12, 34, 31].

*Corresponding author. Department of Computer Sciences at KAIST, 373-1 Guseong-dong Yuseong-gu Daejeon, Republic of Korea.
Telephone: +82-42-350-3536; Fax: +82-42-350-3510.
Email addresses: bkkang@ai.kaist.ac.kr (Byung Kon Kang), kekim@cs.kaist.ac.kr (Kee-Eung Kim)

POMDPs provide a model for single-agent sequential decision making under state uncertainty thus turning the decision making problem into one of planning ([12]). Different from MDPs, POMDPs do not provide the agent with full observability of the states. Instead, the agent must infer which state it is in based on the noisy observations. This results in defining a probability distribution over the states, defined as a *belief state*, to represent the uncertainty of the states. With this single extra assumption, the computational complexity of solving a POMDP problem jumps from P-complete (MDP) to PSPACE-complete even for finite-horizon POMDPs [23]. Solving infinite-horizon POMDPs is known to be undecidable [17].

There has been a lot of work on alleviating this intractability by means of computing approximate solutions. One of the most well-known works that shows both practicality and theoretical guarantees is Point-Based Value Iteration (PBVI) by Pineau et al. [25]. PBVI proceeds by sampling reachable belief states according to various heuristics in order to avoid the curse of dimensionality induced by the continuous nature of the belief states. The value backups are performed only on those sampled belief states before collecting additional belief states. The main factor that determines the performance of PBVI is the belief point selection heuristic. The heuristics used are intended to capture the reachability of the belief points, thereby avoiding unnecessary computation on unreachable beliefs. One popular heuristic used is the Greedy Error Reduction heuristic, which samples belief points that result in the largest error bound. PBVI belongs to a class of algorithms called *point-based* methods, because value computation is performed on a finite set of belief states, often called *belief points*.[1]

Heuristic Search Value Iteration (HSVI) by Smith and Simmons [30] is another point-based method that approximates the value function via heuristic exploration of belief states. It maintains an upper- and a lower-bound for the true value function, and decreases the bound gaps by recursively selecting belief states. The upper bound is initialized at the corners of the belief simplex and is maintained as a point set. Update to this bound is performed by adding a new belief point, whose value is computed as a projection onto a convex hull formed by belief-value pairs. The lower bound is a vector set, meaning that the value is updated at the newly added belief, much like the updates performed in PBVI. The belief point to be added is selected by a depth-first search from the initial belief.

Another approach by which the intractability of the POMDP solution can be eased (in a practical manner) is to take advantage of the structural regularities present in POMDPs. One popular method uses the concept of *homomorphism* to reduce the state space itself, thereby forming an equivalent model with a potentially much smaller size. This technique, often called *model minimization*, has been extensively studied in the MDP domain, making use of *stochastic bisimulation* of the states. Informally, bisimilar states can be grouped together to form a smaller state space than the original MDP, yet the optimal policies of the original and the reduced MDP are directly related with each other. The structural characteristics that allow for state grouping are reward equivalence and block transition equivalence. The former property states that the states in a group should yield the same reward for any given action, and the latter that grouped

[1] In the sequel, we use the terms "belief points" and "belief states" interchangeably.

states should have the same transition probability into the group of original destination states. It is known that the optimal policy of the reduced MDP can be *lifted* to be converted into the optimal policy of the original MDP, hence model reduction results in less computation [8, 10].

A different structural feature that is of interest to us is *automorphism*. An automorphism of a model is a homomorphism to itself. By finding the automorphisms, or *symmetries*, present in the model, one may expect to reduce possibly redundant computation performed on the symmetric portion of the solution space. It is this feature that we propose to use on POMDPs in order to reduce computational resources needed for computing optimal solutions. In particular, we are interested in the POMDP symmetry that is not related to reducing the size of the model, but can nonetheless be exploited to speed up conventional point-based POMDP algorithms introduced above.

The subject of symmetry in sequential decision making has not been carried out actively, with a few exceptions: Ravindran and Barto [26] were the first to extend the model minimization method to cover symmetries in MDPs. More recently, Narayanamurthy and Ravindran [20] constructively proved that the problem of finding symmetries in MDPs belongs to the complexity class graph isomorphism-complete (GI-complete). In this latter work, the authors use a graph-based encoding of the MDP to cast the problem of finding MDP symmetries to that of graph automorphism (GA). Our work is similar to this, in that we also reduce the problem to discovering GA, but provides a simpler and more intuitive approach along with a practical guide to applying symmetries to existing algorithms. We also extend the domain to multi-agent settings.

Another work similar to ours is that of permutable POMDPs by Doshi and Roy [9]. This work was presented in the context of preference elicitation where the belief states have permutable state distribution. Similarly to the approach we present, this permutable POMDP framework is based on the idea that the value functions of certain classes of POMDPs are permutable with respect to the state permutation. That is, the components of the value function can be permuted according to the permutation of their corresponding states while maintaining value invariance. While the overall idea is in league with our approach, there are two important differences. First, the permutable POMDP only considers a specific type of symmetry that can be found in preference elicitation problems and models similar to them. More specifically, they show how certain preference elicitation problems can be set up to exhibit symmetric properties. That is, they first provide certain conditions a state permutation should satisfy and show how a preference elicitation POMDP can have its parameters set in order to satisfy the stated conditions. As opposed to such setting, our research aims to provide an algorithmic framework with which symmetries can be discovered and exploited in general POMDP problems. Second, their symmetry definition requires that the equality condition hold for all $n!$ permutations, where n is the number of states. This is a very strict condition, and is therefore suitable for only a very limited set of problems. On the other hand, our formulation relaxes this restriction by considering the state, action, and observation permutations in *groups*.

Partially Observable Stochastic Games (POSGs) are a multi-agent extension to the POMDPs, where the actions and observations now take a collective form of all agents. This change induces another leap in the complexity hierarchy: planning in finite-horizon Decentralized POMDPs (DEC-POMDPs), which is a special class of

POSGs with common payoffs, is known to be NEXP-complete [3]. Planning in infinite-horizon DEC-POMDP is again undecidable since DEC-POMDPs is a generalization of POMDPs. Hansen et al. [11] give an exact algorithm for solving POSGs, by means of Multi-Agent Dynamic Programming (MADP). MADP performs dynamic programming backups over an extended, *multi-agent belief space*, which is a distribution over both the latent state and the policies of other agents. In order to keep memory usage in check, the notion of *dominance* is used to prune unnecessary intermediate solutions at each iteration.

In this paper, as an extended version of our previous work [13], we extend the algorithm to that of exploiting symmetries for POSGs as well. In particular, we will show how the notion of symmetries can be extended to a multi-agent case and how it affects some of the game-theoretic concepts in POSGs.

2. Formal Models: POMDPs and POSGs

Before we present our main results and algorithms in detail, we first review the preliminaries of formal models for the single and multi-agent sequential decision making problems in partially observable environments used in this paper. We also define optimal solutions for the models and representations for these solutions.

2.1. POMDPs

The *partially observable Markov decision process (POMDP)* [12] is a model for sequential decision making problems in single agent settings. It is a generalization of the MDP model that relaxes the assumption that the agent has complete information about the environment states.

Formally, a POMDP is defined as a tuple $\langle S, A, Z, T, O, R, b_0 \rangle$, where

- S is the set of environment states,
- A is the set of actions available to the agent,
- Z is the set of all possible observations,
- $T : S \times A \times S \to [0, 1]$ is the transition function with $T(s, a, s') = P(s'|s, a)$ denoting the probability of changing to state s' from state s by executing action a,
- $O : S \times A \times Z \to [0, 1]$ is the observation function with $O(s, a, z) = P(z|s, a)$ denoting the probability of making observation z when executing action a and arriving in state s,
- $R : S \times A \to \Re$ is the reward function where $R(s, a)$ denotes the immediate reward received by the agent when executing action a in state s,
- b_0 is the initial state distribution with $b_0(s)$ denoting the probability that the environment starts in state s.

Since the agent cannot directly observe the states, it has to consider the history of its past actions and observations to decide the current action. The *history* at time t is defined as

$$h_t = \{a_0, z_1, a_1, z_2, \ldots, a_{t-1}, z_t\}$$

The action is determined by a policy π, which is a function that maps from the histories to actions. For finite-horizon problems, where we assume that the agent can execute actions for a finite time steps, the policy can be represented using a *policy tree*, where each node is labeled with the action to execute, and each edge is labeled with the observation that the agent can receive at each time step. Following an observation edge, the agent faces the next level subtree, whose root node specifies the action to execute at the next time step. The sequence of action nodes and observation edges traversed while executing the policy naturally becomes the history.

The history leads to the definition of a *belief state*, which is the probability distribution on the states given the history of actions and observations:

$$b_t(s) = P(s_t = s | h_t, b_0)$$

Upon executing action a_t and receiving observation z_{t+1}, the belief state $b_{t+1} = \tau(b_t, a_t, z_{t+1})$ at the next time step is computed by the Bayes rule:

$$b_{t+1}(s') = \frac{O(s', a_t, z_{t+1}) \sum_{s \in S} T(s, a_t, s') b_t(s)}{P(z_{t+1} | b_t, a_t)},$$

where

$$P(z_{t+1} | b_t, a_t) = \sum_{s' \in S} O(s', a_t, z_{t+1}) \sum_{s \in S} T(s, a_t, s') b_t(s).$$

The belief state b_t constitutes a sufficient statistic for history h_t, and can be represented as an $|S|$-dimensional vector. We can thus re-define the policy as a mapping from belief states to actions.

The *value* of a policy is the expected discounted sum of rewards by following the policy starting from a certain belief state. The optimal value function is the one obtained by following an optimal policy, and can be defined recursively: given the t-1 step optimal value function, the t-step optimal value function is defined as

$$V_t^*(b) = \max_a \left[R(b, a) + \gamma \sum_{z \in Z} P(z | b, a) V_{t-1}^*(\tau(b, a, z)) \right] \tag{1}$$

where $R(b, a) = \sum_s b(s) R(s, a)$ and $\gamma \in [0, 1)$ is the discount factor.

2.2. POSGs

The *partially observable stochastic game (POSG)* [3, 11] is an extension of the POMDP framework to multi-agent settings. More formally, a POSG with n agents is defined as a tuple $\langle I, S, b_0, \{A_i\}, \{Z_i\}, T, O, \{R_i\} \rangle$, where

- I is the finite set of agents indexed $1, \ldots, n$.
- S is the finite set of environment states.

- b_0 is the initial state distribution where $b_0(s)$ is the probability that the environment starts in state s.
- A_i is the finite set of actions available to agent i. Also, the set of *joint actions* is specified as $\vec{A} = \prod_{i \in I} A_i$.
- Z_i is the finite set of observations available to agent i. Similarly, the set of *joint observations* is defined as $\vec{Z} = \prod_{i \in I} Z_i$.
- T is the transition function where $T(s, \vec{a}, s') = P(s'|s, \vec{a})$, the probability of resulting in state s' when executing joint action $\vec{a}$ in state s.
- O is the observation function where $O(s, \vec{a}, \vec{z}) = P(\vec{z}|\vec{a}, s)$, the probability of making joint observation $\vec{z}$ when executing joint action $\vec{a}$ and arriving in state s.
- R_i is the individual reward function where $R_i(s, \vec{a})$ denotes the reward (payoff) received by agent i when joint action $\vec{a}$ is executed in state s.

If we restrict every agent to share the same individual reward function, the model becomes the Decentralized POMDP (DEC-POMDP) [3].

In POSGs, each agent independently makes its own decision based on the local information available to the agent. The local information at time t for agent i can be represented as the local history

$$h_{i,t} = \{a_{i,0}, z_{i,1}, a_{i,1}, z_{i,2}, \ldots, a_{i,t-1}, z_{i,t}\}$$

where actions $a_{i,*}$ and observations $z_{i,*}$ are from the set A_i and Z_i, respectively. The local policy (*i.e.*, strategy) π_i executed by agent i is then essentially a mapping from the local histories to local actions. A joint policy is a set of local policies for each agent. Algorithms for POSGs find the joint policy, which is the set of local policies $\vec{\pi} = \{\pi_1, \ldots, \pi_n\}$ for each agent, for solution concepts such as Nash equilibrium or correlated equilibrium. In the case of DEC-POMDPs where the agents have to cooperate, the algorithms search for the optimal joint policy that maximizes the expected sum of rewards over the planning horizon.

The agents in POSGs have to reason about other agents' policies as well as the true state, since they collectively affect the rewards and the state transitions, and hence the value. This leads to the definition of *multi-agent belief state*, which is a probability distribution over the hidden states and other agents' policies [19]. Hence, while dynamic programming methods for POMDPs involve belief states and value vectors defined only over the system states, methods for POSGs involve multi-agent belief states and value vectors defined over the joint space of the states and other agents' policies. Thus, for each policy $\pi \in \Pi_i$ of agent i, there exists a value vector V_i^{π} of dimension $|S||\vec{\Pi}_{-i}|$, where $\vec{\Pi}_{-i}$ is the set of policies for all other agents except agent i. In this paper, we focus on finite-horizon problems, and assume the local policy is represented as a decision tree.

Formally, agent i's t-step value function of executing policy π while others are executing policy $\vec{\pi}_{-i}$ can be defined as

$$V_{i,t}^{\pi}(s, \vec{\pi}_{-i}) = R_i(s, \vec{a}_{\vec{\pi}}) + \gamma \sum_{\vec{z} \in \vec{Z}} O(s, \vec{a}_{\vec{\pi}}, \vec{z}) \sum_{s' \in S} T(s, \vec{a}_{\vec{\pi}}, s') V_{i,t-1}^{\pi(z_i)}(s', \vec{\pi}_{-i}(\vec{z}_{-i})) \tag{2}$$

where $\vec{\pi} = \{\pi, \vec{\pi}_{-i}\}$ is the joint policy formed by π for agent i and $\vec{\pi}_{-i}$ for other agents, $\vec{a}_{\vec{\pi}}$ is the joint action for the current time step prescribed by the policy $\vec{\pi}$, $\pi(z_i)$ is the (t-1)-step local policy for agent i after observation of z_i, and $\vec{\pi}_{-i}(\vec{z}_{-i})$ is the (t-1)-step joint policy for other agents after observation of $\vec{z}_{-i}$. For a given multi-agent belief state b_i, the agent i's value of executing local policy π is defined as

$$V_{i,t}^{\pi}(b_i) = \sum_{s \in S} \sum_{\vec{\pi}_{-i} \in \vec{\Pi}_{-i}} b_i(s, \vec{\pi}_{-i}) V_{i,t}^{\pi}(s, \vec{\pi}_{-i}) \tag{3}$$

3. Solution Methods

In this section, we briefly review some important solution techniques for POMDPs and POSGs. There exists a wealth of literature presenting various algorithms on this matter, but in this paper, we only discuss point-based value iteration (PBVI) [25] for POMDPs and multi-agent dynamic programming (MADP) [11] for POSGs, which will be discussed in the later sections.

3.1. PBVI for POMDPs

The definition of the optimal value function in Equation 1 leads to a dynamic programming update to obtain the t-step optimal value function V_t^* from the $(t-1)$-step optimal value function V_{t-1}^*. The dynamic programming update could be represented as a *backup operator* H on the value functions, such that given a belief state b,

$$V_t(b) = HV_{t-1}(b) = \max_a \left[R(b,a) + \gamma \sum_{z \in Z} P(z|a,b) V_{t-1}(\tau(b,a,z)) \right].$$

Since belief states provide a sufficient statistic for the histories, they can be treated as states in a continuous MDP, namely the belief state MDP. One shortcoming of this approach is that the belief state is continuous, and so we cannot simply use tabular representation for value functions as in discrete state space MDPs, hence naively performing the backup operation for every possible belief state becomes intractable. However, Sondik [31] pointed out that the value function for each horizon t can be represented by a set $\Gamma_t = \{\alpha_0, \cdots, \alpha_m\}$ of α-vectors, so that the value at a particular belief state b is calculated as:

$$V_t(b) = \max_{\alpha \in \Gamma_t} \sum_{s \in S} \alpha(s) b(s).$$

The construction of Γ_t is carried out via a series of intermediate Γ generation:

$$\begin{aligned}
\Gamma_t^{a,*} &= \{\alpha^{a,*} | \alpha^{a,*}(s) = R(s,a)\} \\
\Gamma_t^{a,z} &= \{\alpha_i^{a,z} | \alpha_i^{a,z}(s) = \gamma \textstyle\sum_{s' \in S} T(s,a,s') O(s',a,z) \alpha_i(s'), \forall \alpha_i \in \Gamma_{t-1}\} \\
\Gamma_t^{a} &= \Gamma_t^{a,*} + \oplus_{z \in Z} \Gamma_t^{a,z} \\
\Gamma_t &= \cup_{a \in A} \Gamma_t^a,
\end{aligned}$$

where the *cross-sum* operator $\oplus$ on sets A and B is defined as:

$$A \oplus B = \{a + b | \forall a \in A, b \in B\}.$$

However, $|\Gamma_t|$ can be in the order of $O(|A|\,|\Gamma_{t-1}|^{|Z|})$ in the worst case, leading to a very high computational cost. The doubly exponential growth of $|\Gamma_t|$ in t can be alleviated by pruning dominated α-vectors for all possible belief states, but the effect of pruning is limited in practice. This is mainly due to the fact that the backup is done over all possible belief states. Point-based value iteration (PBVI) [25] attempts to limit this growth by performing backups only on a finite set B of reachable belief states. Hence, in finding Γ_t for V_t, PBVI constructs $\Gamma_t^{a,b}$, $\forall a \in A$, $\forall b \in B$, whose elements are calculated by

$$\Gamma_t^{a,b} = \left\{ \alpha_b^a | \alpha_b^a(s) = R(s,a) + \textstyle\sum_{z \in Z} \left[\mathrm{argmax}_{\alpha \in \Gamma_t^{a,z}} (\alpha \cdot b) \right](s) \right\}$$

and finally compute the best action for each belief state

$$\Gamma_t^B = \{\alpha | \alpha = \mathrm{argmax}_{a \in A, \alpha_b^a \in \Gamma_t^{a,b}} (\alpha_b^a \cdot b), \;\; \forall b \in B\}$$

Using Γ_t (or Γ_t^B as an approximation) for V_t, the policy simply takes the form of choosing the action associated with $\mathrm{argmax}_{\alpha \in \Gamma_t}(\alpha \cdot b)$. Table 1 outlines PBVI. The BACKUP routine refers to the process of creating Γ_t^B, described above. The EXPAND routine characterizes the heuristic aspect of PBVI, whose task is to collect reachable belief states from the given set B of beliefs. Heuristics for EXPAND include: *greedy error reduction*, where the belief states that reduce the expected error bound are greedily chosen, and *stochastic simulation with explorative action*, where the belief states that mostly reduce the maximum distance among sampled belief states are greedily chosen. In later sections, we will modify the BACKUP routine in order to exploit the symmetries in POMDPs.

3.2. MADP for POSGs

Hansen et al. [11] propose a multi-agent dynamic programming (MADP) algorithm for POSGs. The dynamic programming update in MADP consists of two stages, first enumerating t-step policies from (t-1)-step policies and evaluating these policies to obtain value functions, and then eliminating policies that are not useful for any multi-agent belief state.

Note that the multi-agent value function in Equation 3 was represented as the set of $|S||\vec{\Pi}_{-i}|$-dimensional vectors. While the dynamic programming methods for POMDPs, such as PBVI, involve belief states and value vectors defined only over the

Require: B_{init} (initial set of belief states), K (maximum number of belief state expansions), and T (maximum number of backups)
  $B = B_{init}$
  $\Gamma = \{\}$
  for $k = 1, \ldots, K$ **do**
    for $t = 1, \ldots, T$ **do**
      $\Gamma =$ BACKUP(B, Γ)
    end for
    $B_{new} =$ EXPAND(B, Γ)
    $B = B \cup B_{new}$
  end for
  Return Γ

Table 1: The PBVI algorithm

environment states, the methods for POSGs involve multi-agent belief states and value vectors defined over the joint space of environment states and other agents' policies. Hence the dimension of value vectors will vary whenever a policy is eliminated in the second stage of dynamic programming update. A more convenient way to represent the value is to prepare a value vector for each joint policy $\vec{\pi} \in \vec{\Pi}_t$, so that the state value vectors and belief vectors be of a fixed dimension $|S|$:

$$V_{i,t}^{\vec{\pi}}(s) = R_i(s, \vec{a}_{\vec{\pi}}) + \gamma \sum_{\vec{z} \in \vec{Z}} O(s, \vec{a}_{\vec{\pi}}, \vec{z}) \sum_{s' \in S} T(s, \vec{a}_{\vec{\pi}}, s') V_{i,t-1}^{\vec{\pi}(\vec{z})}(s') \quad (4)$$

The corresponding value function for a specific belief $b \in [0,1]^{|S|}$ is:

$$V_{i,t}^{\vec{\pi}}(b) = \sum_{s \in S} b(s) V_{i,t}^{\vec{\pi}}(s). \quad (5)$$

Notice that given Equation 2, we can convert it into Equation 4 by concatenating $\vec{\pi}_{-i}$ and π to construct the joint policy $\vec{\pi}$. Also, given a joint policy, a state belief vector of dimension $|S|$ can be computed for any horizon t based on the given initial state distribution b_0 and the action/observation history up to time t. Thus, Equation 3 can be represented as Equation 5. We will use Equation 5 to represent the value for the rest of the section, for ease of exposition.

Given the set $\vec{\Pi}_{t-1} = \Pi_{1,t-1} \times \cdots \times \Pi_{i,t-1} \times \cdots \times \Pi_{n,t-1}$ of (t-1)-step joint policies and the value vectors $V_{i,t-1}^{\vec{\pi}}$ for all $\vec{\pi} \in \vec{\Pi}_{t-1}$, the first stage of the dynamic programming update exhaustively generates $\Pi_{i,t}$ using $\Pi_{i,t-1}$ for each agent i, which is the set of t-step local policies for agent i. Assuming tree representations for policies, the t-step local policy for agent i can be created by preparing $|A_i|$ root action nodes, and appending all possible combinations of (t-1)-step local policies to the observation edges of the root action node. The number of exhaustively generated t-step local policies will be $|\Pi_{i,t}| = |A_i||\Pi_{i,t-1}|^{|Z_i|}$. Combining $\Pi_{i,t}$ for all the agents yields the set of t-step joint policies $\vec{\Pi}_t$ with size $|\Pi_{1,t}||\Pi_{2,t}| \cdots |\Pi_{i,t}| \cdots |\Pi_{n,t}|$. The first stage of dynamic programming update is concluded by computing the values of joint policies, $V_{i,t}^{\vec{\pi}}$ for all $\vec{\pi} \in \vec{\Pi}_t$ and agent i, using Equation 5.

Require: $\Pi_{i,0} = \varnothing$ and $\mathcal{V}_{i,0} = \{\vec{0}\}$ (initial value function) for each agent i.
  for $t = 1, \ldots, T$ **do**
    # The first stage of dynamic programming backup
    for $i = 1, \ldots, n$ **do**
      Perform backup on (t-1)-step local policies $\Pi_{i,t-1}$ to produce the exhaustive set of t-step local policies $\Pi_{i,t}$.
    end for
    Let $\vec{\Pi}_t = \Pi_{1,t} \times \cdots \times \Pi_{i,t} \times \cdots \times \Pi_{n,t}$.
    for all $\vec{\pi} \in \vec{\Pi}_t$ **do**
      Compute $V_{i,t}^{\vec{\pi}}$ (Equation 5) and add the value vector to $\mathcal{V}_{i,t}$.
    end for
    # The second stage of dynamic programming backup
    repeat
      for $i = 1, \ldots, n$ **do**
        for all $\pi \in \Pi_{i,t}$ **do**
          Prune π if very weakly dominated (Equation 6)
        end for
      end for
    until no local policy was pruned in the loop
  end for
  return Sets of T-step policies $\Pi_{i,T}$ and corresponding value vectors $\mathcal{V}_{i,T}$ for each agent i

Table 2: The MADP algorithm

With all the necessary policy backup and value computation completed, the update continues to the second stage, where the *very weakly dominated policies* are pruned. A local policy π of agent i is said to be *weakly dominated* if the agent does not decrease its value by switching to some other local policy while all others maintain their own local policies, and there exists at least one $\vec{\pi}_{-i} \in \vec{\Pi}_{-i,t}$ such that switching away from π strictly increases agent i's value. A *very weakly dominated* policy is one where the weak dominance relation holds without the existence requirement of the strict improvement in the value. The test for very weak dominance of a local policy π of agent i can be determined by checking the existence of a probability distribution p on other policies $\Pi_{i,t} \backslash \pi$ such that

$$\sum_{\pi' \in \Pi_{i,t} \backslash \pi} p(\pi') V_{i,t}^{\{\pi', \vec{\pi}_{-i}\}}(s) \geq V_{i,t}^{\{\pi, \vec{\pi}_{-i}\}}(s), \ \ \forall s \in S, \forall \vec{\pi}_{-i} \in \vec{\Pi}_{-i,t} \ , \qquad (6)$$

where $V_{i,t}^{\{\pi, \vec{\pi}_{-i}\}}$ is the value vector of the joint policy formed by π for agent i and $\vec{\pi}_{-i}$ for other agents. If there exists such a distribution, then π is prunable since it is possible for agent i to take a *stochastic policy* determined by p, while achieving the value no worse than that of π. This test for dominance is carried out by linear programming (LP). A very weakly dominated policy can thus be safely pruned without any concern for the loss in value. The pruning proceeds in an iterated fashion where each agent alternately tests for dominance and prunes accordingly. This iteration stops when no

agent can prune any more local policies.

Table 2 outlines the MADP algorithm for computing the set of T-step joint policies. Note that this algorithm requires additional computation to select the joint policy depending on the solution concept such as Nash equilibrium. For DEC-POMDPs that assume cooperative settings, a joint policy with the maximum value for the initial state distribution b_0 is selected as an optimal joint policy.

4. Symmetries in POMDPs and POSGs

In this section, we show how symmetries are defined in POMDPs and POSGs. We show that finding symmetries for botch cases is a graph isomorphism complete (GI-complete) problem - the complexity class of finding automorphisms in general graphs. We present the graph encoding of a given POMDP and POSG in order to apply algorithms for finding graph automorphisms. We also describe how POMDP and POSG algorithms can be extended to exploit the symmetries discovered in the models.

4.1. Definition of Symmetries in POMDPs

There have been a number of works in the past to take advantage of the underlying structure in decision theoretic planning models. Perhaps one of the most extensively studied types of structural regularities would be that of homomorphism. It is directly related to abstraction and model minimization techniques that try to reduce the size of the model.

A *homomorphism* ϕ of a POMDP is defined as $\langle \phi_S, \phi_A, \phi_Z \rangle$ where $\phi_S : S \to S'$ is the function that maps the states, $\phi_A : A \to A'$ maps the actions, and $\phi_Z : Z \to Z'$ maps the observations. Note that the mapped POMDP $M' = \langle S', A', Z', T', O', R' \rangle$ is a *reduced model* of M if any of the mappings is *many-to-one*. Because of this property, model minimization methods for POMDPs search for a homomorphism ϕ that maps M to an equivalent POMDP M' with the minimal model size. Depending on the definition of homomorphism ϕ, we obtain different definitions of the minimal model.

A simple extension of MDP model minimization [10] to POMDPs leads to a homomorphism ϕ of form $\langle \phi_S, 1, 1 \rangle$, where 1 denotes the identity mapping. In order to hold equivalence between M and M', ϕ_S should satisfy the following constraints:

$$\begin{aligned} T'(\phi_S(s), a, \phi_S(s')) &= \textstyle\sum_{s'' \in \phi_S^{-1}(s')} T(s, a, s'') \\ R'(\phi_S(s), a) &= R(s, a) \\ O'(\phi_S(s), a, z) &= O(s, a, z) \end{aligned}$$

Pineau et al. [24] extend the approach to the case when a task hierarchy is given by an expert, and they achieve a further reduction in the state space since some of the actions become irrelevant under the task hierarchy.

Wolfe [36] extends the minimization method to compute homomorphism of a more general form $\langle \phi_S, \phi_A, \phi_Z \rangle$ where the observation mapping ϕ_Z can change depending

	$T(s, a_{\text{LISTEN}}, s')$		$T(s, a_{\text{LEFT}}, s')$		$T(s, a_{\text{RIGHT}}, s')$	
$s' =$	s_{LEFT}	s_{RIGHT}	s_{LEFT}	s_{RIGHT}	s_{LEFT}	s_{RIGHT}
$s = s_{\text{LEFT}}$	1.0	0.0	0.5	0.5	0.5	0.5
$s = s_{\text{RIGHT}}$	0.0	1.0	0.5	0.5	0.5	0.5

Figure 1: Transition probabilities of the tiger domain

on the action. The constraints for the equivalence are given by:

$$\begin{aligned} T'(\phi_S(s), \phi_A(a), \phi_S(s')) &= \textstyle\sum_{s'' \in \phi_S^{-1}(s')} T(s, a, s'') \\ R'(\phi_S(s), \phi_A(a)) &= R(s, a) \\ O'(\phi_S(s), \phi_A(a), \phi_Z^a(z)) &= O(s, a, z) \end{aligned}$$

Note that the above methods are interested in finding many-to-one mappings in order to find a model with reduced size. Hence, they focus on computing *partitions* of the state, action, and observation spaces of which blocks represent aggregates of equivalent states, actions, and observations, respectively. Once the partitions are found, we can employ conventional POMDP algorithms on the abstract POMDP with reduced number of states, actions, or observations, which in effect reduces the computational complexities of algorithms.

In this paper, we are interested in automorphism, which is a special class of homomorphism:

Definition 1. *An* automorphism ϕ *is defined as* $\langle \phi_S, \phi_A, \phi_Z \rangle$ *where the state mapping* $\phi_S : S \to S$*, the action mapping* $\phi_A : A \to A$*, and the observation mapping* $\phi_Z : Z \to Z$ *are all one-to-one mappings satisfying:*

$$\begin{aligned} T(s, a, s') &= T(\phi_S(s), \phi_A(a), \phi_S(s')) \\ O(s, a, z) &= O(\phi_S(s), \phi_A(a), \phi_Z(z)) \\ R(s, a) &= R(\phi_S(s), \phi_A(a)) \end{aligned}$$

Hence, ϕ maps the original POMDP to itself, and there is no assumption regarding the reduction in the size of the model.

The classic tiger domain [12] is perhaps one of the best examples to describe automorphisms in POMDPs. The state space S of the tiger domain is defined as $\{s_{\text{LEFT}}, s_{\text{RIGHT}}\}$, representing the state of the world when the tiger is behind the left door or the right door, respectively. The action space A is defined as $\{a_{\text{LEFT}}, a_{\text{RIGHT}}, a_{\text{LISTEN}}\}$, representing actions for opening the left door, opening the right door, or listening, respectively. The observation space Z is defined as $\{z_{\text{LEFT}}, z_{\text{RIGHT}}\}$ representing hearing the sound of the tiger from the left door or the right door, respectively. The specifications of transition probabilities, observation probabilities, and the rewards are as given in Figure 1, Figure 2, and Figure 3. The initial belief is given as $b_0(s_{\text{LEFT}}) = b_0(s_{\text{RIGHT}}) = 0.5$.

Note that the tiger domain is already compact in the sense that minimization methods previously mentioned cannot reduce the size of the model: examining the reward

	$O(s, a_{\text{LISTEN}}, z)$		$O(s, a_{\text{LEFT}}, z)$		$O(s, a_{\text{RIGHT}}, z)$	
$z =$	z_{LEFT}	z_{RIGHT}	z_{LEFT}	z_{RIGHT}	z_{LEFT}	z_{RIGHT}
$s = s_{\text{LEFT}}$	0.85	0.15	0.5	0.5	0.5	0.5
$s = s_{\text{RIGHT}}$	0.15	0.85	0.5	0.5	0.5	0.5

Figure 2: Observation probabilities of the tiger domain

	$R(s, a)$		
$a =$	a_{LISTEN}	a_{LEFT}	a_{RIGHT}
$s = s_{\text{LEFT}}$	-1	-100	10
$s = s_{\text{RIGHT}}$	-1	10	-100

Figure 3: Reward function of the tiger domain

function alone, we cannot aggregate a_{LEFT} and a_{RIGHT} since the rewards are different depending on the current state being either s_{LEFT} or s_{RIGHT}. By a similar argument, we cannot reduce the state space nor the observation space.

However, s_{LEFT} and s_{RIGHT} can be interchanged to yield an equivalent POMDP, while simultaneously changing the corresponding actions and observations:

$$\phi_S(s) = \begin{cases} s_{\text{RIGHT}} & \text{if } s = s_{\text{LEFT}} \\ s_{\text{LEFT}} & \text{if } s = s_{\text{RIGHT}} \end{cases}$$

$$\phi_A(a) = \begin{cases} a_{\text{LISTEN}} & \text{if } a = a_{\text{LISTEN}} \\ a_{\text{RIGHT}} & \text{if } a = a_{\text{LEFT}} \\ a_{\text{LEFT}} & \text{if } a = a_{\text{RIGHT}} \end{cases}$$

$$\phi_Z(z) = \begin{cases} z_{\text{RIGHT}} & \text{if } z = z_{\text{LEFT}} \\ z_{\text{LEFT}} & \text{if } z = z_{\text{RIGHT}} \end{cases}$$

Furthermore, this property yields symmetries in the belief states and α-vectors in the tiger domain, as can be seen in Figure 4.

The automorphism in POMDPs is the type of regularity we intend to discover and exploit in this paper: the symmetry in the model that does not necessarily help the model minimization algorithm further reduce the size of the model. Hence, rather than computing partitions, we focus on computing all possible automorphisms of the original POMDP.

Note that if the original POMDP can be reduced in size, we can have exponentially many automorphisms in the number of blocks in the partition. For example, if the model minimization yields a state partition with K blocks of 2 states each, the number of automorphisms becomes 2^K. Hence, it is advisable to compute automorphism after we compute the minimal model of POMDP.

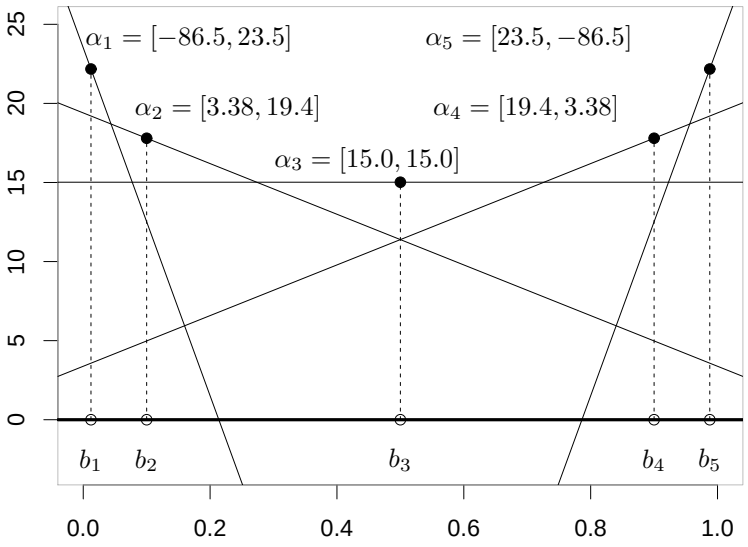

Figure 4: Value function of tiger domain obtained by PBVI with 5 belief states. b_1 and b_5 are symmetric, hence the corresponding α-vectors α_1 and α_5 are symmetric. The same argument applies to b_2 and b_4. Although the illustration uses an approximate value function computed by PBVI, the value function from exact methods will show the same phenomenon.

4.2. Properties of Symmetries in POMDPs

As shown in the tiger domain, the automorphisms of POMDPs reveal the symmetries present in belief states and α-vectors; Given a POMDP M with automorphism $\phi = \langle \phi_S, \phi_A, \phi_Z \rangle$, let Γ^* be the set of α-vectors for the optimal value function. In this setting, we provide the following two theorems that can be exploited when computing a solution to a given POMDP.

By a slight abuse of notation, for a vector v of dimension $|S|$, we let $\phi_S(v)$ be the transformed vector whose elements are permuted by ϕ_S.

Theorem 1. *If b is a reachable belief state, then $\phi_S(b)$ is also a reachable belief state.*

Proof. First, given $\phi = \langle \phi_S, \phi_A, \phi_Z \rangle$, note that

$$b_z^a(s) = b_{\phi_Z(z)}^{\phi_A(a)}(\phi_S(s)),$$

because the automorphism ensures that $T(s, a, s') = T(\phi_S(s), \phi_A(a), \phi_S(s'))$ and $O(s, a, z) = O(\phi_S(s), \phi_A(a), \phi_Z(z))$. This means that the symmetric image of a reachable belief vector b, that is, $\phi_S(b)$, is also reachable from the initial belief b_0 by executing a "symmetric policy", where the action a is mapped to $\phi_A(a)$.

In other words, if b is reachable from initial belief state b_0 by executing a policy tree, $\phi_S(b)$ can also be reached by executing the policy tree where action nodes are relabeled using $\phi_A(a)$ and the observation edges are relabeled using $\phi_Z(z)$. □

Theorem 2. *If $\alpha \in \Gamma^*$, then $\phi_S(\alpha) \in \Gamma^*$.*

Proof. We prove by induction on horizon t in Γ_t. By the definition of automorphism, $R(s,a) = R(\phi_S(s), \phi_A(a))$. Hence, if $\alpha \in \Gamma_0$ then $\phi_S(\alpha) \in \Gamma_0$.

Suppose that the argument holds for Γ_{t-1}. This implies that $\forall \alpha \in \Gamma_t^{a,z}$, $\phi_S(\alpha) \in \Gamma_t^{\phi_A(a),\phi_Z(z)}$ by the definition of $\Gamma_t^{a,z}$. If $\alpha \in \Gamma_t$, then by definition, for some a and b,

$$\alpha(s) = \alpha_b^a(s) = R(s,a) + \sum_{z \in Z} \operatorname*{argmax}_{\alpha' \in \Gamma_t^{a,z}} (\alpha' \cdot b)$$

Consider its symmetric image defined as

$$\alpha(\phi_S(s)) = R(\phi_S(s), \phi_A(a)) + \sum_{\phi_Z(z) \in Z} \operatorname*{argmax}_{\alpha'' \in \Gamma_t^{\phi_A(a),\phi_Z(z)}} (\alpha'' \cdot \phi_S(b)).$$

For each observation $\phi_Z(z)$, the argmax will select α'' which is the symmetric image of α' selected in the $\operatorname{argmax}_{\alpha' \in \Gamma_t^{a,z}} (\alpha' \cdot b)$. Hence we have $\phi_S(\alpha) \in \Gamma_t$. □

In this work, we specialize the PBVI algorithm to exploit symmetries, as will be shown in later sections. However, the theorems we provide are general enough to be applied to a variety of different value function-based algorithms. We argue so, because the unifying theme of all value function-based algorithms is the dependence on α-vectors and/or belief points, and the two theorems we presented indicate that the symmetric images of the sampled belief points and α-vectors contribute equivalently to the overall value. For example, the randomized point-based backup of Perseus [32] can benefit from our results by not having to perform redundant backup operation on symmetric beliefs. Symmetries can be exploited in search based methods such as HSVI or Forward Search Value Iteration (FSVI) [29] in a similar manner. In particular, multiple backups can be performed for a single sampled belief point by taking the symmetric image of that sampled belief. The gist is that, while different value function-based methods provide different sampling approaches, our framework can be universally applied to enhance the sampling procedure.

4.3. Definition of Symmetries in POSGs

Extending the definition to POSGs introduces agent-to-agent mappings, where the local actions and observations of an agent are mapped to those of another agent. Formally, the automorphism for POSGs is defined as follows:

Definition 2. *An automorphism ϕ for agent i on a POSG is a tuple $\langle \phi_I, \phi_S, \phi_{\vec{A}}, \phi_{\vec{Z}} \rangle$ with $\phi_{\vec{A}} = \{\phi_{A_i} | i \in I\}$ and $\phi_{\vec{Z}} = \{\phi_{Z_i} | i \in I\}$, where agent mapping $\phi_I : I \to I$, state mapping $\phi_S : S \to S$, action mappings $\phi_{A_i} : A_i \to A_{\phi_I(i)}$, and observation mappings $\phi_{Z_i} : Z_i \to Z_{\phi_I(i)}$ are all bijections satisfying*

$$T(s, \vec{a}, s') = T(\phi_S(s), \phi_{\vec{A}}(\vec{a}), \phi_S(s'))$$
$$O(s, \vec{a}, \vec{z}) = O(\phi_S(s), \phi_{\vec{A}}(\vec{a}), \phi_{\vec{Z}}(\vec{z}))$$
$$R_i(s, \vec{a}) = R_{\phi_I(i)}(\phi_S(s), \phi_{\vec{A}}(\vec{a}))$$

for all s, s', $\vec{a}$, and $\vec{z}$.

Joint Action	$s_{\text{LEFT}} \rightarrow s_{\text{RIGHT}}$	$s_{\text{LEFT}} \rightarrow s_{\text{LEFT}}$	$s_{\text{RIGHT}} \rightarrow s_{\text{LEFT}}$	$s_{\text{RIGHT}} \rightarrow s_{\text{RIGHT}}$
$\{a_{1,\text{LISTEN}}, a_{2,\text{LISTEN}}\}$	0	1	0	1
Other	0.5	0.5	0.5	0.5

Figure 5: State transition probabilities of the Dec-Tiger domain. The second row shows the transition probabilities of all joint actions composed of at least one non-listen individual action.

A special case when agent mapping ϕ_I is an identity mapping, ϕ is said to be an *intra-agent automorphism*. On the other hand, if ϕ_I is a non-identity mapping, it is said to be an *inter-agent automorphism*. Informally speaking, inter-agent automorphism allows interchanging agents as long as the local actions and observations are interchanged accordingly. On the other hand, intra-agent automorphism is confined to interchanging the local actions and observations within an agent. It can be thought that intra-agent automorphism captures the symmetry present in the single-agent POMDP level, while the inter-agent automorphism extends the symmetry to the multi-agent level.

To illustrate, we present the decentralized tiger (Dec-Tiger) domain [19]. Dec-Tiger is a multi-agent extension to the classical tiger domain. There are now two agents, setting the agent set $I = \{1, 2\}$, that must make a sequence of decisions as to whether they should open the door (jointly or separately) or listen. The states are the same as the tiger domain: s_{LEFT} and s_{RIGHT}. Each agent has the same set of actions that are equivalent to the single agent case: $\{a_{i,\text{LISTEN}}, a_{i,\text{RIGHT}}, a_{i,\text{LEFT}} | i = 1 \text{ or } 2\}$, where $a_{i,X}$ indicates the action X of agent i. The observation space is duplicated from the single-agent case as well: $\{z_{i,\text{LEFT}}, z_{i,\text{RIGHT}} | \ i = 1 \text{ or } 2\}$, with the notations defined similarly.

If at least one agent performs an open action, the state resets to either one with 0.5 probability. If both continue with a listen action, then there is no change of state.

Each agent individually observes the tiger from the correct room with probability 0.85 when performing a listen action. When both agents perform a joint listen action, then the resulting joint observation probability is computed as a product of the individual probabilities. All other joint actions where at least one agent performs a non-listen action result in a uniform distribution over the joint observations.

Rewards are given equally to both agents, and are designed to encourage cooperation. The maximum reward can be attained by cooperatively opening the tiger-free room. If each agent chooses a different room, then a high penalty is given. If they cooperatively open the tiger room, then they still suffer a penalty, but at a much lesser cost. Jointly listening costs a small penalty, whereas opening the tiger-free room while the other agent listens will result in a small reward. If, on the other hand, one agent opens the tiger room while the other is listening, then they incur the worst possible penalty. The transition probabilities, observation probabilities, and rewards are summarized in Figure 5, Figure 6, and Figure 7, respectively.

One possible symmetry that exhibits an inter-agent mapping is presented in Figure 8. For the complete list of symmetries in Dec-Tiger, we invite the reader to consult Figure 15 in Section 7 where we report experimental results.

Joint Observation	s_{LEFT}	s_{RIGHT}
$\{z_{1,\text{LEFT}}, z_{2,\text{LEFT}}\}$	0.7225	0.0225
$\{z_{1,\text{LEFT}}, z_{2,\text{RIGHT}}\}$	0.1275	0.1275
$\{z_{1,\text{RIGHT}}, z_{2,\text{LEFT}}\}$	0.1275	0.1275
$\{z_{1,\text{RIGHT}}, z_{2,\text{RIGHT}}\}$	0.0225	0.7225

Figure 6: Observation probabilities of the Dec-Tiger domain for joint action $\{a_{1,\text{LISTEN}},a_{2,\text{LISTEN}}\}$. The probabilities for other joint actions are uniform, and are omitted.

Joint Action	s_{LEFT}	s_{RIGHT}
$\{a_{1,\text{RIGHT}}, a_{2,\text{RIGHT}}\}$	20,20	0,0
$\{a_{1,\text{LEFT}}, a_{2,\text{LEFT}}\}$	0,0	20,20
$\{a_{1,\text{RIGHT}}, a_{2,\text{LEFT}}\}$	-100,-100	-100,-100
$\{a_{1,\text{LEFT}}, a_{2,\text{RIGHT}}\}$	-100,-100	-100,-100
$\{a_{1,\text{LISTEN}}, a_{2,\text{LISTEN}}\}$	-2,-2	-2,-2
$\{a_{1,\text{LISTEN}}, a_{2,\text{RIGHT}}\}$	9,9	-101,-101
$\{a_{1,\text{RIGHT}}, a_{2,\text{LISTEN}}\}$	9,9	-101,-101
$\{a_{1,\text{LISTEN}}, a_{2,\text{LEFT}}\}$	-101,-101	9,9
$\{a_{1,\text{LEFT}}, a_{2,\text{LISTEN}}\}$	-101,-101	9,9

Figure 7: Individual rewards of the Dec-Tiger domain

4.4. Properties of Symmetries in POSGs

As with the case with POMDPs, the symmetries in POSGs reveal useful regularities present in the model. In this section, we formally state the properties of symmetries in POSGs, which will be used to extend MADP in the later sections. Again, with a slight abuse of notation, we extend the domain of ϕ to local and joint policy trees, the output of which is another policy tree with all the actions and observations permuted accordingly. That is, $\phi(\pi)$ for any policy tree π is a permuted policy tree whose action nodes have been mapped by π_A and the observation edges have been permuted by π_Z.

Theorem 3. *Given an automorphism* $\phi = \langle \phi_I, \phi_S, \phi_{\vec{A}}, \phi_{\vec{Z}} \rangle$,

$$V^{\vec{\pi}}_{i,t}(s) = V^{\phi(\vec{\pi})}_{\phi_I(i),t}(\phi_S(s))$$

for all s *at all time steps* $1 \leq t \leq T$.

Proof. We prove by induction on t. For $t = 1$, only the immediate reward matters:

$$V^{\vec{\pi}}_{i,1}(s) = R_i(s, \vec{a}) = R_{\phi_I(i)}(\phi_S(s), \phi_{\vec{A}}(\vec{a})) = V^{\phi(\vec{\pi})}_{\phi_I(i),1}(\phi_S(s)).$$

The first and last equalities follow from the fact that a 1-step policy tree is simply a single action node. The second equality holds by the definition of automorphism.

$$\phi_I(i) = \begin{cases} \text{Agent 2} & \text{if } i \text{ is Agent 1} \\ \text{Agent 1} & \text{if } i \text{ is Agent 2} \end{cases}$$

$$\phi_S(s) = \text{Identity mapping}$$

$$\phi_{A_1}(a) = \begin{cases} a_{2,\text{LISTEN}} & \text{if } a = a_{1,\text{LISTEN}} \\ a_{2,\text{RIGHT}} & \text{if } a = a_{1,\text{RIGHT}} \\ a_{2,\text{LEFT}} & \text{if } a = a_{1,\text{LEFT}} \end{cases}$$

$$\phi_{A_2}(a) = \begin{cases} a_{1,\text{LISTEN}} & \text{if } a = a_{2,\text{LISTEN}} \\ a_{1,\text{RIGHT}} & \text{if } a = a_{2,\text{RIGHT}} \\ a_{1,\text{LEFT}} & \text{if } a = a_{2,\text{LEFT}} \end{cases}$$

$$\phi_{Z_1}(z) = \begin{cases} z_{2,\text{RIGHT}} & \text{if } z = z_{1,\text{LEFT}} \\ z_{2,\text{LEFT}} & \text{if } z = z_{1,\text{RIGHT}} \end{cases}$$

$$\phi_{Z_2}(z) = \begin{cases} z_{1,\text{RIGHT}} & \text{if } z = z_{2,\text{LEFT}} \\ z_{1,\text{LEFT}} & \text{if } z = z_{2,\text{RIGHT}} \end{cases}$$

Figure 8: An example of an inter-agent symmetry for Dec-Tiger

Assume that the theorem holds for all t's up to $t = k$-1 (*i.e.* for policy trees of depth $k-1$). For $t = k$, the Bellman equation unfolds as

$$\begin{aligned} & R_i(s,\vec{a}) + \gamma \sum_{s' \in S, \vec{z} \in \vec{Z}} T(s,\vec{a},s') O(s',\vec{a},\vec{z}) V_{i,k-1}^{\vec{\pi}(\vec{z})}(s') \\ & \qquad = R_{\phi_I(i)}(\phi_S(s), \phi_{\vec{A}}(\vec{a})) + \gamma \sum_{\substack{\phi_S(s') \in S \\ \phi_{\vec{Z}}(\vec{z}) \in \vec{Z}}} \begin{pmatrix} T(\phi_S(s), \phi_{\vec{A}}(\vec{a}), \phi_S(s')) \\ \cdot O(\phi_S(s'), \phi_{\vec{A}}(\vec{a}), \phi_{\vec{Z}}(\vec{z})) \\ \cdot V_{\phi_I(i),k-1}^{\phi(\vec{\pi}(\phi_{\vec{Z}}(\vec{z})))}(\phi_S(s')) \end{pmatrix}. \end{aligned}$$

All the terms except the $V(\cdot)$ can be shown equal by the definition of automorphism. The equality of the next-step value term is established by the inductive hypothesis, since the subtrees (all of which are $(k-1)$-level subtrees) encountered by following $\vec{z}$ in $\vec{\pi}$ are symmetric to the ones encountered by following $\phi_{\vec{Z}}(\vec{z})$ in $\phi(\vec{\pi})$. Therefore, the equality holds for all $t \geq 1$. □

Because Theorem 3 holds for all values of t, we will henceforth drop the horizon superscript t whenever possible. Based on the above theorem, we can make the following statement regarding very weak dominance under the presence of symmetries:

Theorem 4. *If the local policy π of agent i is very weakly dominated, then the local policy $\phi(\pi)$ of agent $\phi_I(i)$ is also very weakly dominated for any automorphism ϕ.*

Proof. From Equation 6, the local policy π of agent i is very weakly dominated if there exists a probability distribution p on other local policies $\Pi_i \backslash \pi$ such that

$$\sum_{\pi' \in \Pi_i \backslash \pi} p(\pi') V_i^{\{\pi', \vec{\pi}_{-i}\}}(s) \geq V_i^{\{\pi, \vec{\pi}_{-i}\}}(s), \quad \forall s, \forall \vec{\pi}_{-i} \in \vec{\Pi}_{-i}.$$

Consider the local policy $\phi(\pi)$ of agent $\phi_I(i)$. In order to prove that $\phi(\pi)$ is very weakly dominated, we have to show that there exists a probability distribution p' on agent $\phi_I(i)$'s other local policies $\Pi_{\phi_I(i)} \backslash \phi(\pi)$ such that

$$\sum_{\pi'' \in \Pi_{\phi_I(i)} \backslash \phi(\pi)} p'(\pi'') V_{\phi_I(i)}^{\{\pi'', \vec{\pi}_{-\phi_I(i)}\}}(s) \geq V_{\phi_I(i)}^{\{\phi(\pi), \vec{\pi}_{-\phi_I(i)}\}}(s), \forall s, \forall \vec{\pi}_{-\phi_I(i)} \in \vec{\Pi}_{-\phi_I(i)}.$$

Note that the local policy π' of agent i corresponds to the local policy $\phi(\pi')$ of agent $\phi_I(i)$. Hence for each $\pi'' \in \Pi_{\phi_I(i)} \backslash \phi(\pi)$, we can always find $\pi' \in \Pi_i \backslash \pi$ such that $\pi'' = \phi(\pi')$ since ϕ is bijective. If we set $p'(\pi'') = p(\pi')$ where $\pi'' = \phi(\pi')$, we have found a probability distribution p' that satisfies the above inequality. □

From Theorem 4, it follows that a policy tree and all of its symmetric images can be pruned without loss in the value if any of them is known to be very weakly dominated:

Corollary 1. *If a policy π can be pruned, then $\phi(\pi)$ can be pruned as well.*

As in the case of POMDPs, we adopt MADP to demonstrate the usefulness of symmetries in POSGs. While this approach may seem algorithm-specific, we argue that the theoretical basis on which such exploitations are made is general enough to be applied to other algorithms as well.

For example, there has been much significant work on solving DEC-POMDPs in recent years, including Bounded Policy Iteration (BPI) [4], Memory-Bounded Dynamic Programming (MBDP) [28], Heuristic Policy Iteration (HPI) [2], Point-Based Bounded Policy Iteration (PB-BPI) [14], Point-Based Policy Generation (PBPG) [37], Constraint Based Policy Backup (CBPB) and Team Decision problem based Policy Iteration (TDPI) [15]. These algorithms often share common computational steps, such as exhaustive or partial dynamic programming backup of policies, pruning dominated policies and improving policies using mathematical programming. The theoretical results above can be used to reduce the number of policies generated by the dynamic programming backup, as well as the number of mathematical programs to solve. We can also apply recent results on exploiting symmetries to reduce the sizes of mathematical programs themselves [5], but the details are left for future work.

The symmetries also have various impacts on the game theoretic analysis of the given POSG. To facilitate our discussion, we will convert the given POSG to a normal form game. We will also adhere to the term "policy" for the sake of consistency, although "strategy" is more widely adopted in game theory. As pointed out by Hansen et al. [11], a POSG at time horizon t can be converted to a normal form game by enlisting all the policy trees as possible actions. We also include the initial state distribution in order to have scalar payoffs rather than $|S|$-dimensional vector payoffs. This is done by taking the inner product of each value vector $V_i^{\vec{\pi}}$ and the initial state distribution b_0. This inner product will become the payoff entry into our converted game.

We denote the payoff of a joint policy $\vec{\pi}$ for agent i as $u_i(\vec{\pi})$, or equivalently, $u_i(\{\pi_i, \vec{\pi}_{-i}\})$. It follows that $u_i(\vec{\pi}) = u_{\phi_I(i)}(\phi(\vec{\pi}))$, due to Equation 7.

$$\sum_s b_0(s) V_i^{\vec{\pi}}(s) = \sum_s b_0(\phi_S(s)) V_{\phi_I(i)}^{\phi(\vec{\pi})}(\phi_S(s)) \tag{7}$$

For our discussion on symmetries for game theoretic solution concepts, we begin with the *Nash equilibrium.* A (pure-strategy) Nash equilibrium is a joint policy such that for any fixed agent, that agent has no incentive to unilaterally switch its policy provided that others do not change theirs.

Proposition 1. *If a joint policy $\vec{\pi}$ is a Nash equilibrium in a normal form representation of the given POSG, then its symmetric image $\phi(\vec{\pi})$ also constitutes a Nash equilibrium.*

Proof. Given a Nash equilibrium $\vec{\pi}^*$, the following inequality holds by the definition:

$$u_i(\{\pi_i^*, \vec{\pi}_{-i}^*\}) \geq u_i(\{\pi_i, \vec{\pi}_{-i}^*\}), \forall i, \pi_i \neq \pi_i^*.$$

The automorphism guarantees $u_i(\vec{\pi}) = u_{\phi_I(i)}(\phi(\vec{\pi}))$, for any joint policy $\vec{\pi}$. Therefore, $u_{\phi_I(i)}(\phi(\vec{\pi}^*)) \geq u_{\phi_I(i)}(\{\phi(\pi_i), \phi(\vec{\pi}_{-i}^*)\}),\ \forall i, \pi_i \neq \pi_i^*$, which establishes the fact that $\phi(\vec{\pi}^*)$ is a Nash equilibrium as well. □

Proposition 1 easily generalizes to mixed-strategy Nash equilibrium. Note that our notion of symmetries generalize the definition used in classical *symmetric games*, which requires that there exists an invariant action mapping ϕ_A and observation mapping ϕ_Z for all possible permutations of agents. Our theoretical results could be used in making game solvers more scalable, widening the applicability of the techniques by Cheng et al. [7]. The facts presented in this section lead to a more efficient procedure for finding the equilibria of symmetric POSGs. Instead of searching for every single equilibrium present in POSGs, we can speed up the process by applying the symmetries of the POSGs to the equilibria that have already been discovered.

The *correlated equilibrium* (CE) [21] generalizes the mixed-strategy Nash equilibrium. Whereas the mixed-strategy Nash equilibrium is defined to be an independent probability over the local policies, the CE is a probability over the joint policies allowing for the dependencies among agents' local policies. That is, the probability p over the joint policies is a CE if

$$\textstyle\sum_{\vec{\pi}_{-i}} p(\vec{\pi}) u_i(\vec{\pi}) \geq \sum_{\vec{\pi}_{-i}} p(\vec{\pi}) u_i(\{\pi_i', \vec{\pi}_{-i}\}),\ \forall i \text{ and } \forall \pi_i' \neq \pi_i. \tag{8}$$

With symmetries present in the normal form game representation of the POSG, we can prove a symmetric property of a CE.

Proposition 2. *Let p be a CE of the normal form representation of a given POSG. Then there exists a (possibly same) CE p' such that $p'(\phi(\vec{\pi})) = p(\vec{\pi})$ for any automorphism ϕ of the given POSG, and any joint policy $\vec{\pi}$.*

Proof. Given a CE p, we can re-write Equation 8 as

$$\textstyle\sum_{\vec{\pi}_{-\phi_I(i)}} p(\vec{\pi}) u_{\phi_I(i)}(\phi(\vec{\pi})) \geq \sum_{\vec{\pi}_{-\phi_I(i)}} p(\vec{\pi}) u_{\phi_I(i)}(\{\phi(\pi_i'), \phi(\vec{\pi}_{-i})\}), \forall i, \forall \pi_i' \neq \pi_i.$$

Note that since $\pi'_i \neq \pi_i$, $\phi(\pi'_i) \neq \phi(\pi_i)$ due to ϕ being bijective. This modified form states that $p(\vec{\pi})$ can also be used as a probability with which $\phi(\vec{\pi})$ is chosen. Therefore, there exists a CE that assigns probability $p(\vec{\pi})$ to $\phi(\vec{\pi})$. □

5. Symmetry Discovery in the Models

In this section, we show that finding the symmetries present in POMDPs and POSGs is a graph isomorphism (GI) complete problem, the computational complexity class of finding the automorphism groups of general graphs. We thus present the graph encoding of a given POMDP and POSG in order to use a graph automorphism algorithm for finding symmetries in the model.

5.1. Graph Encoding of a POMDP

We first describe how we can cast the problem of finding automorphisms in POMDPs as that of finding automorphisms in graphs. Specifically, we will show how we can encode a given POMDP as a vertex-colored graph, so that the automorphism found in the graph corresponds to the automorphism in the POMDP. Our approach here will prove useful when we discuss the computational complexity of discovering POMDP automorphisms in the later part of this section.

A vertex-colored graph G is specified by $\langle V, E, C, \psi \rangle$, where V denotes the set of vertices, E denotes the set of edges $\langle v_i, v_j \rangle$, C is the set of colors, and $\psi : V \rightarrow C$ denotes the color associated with each vertex. An automorphism $\phi : V \rightarrow V$ is a permutation of V with the property that for any edge $\langle v_i, v_j \rangle \in E$, $\langle \phi(v_i), \phi(v_j) \rangle$ is also in E, and for any vertex $v_i \in V$, $\psi(v_i) = \psi(\phi(v_i))$.

We can encode a POMDP as a vertex-colored graph in order to apply graph automorphism algorithms. The encoded graph is composed of the following classes of vertices and edges, their counts being presented in parentheses:

- States ($|S|$ vertices): for every state s, we prepare vertex v_s and make every vertex share the same unique color: $\forall s \in S$, $\psi(v_s) = c_{\text{state}}$.
- Actions ($|A|$ vertices): for every action a, we prepare vertex v_a and make every vertex share the same unique color: $\forall a \in A$, $\psi(v_a) = c_{\text{action}}$.
- Next states ($|S|$ vertices and $|S|$ edges): for every state s', we prepare vertex $v_{s'}$ and make every vertex share the same unique color: $\forall s' \in S$, $\psi(v_{s'}) = c_{\text{state'}}$. We connect the next-state vertex $v_{s'}$ to the state vertex v_s if and only if $s' = s$.
- Observations ($|Z|$ vertices): for every observation z, we prepare vertex v_z and make every vertex share the same unique color: $\forall z \in Z$, $\psi(v_z) = c_{\text{obs}}$.
- Transition probabilities ($|S|^2|A|$ vertices and $3|S|^2|A|$ edges): for every triplet (s, a, s'), we prepare vertex $v_{T(s,a,s')}$ that represents the transition probability $T(s, a, s')$ and assign colors so that two vertices share the same color if and only if the transition probabilities are the same: $\forall (s, a, s')$, $\forall (s'', a', s''')$, $\psi(v_{T(s,a,s')}) = \psi(v_{T(s'',a',s'')})$ iff $T(s, a, s') = T(s'', a', s''')$. We connect the the transition probability vertex $v_{T(s,a,s')}$ to the corresponding state, action, and next-state vertices, v_s, v_a and $v_{s'}$.

- Observation probabilities ($|S||A||Z|$ vertices and $3|S||A||Z|$ edges): for every triplet (s, a, z), we prepare vertex $v_{O(s,a,z)}$ that represents the observation probability $O(s, a, z)$ and assign colors so that two vertices share the same color if and only if the observation probabilities are the same: $\forall(s, a, z)$, $\forall(s', a', z')$, $\psi(v_{O(s,a,z)}) = \psi(v_{O(s',a',z')})$ iff $O(s, a, z) = O(s', a', z')$. We connect the observation probability vertex $v_{O(s,a,z)}$ to the corresponding state, action, and observation vertices, v_s, v_a and v_z.

- Reward function ($|S||A|$ vertices and $2|S||A|$ edges): for every pair (s, a), we prepare vertex $v_{R(s,a)}$ that represents the reward $R(s, a)$ and assign colors so that two vertices share the same color if and only if the rewards are the same: $\forall(s, a), \forall(s', a')$, $\psi(v_{R(s,a)}) = \psi(v_{R(s',a')})$ iff $R(s, a) = R(s', a')$. We connect the reward vertex $v_{R(s,a)}$ to the corresponding state and action vertices, v_s and v_a.

- Initial state distribution ($|S|$ vertices and $|S|$ edges): for every state s, we prepare vertex $v_{b_0(s)}$ that represents the initial state probability $b_0(s)$ and assign colors so that two vertices share the same color if and only if the initial state probabilities are the same: $\forall s, \forall s'$, $\psi(v_{b_0(s)}) = \psi(v_{b_0(s')})$ iff $b_0(s) = b_0(s')$. We connect the initial state probability vertex $v_{b_0(s)}$ to the corresponding state vertex v_s.

The graph encoding process is mechanical, and the colors and edges are carefully prepared in order to preserve the equivalence of the model under any graph automorphism. Figure 9 shows the result of the graph encoding process for the tiger domain.

The encoded graph is sparse, consisting of $O(|S|^2|A||Z|)$ vertices and $O(|S|^2|A||Z|)$ edges, hence the number of edges is linear in the number of vertices. Despite super-polynomial running time in the worst case, typical graph automorphism solvers are efficient for sparse graphs. As we report in Section 7, we used nauty [18] for the graph automorphism solver, and it quickly found automorphisms in the encoded graphs of benchmark POMDP domains with up to 6×10^7 vertices.

As a minor remark, note that we choose the colors such that $\psi(v_{T(s,a,s')}) \neq \psi(v_{O(s,a,z)})$ even if $T(s, a, s') = O(s, a, z)$. This is to prevent the transition probability being permuted with observation probability vertices. Similar restrictions apply to all other vertices of different classes.

5.2. Graph Encoding of a POSG

Similar to the POMDP case, the problem of finding POSG automorphisms can be reduced to finding the automorphism group of a properly encoded graph. The graph encoding we use here is not so much different from the POMDP approach, with the exception of the vertices that reflect the multi-agent aspects.

The encoded graph is composed of the following classes of vertices and their edges:

- Agents ($|I|$ vertices): we prepare one vertex per agent, assigning the same unique color.

- States ($|S|$ vertices): we prepare one vertex per state, assigning the same unique color.

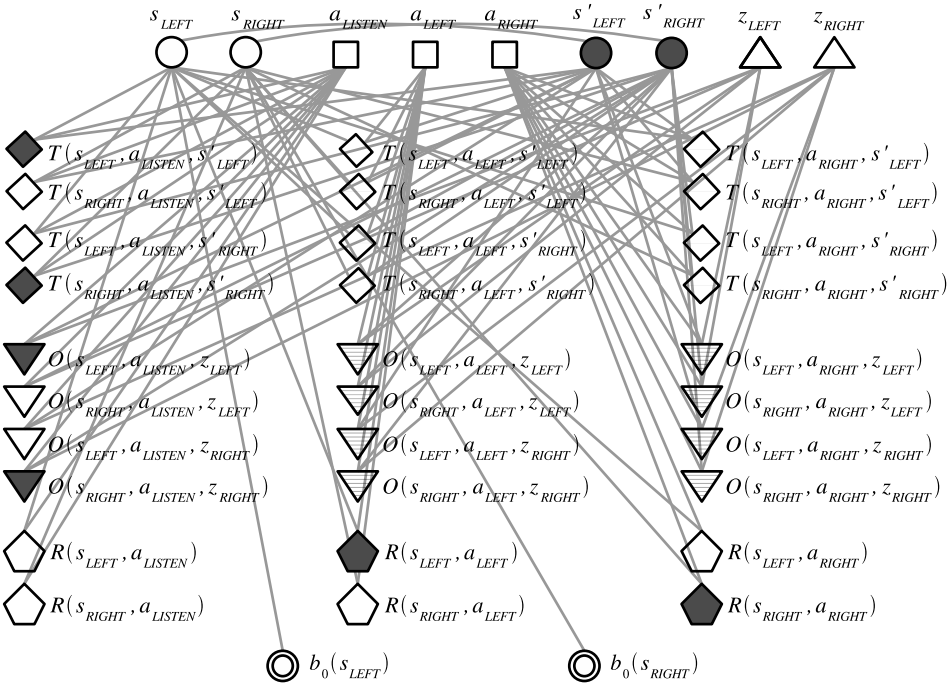

Figure 9: Encoding the tiger domain as a vertex-colored graph. Two vertices have the same color if and only if their shapes and fillings are the same.

- Next states ($|S|$ vertices and $|S|$ edges): we prepare another vertex per state, assigning the same unique color, however different from the color of state vertices. We connect each next-state vertex to the corresponding state vertex, so that permuting state vertices yields permuting next-state vertices in the same order.
- Actions ($\sum_i |A_i|$ vertices and $\sum_i |A_i|$ edges): we prepare one vertex per action, assigning the same unique color. We connect each action vertex to the corresponding agent vertex to represent to which agent the action is available.
- Observations ($\sum_i |Z_i|$ vertices and $\sum_i |Z_i|$ edges): we prepare one vertex per observation, assigning the same unique color. We connect each observation vertex to the corresponding agent vertex to represent to which agent the observation is available.
- Transition probabilities ($|S|^2 \prod_i |A_i|$ vertices and $(|I|+2)|S|^2 \prod_i |A_i|$ edges): we prepare one vertex per transition probability, assigning the same unique color if and only if they have the same probability. We connect each transition probability vertex to the corresponding state, next-state, and action vertices.
- Observation probabilities ($|S| \prod_i |A_i||Z_i|$ vertices and $(2|I|+1)|S| \prod_i |A_i||Z_i|$ edges): we prepare one vertex per observation probability, assigning the same

color if and only if they have the same probability. We connect each observation probability vertex to the corresponding state, action, and observation vertices.

- Individual reward functions ($|I||S|\prod_i |A_i|$ vertices and $(|I|+2)|I||S|\prod_i |A_i|$ edges): we prepare one vertex per individual reward, assigning the same color if and only if they have the same reward. We connect each reward vertex to the corresponding agent, state, and action vertices.

- Initial state distribution ($|S|$ vertices and $|S|$ edges): we prepare the vertices corresponding to $v_{b0}(s)$ the same way as they are for POMDPs.

The resulting graph has $O(|I||S|^2\prod_i |A_i||Z_i|)$ vertices and $O(|I|^2|S|^2\prod_i |A_i||Z_i|)$ edges. For DEC-POMDPs where the agents share the same reward function, there will be $O(|I||S|^2\prod_i |A_i||Z_i|)$ edges so that the number of edges is linear in the number of vertices.

5.3. Computational Complexity

A recent study on the computational complexity of finding MDP symmetries [20] showed that the problem of finding the symmetries of a given MDP can be polynomially reduced to the problem of finding the automorphisms of the corresponding graph encoding. Hence, it is known that the computational complexity of finding the symmetries of an MDP belongs to the graph isomorphism-complete (GI-complete) class. In this section, we extend the result on MDPs to POMDPs and POSGs, taking a similar but slightly different approach.

For ease of exposition, we provide two lemmas that will be useful in proving the main theorem regarding the results for POSGs. We will use the following definitions for the proof in the first lemma:

Definition 3. *Given POSG M, G_M denotes the vertex-colored undirected graph representation of M. The* model vertices *of G_M are the vertices corresponding to the state, action, observation, and agents of M. The* parameter vertices *of G_M are the ones corresponding to transition, observation, and reward functions of M.*

We also adjust notations regarding symmetries in order to prevent confusion: Symmetries pertaining to graphs will be denoted as ϕ_G with a G subscript, whereas symmetries of POSGs will retain the notations introduced in Definitions 1 and 2.

Lemma 1. *A symmetry of M corresponds to a unique automorphism of G_M, and vice versa.*

Proof. First, assume that a symmetry ϕ of M is given. From this, we can prove the existence of a unique automorphism ϕ_G of G_M. To construct a unique ϕ_G from ϕ, proceed by first mapping the model vertices according to ϕ. For example, given an action vertex v_{a_i}, we set $\phi_G(v_{a_i}) \leftarrow v_{\phi_{A_{\phi(i)}}(a_{\phi(i)})}$. Note that mapping the agent vertices simultaneously still maintains the edge connectivity because their corresponding action and observation vertices are mapped accordingly. Next, we permute the parameter vertices that are connected to the model vertices. This latter permutation must be possible because ϕ preserves the probabilities and rewards (whose corresponding

vertex colors are the same). To specify the permutations of the parameter vertices, consider a pair of tuples $t_1 = \langle s, \vec{a}, \vec{z}, s' \rangle$ and $t_2 = \langle \phi_S(s), \phi_{\vec{A}}(\vec{a}), \phi_{\vec{Z}}(\vec{z}), \phi_S(s') \rangle$. By construction of G_M, the vertices $v_{T(s,\vec{a},s')}$ and $v_{T(\phi_S(s),\phi_{\vec{A}}(\vec{a}),\phi_S(s'))}$, corresponding to $T(s, \vec{a}, s')$ and $T(\phi_S(s), \phi_{\vec{A}}(\vec{a}), \phi_S(s'))$, respectively, share the same color since the two probabilities are equal under ϕ. This assertion holds for any arbitrary choice of t_1, by definition of POSG symmetry. The only components connected to the relevant parameter vertices are the participating model vertices. Therefore, $\phi_G(v_{T(s,\vec{a},s')}) \leftarrow v_{T(\phi_S(s),\phi_{\vec{A}}(\vec{a}),\phi_S(s'))}$ preserves the color and edge constraints of graph automorphism. The same argument applies to the observation probability and reward vertices, completing the construction of ϕ_G. The construction of ϕ_G is tailored to a specific ϕ, and is different for some other $\phi' \neq \phi$, since the POSG symmetries are bijections. Thus, there is only one ϕ_G for a specific ϕ.

To show the other direction, assume we are given a ϕ_G of G_M. Consider two tuples $t_1 = \langle v_s, \{v_{a_i}\}, \{v_{z_i}\}, v_{s'} \rangle$ and $t_2 = \langle \phi_G(v_s), \{\phi_G(v_{a_i})\}, \{\phi_G(v_{z_i})\}, \phi_G(v_{s'}) \rangle$. The set of vertices for both tuples run over all agents $i \in I$. The vertices $v_s, v_{s'}, \{v_{a_i}\}$ of t_1 are connected to a transition probability vertex $v_{T(s,\vec{a},s')}$ that corresponds to $T(s, \vec{a}, s')$, where $\vec{a}$ is the joint action formed by concatenating the actions corresponding to the vertices $\{v_{a_i}\}$. The analogue holds for another transition probability vertex $v_{T(\phi_S(s),\phi_{\vec{A}}(\vec{a}),\phi_S(s'))}$ of t_2. Because there can be only one $v_{T(s,\vec{a},s')}$ for the triple $(s, \vec{a}, s')$, only the vertices in t_2 are connected to $v_{T(\phi_S(s),\phi_{\vec{A}}(\vec{a}),\phi_S(s'))}$, and no other. It follows that the following two equalities hold:

$$\begin{aligned} \phi_G(v_{T(s,\vec{a},s')}) &= v_{T(\phi_S(s),\phi_{\vec{A}}(\vec{a}),\phi_S(s'))} \\ \psi(\phi_G(v_{T(s,\vec{a},s')})) &= \psi(v_{T(\phi_S(s),\phi_{\vec{A}}(\vec{a}),\phi_S(s'))}) \end{aligned}$$

If this were not true, there must exist another vertex $v_{T''}$ that is mapped to $v_{T(s,\vec{a},s')}$. Then, by the property of graph automorphism, the corresponding vertices in t_2 should also be connected to $v_{T''}$ - otherwise, vertices in t_1 cannot be mapped to those of t_2. However, this is a contradiction to the way G_M is constructed, since the vertices in t_2 are connected to two transition probability vertices. Therefore, there exists an automorphism ϕ such that $T(s, \vec{a}, s') = T(\phi_S(s), \phi_{\vec{A}}(\vec{a}), \phi_S(s'))$, where $\phi_G(v_s) = v_{\phi_S(s)}, \phi_G(v_{a_i}) = v_{\phi_{\vec{A}}(a_{\phi_I(i)})}, \forall i \in I, a_i \in A_i, s \in S$. The analogous equalities for the observation and reward functions can be proved similarly. Furthermore, similar to the proof of the reverse direction, ϕ is unique to the given ϕ_G because ϕ_G is a bijection. □

We also show that, given any vertex-colored undirected graph G, we can construct POSG so that an automorphism of G corresponds to a unique symmetry of the POSG, and vice versa. The constructed POSG consists of a single agent, action, and observation. Each state of the POSG corresponds to each vertex of G. In more detail, the construction is as follows: Prepare a POSG state per each $v \in V$. With a slight abuse of notation, we will use the notations for states and vertices interchangeably. We take the agent set to be a singleton set. There is only a single action, a, for this POSG, and the transition probabilities are determined as follows: Let $deg(v)$ denote the degree of vertex v. Then $T(v, a, u) = \frac{1}{deg(v)}, \forall (v, u) \in E$. A self-transition of probability 1 is implicitly assigned to zero-degree vertices. Therefore, the transition probability assignment will need $O(|V|D)$ time, where $D = \max_{v \in V} deg(v)$. This complexity is again

upper-bounded by $O(|V|^2)$, since there can be at most $|V|-1$ edges connected to any given vertex. There is only one observation z, leading to an identical observation probability function of 1 to all (s,a) pairs. That is, $O(v,a,z)=1, \forall v \in V$. This assignment is done in $O(|V|)$ time. For the reward component, we assign the reward according to the color of the vertex at which the action is taken. That is, $R(v,a)=N(\psi(v))$, where $N: C \rightarrow \Re$ is taken to be any bijection that maps colors to real numbers.

Definition 4. *Given a vertex-colored undirected graph $G=\langle V,E,C,\psi\rangle$, M_G denotes the POSG representation of G via construction steps described above.*

Lemma 2. *An automorphism of M_G corresponds to a unique symmetry of M, and vice versa.*

Proof. First, we show that there is a unique symmetry ϕ of M_G for an automorphism ϕ_G of G. Because the vertices constitute the state space of M_G, only the states are permuted. By the edge-preserving property of ϕ_G, $deg(v)=deg(\phi_G(v))$ for all vertices in G. It follows that $T(v,a,u)=T(\phi_G(v),a,\phi_G(u)), \forall (v,u) \in E$. By the color-preserving property of ϕ_G, $R(v,a)=\psi(v)=\psi(\phi_G(v))=R(\phi_G(v),a)$ holds. Lastly, the observation probability remains invariant to any automorphism since it is constant for all states. Therefore, we can construct ϕ by permuting the states the way they were permuted by ϕ_G. Notice that because $\phi_G(v) \neq \phi'_G(v), \forall \phi'_G \neq \phi_G$, ϕ is unique.

To prove the other direction, we assume the symmetry ϕ of M_G is given. By definition of ϕ, $T(s,a,s')=T(\phi_S(s),a,\phi_S(s'))$ and $R(s,a)=R(\phi_S(s),a)$ holds. The equivalence of the transition probabilities implies that $deg(v_s)=deg(v_{\phi(s)})$ for the vertex v_s corresponding to state s. This equality holds for all $v \in V$. To this end, we can set $\phi_G(v_s) \leftarrow v_{\phi(s)}$ as our unique ϕ_G. To see that this ϕ_G supports edge-preservation, take any $(v,u) \in E$. Let s_v and s_u be the states mapped to v and u, respectively. Then $T(s_v,a,s_u)=\frac{1}{deg(v)}=\frac{1}{deg(\phi_G(v))}=T(\phi(s_v),a,\phi(s_u))$. The fact that the last term is non-zero indicates that $(\phi_G(v),\phi_G(u)) \in E$ as well. Also, for any $(v,u) \notin E$, $T(s_v,a,s_u)=T(\phi_G(v),a,\phi_G(u))=0$, hence $(\phi_G(v),\phi_G(u)) \notin E$ as well. □

We now state the main theorem regarding the computational complexity of finding symmetries of POSGs. We denote the problem of finding the generators[2] of automorphism groups of a graph G by AGEN(G), and the problem of finding the symmetries of a given POSG M by PSYMM(M). It is known that AGEN(G) belongs to the class GI-complete [6]. We use this fact to prove that the computational complexity of PSYMM(M) is GI-complete as well.

To prove that PSYMM(M) is GI-complete, we need to show that PSYMM(M) $\leq_p$ AGEN(G_M) and AGEN(G) $\leq_p$ PSYMM(M_G), where $A \leq_p B$ denotes polynomial reducibility of problem A to problem B.

Theorem 5. *PSYMM(M) belongs to the class GI-complete.*

[2] Simply put, an *automorphism generator* of a graph is a set of permutations on the vertices such that when applied, yields permuted graphs.

Proof. We first show that PSYMM(M) $\leq_p$ AGEN(G_M). The number of vertices in G_M is $O(|I|^2|S|^2\prod_i|A_i||Z_i|)$, which is polynomial in the number of agents, states, individual actions, and observations. Since the complexity of constructing any undirected graph from n vertices is at most $O(n^2)$ (in the case of a complete graph), it takes polynomial time to convert the POSG to the corresponding vertex colored undirected graph. By Lemma 1, the symmetries of M and the automorphisms of G_M are equivalent.

The second part of the proof aims to show that AGEN(G) $\leq_p$ PSYMM(M_G). For the purpose of parallel argument, we assume that the given graph is vertex-colored, although the argument can be specialized to non-colored graphs. Given a vertex-colored undirected graph $G = \langle V, E, C, \psi\rangle$, we will construct the corresponding POSG M_G. Note that it also takes polynomial time to convert the graph G to the POSG M_G. By Lemma 2, the automorphisms of G and the symmetries of M_G are equivalent. □

By setting $|I| = 1$, a POSG becomes a POMDP and all of the arguments presented in the proof of Theorem 5 carries over without modification. Hence, we can state the same result for an arbitrary POMDP regarding its computational complexity.

Corollary 2. *The problem of finding the symmetries of a POMDP belongs to the class GI-complete.*

Although the class GI-complete belongs to NP, it is neither known to be P nor NP-complete. It is however known to be in the low hierarchy of class NP, and there are a number of implementations that can solve GI problems efficiently.

6. Exploiting Symmetries in the Solution Methods

In this section, we present algorithms for POMDPs and POSGs taking advantage of symmetries present in the model. We first show how we can extend PBVI using the characteristics of POMDP symmetries discussed in Section 4.2. We then present an extended version of MADP for POSGs using the properties of POSG symmetries discussed in Section 4.4.

6.1. Extending PBVI for Symmetry Exploitation in POMDPs

With the set of automorphisms Φ that represents the set of all symmetries present in the model, we can modify PBVI to take advantage of the symmetries in belief states and α-vectors: First, when we sample the set of belief states, one of the heuristics used by PBVI is to select the belief state with the farthest $\|\cdot\|_1$ distance from any belief state already in B. Since we readily know the values at symmetric images of any belief state, we modify the $\|\cdot\|_1$ distance computation to handle symmetries: $\|b - b'\|_1^\Phi = \min_{\phi,\phi'\in\Phi}\|\phi(b) - \phi'(b')\|_1$. We then select the belief state with the farthest $\|\cdot\|_1^\Phi$ distance. This also allows us to exclude *symmetrically identical* belief states. Second, since B will exclude symmetrically identical belief states, we should modify the backup operation to include symmetric images of α-vectors into Γ_t^a. Table 3 shows the pseudo-code for performing the symmetric backup operation.

We also added a small but important improvement for the symmetric backup of α-vectors: some of the belief states will have the same symmetric image, *i.e.*, $b = \phi(b)$.

Require: $\Gamma_t = \text{backup}(B, \Gamma_{t-1}, \Phi)$
  for each $a \in A, z \in Z, \alpha_i \in \Gamma_{t-1}$ **do**
    for each $s \in S$ **do**
      $\alpha_i^{a,z}(s) = \gamma \sum_{s' \in S} T(s, a, s') O(s', a, z) \alpha_i(s')$
    end for
    $\Gamma_t^{a,z} = \cup_i \alpha_i^{a,z}$
  end for
  $\Gamma_t = \{\}$
  for each $b \in B$ **do**
    for each $a \in A, s \in S$ **do**
      $\alpha_b^a(s) = R(s, a) + \sum_{z \in Z} \text{argmax}_{\alpha \in \Gamma_t^{a,z}} (\alpha \cdot b)$
    end for
    $a^* = \text{argmax}_a (\alpha_b^a \cdot b)$
    $\alpha_b = \alpha_b^{a^*}$
    if $\alpha_b \notin \Gamma_t$ **then**
      $\Gamma_t = \Gamma_t \cup \alpha_b$
      for each $\phi = \langle f, g, h \rangle \in \Phi$ **do**
        if $f(\alpha_b) \notin \Gamma_t$ **then**
          $\Gamma_t = \Gamma_t \cup f(\alpha_b)$
        end if
      end for
    end if
  end for

Table 3: The backup operation of PBVI taking into account Φ, the set of all symmetries.

For these belief states, it is often unnecessary to add $\phi(\alpha_b)$ into Γ_t, since $\phi(\alpha_b)$ is relevant to the belief state $\phi(b)$ but b and $\phi(b)$ are the same! We thus identified which automorphisms yield $b \neq \phi(b)$ for each belief state b, and included the symmetric images of α-vectors only for these automorphisms.

6.2. Extending MADP for Symmetry Exploitation in POSGs

We now show how to apply our approach to POSGs. Using the results in Section 4.4, we can expect certain leverages in performance when using MADP. In particular, we make use of the symmetries in the two stages of the method:

- Value computation stage: The first major speed bottleneck occurs during the value computation, where we evaluate all the joint policies generated from the exhaustive backup. However, Theorem 3 states that for any given joint policy, its value vector is merely a permutation of the value vector of its symmetric image. Thus, the value computation for such policies can be avoided – we can simply permute the symmetric value vector whenever we need it. Note that in the case of inter-agent symmetries, all the value vectors of an agent can be obtained by permuting value vectors of its symmetric agent. The total number of value vectors decreases by a factor of $|\Phi|$.

Require: Sets of t-step policies $\Pi_{i,t}$, corresponding value vectors $\mathcal{V}_{i,t}$ for each agent i, and set of symmetries Φ.
The first stage of dynamic programming backup
Perform exhaustive backups to get $\Pi_{i,t+1}$ for each i.
for all $\vec{\pi} \in \vec{\Pi}_{t+1}$ **do**
  if $\nexists \phi \in \Phi,\ V_{i,t+1}^{\phi(\vec{\pi})}$ has been computed **then**
    Compute $V_{i,t+1}^{\vec{\pi}}$ (Eqn 5) and add the value vector to $\mathcal{V}_{i,t+1}$.
  end if
end for
# The second stage of dynamic programming backup
while any agent i has a prunable policy **do**
  $\text{NOPRUNE}_k \leftarrow \{\}, \forall k \in I$.
  for all $\pi \in \Pi_{i,t+1}$ **do**
    if $\pi \notin \text{NOPRUNE}_i$ and π can be pruned (Eqn 6) **then**
      $\Pi_{i,t+1} \leftarrow \Pi_{i,t+1} \backslash \pi$.
      for $\forall \phi \in \Phi$ **do**
        $\Pi_{\phi_I(i),t+1} \leftarrow \Pi_{\phi_I(i),t+1} \backslash \phi(\pi)$.
      end for
    else if π cannot be pruned **then**
      $\text{NOPRUNE}_i \leftarrow \text{NOPRUNE}_i \cup \{\pi\}$.
      for $\forall \phi \in \Phi$ **do**
        $\text{NOPRUNE}_{\phi_I(i)} \leftarrow \text{NOPRUNE}_{\phi_I(i)} \cup \{\phi(\pi)\}$.
      end for
    end if
  end for
end while
return Sets of t+1-step policies $\Pi_{i,t+1}$ and corresponding value vectors $\mathcal{V}_{i,t+1}$ for each agent i

Table 4: Dynamic programming for POSG with symmetries. NOPRUNE_i for each agent i maintains the list of policy trees that are found not prunable by the symmetry.

- Pruning stage: An even greater slowdown is due to LP routines for pruning. The existence of symmetries allows us to reduce the number of LP invocations. First, when a local policy π of agent i is pruned, Corollary 1 states that the local policy $\phi(\pi)$ of agent $\phi_I(i)$ can be pruned for all ϕ. Second, when π is not to be pruned, then all $\phi(\pi)$'s are not to be pruned as well. Therefore, LP need not be performed on those local policies.

The procedure for the multi-agent dynamic programming operator that exploits symmetry is outlined in Table 4.

7. Experiments

In this section, we empirically show how symmetries in POMDPs and POSGs can help reduce burdens on computational resources required to compute solutions. The

experiments are conducted on a number of standard benchmark domains in POMDPs and POSGs.

7.1. POMDP Experiments

Before we demonstrate the performance gain of the PBVI algorithm by using the symmetric backup operator, we first report test results for the existence of automorphisms in standard POMDP benchmark domains. Most of the benchmark domains are already compact in the sense that the model minimization algorithm was not able to further reduce the size in most of the domains. For the tiger-grid domain [16], we were able to reduce the size and find symmetries. For the tiger domain [12], we were not able to reduce the size, but still find symmetries.

We further tested for automorphism existence on larger domains. In the spoken dialogue management domain by Williams et al. [35], the user is trying to buy a ticket to travel from one city to another city, and the machine has to request or confirm information from the user in order to issue the correct ticket. These dialog management problems are denoted as n*-city-ticketing*. In this domain, there are n cities, and a human user is trying to book a flight between two cities. The agent, as an automated response system, needs to take one of the following actions: *greet*, *ask-from/ask-to*, *conf-to-x/conf-from-x*, *submit-x-y*, where x and y are two of the n cities. The user's response is treated as an observation for the agent: *x*, *from-x*, *to-x*, *from-x-to-y*, *yes*, *no*, *null*, where x and y again refer to the cities. The observation function is dependent on how well the speech recognition model performs. The states are factored into three components:

- Whether the *from* has been specified,
- Whether the destination, *to*, has been specified,
- Whether the current turn is the first turn or not.

We instantiated the domain for $n = 2$ and $n = 3$ possible cities, and for two different rates of speech recognition errors p_{err}, where $p_{\text{err}} = 0$ assumes no speech recognition error and $p_{\text{err}} = 0.1$ assumes an error rate of 10%. Note that even in the case where $p_{\text{err}} = 0$, the domain is still a POMDP since the user may provide partial information about the request (*e.g.*, origin city only).

All of these problems could not be reduced in size, but still had symmetries. Regardless of the value of p_{err}, the graphs encoding the POMDP models were exactly the same. The small differences in the nauty execution times may be due to the differences in the orderings of the vertices of the graph. Figure 10 summarizes the result of automorphism finding experiments.

Next, we experimented with the PBVI algorithms on the above benchmark domains using the discovered automorphisms. First, we sampled a fixed number of symmetric belief states (*e.g.*, 300 for the tiger-grid) and ran the symmetric version of PBVI. We then checked the number of unique belief states if the symmetric belief states were to be expanded by the automorphisms. We set this number (*e.g.*, 590 for the tiger-grid) as the number of belief states to be used by the non-symmetric version of PBVI, and ran the algorithm in the same setting without automorphisms. Note that our implementation

Domain	$\|S\|$	Min $\|S\|$	$\|V\|$	nauty exec time	$\|\Phi\|$
Tiger	2	2	39	0.004 s	2
Tiger-grid	36	35	9814	0.061 s	4
2-city-ticketing ($p_{\text{err}} = 0$)	397	397	1624545	31.872 s	4
2-city-ticketing ($p_{\text{err}} = 0.1$)	397	397	1624545	23.873 s	4
3-city-ticketing ($p_{\text{err}} = 0$)	1945	1945	61123604	2585.770 s	12
3-city-ticketing ($p_{\text{err}} = 0.1$)	1945	1945	61123604	2601.543 s	12

Figure 10: Model minimization and graph automorphism results on benchmark domains. $|S|$ is the number of states in the original model, Min $|S|$ is the number of states in the minimized model, $|V|$ is the number of vertices in the graph encoding of the model, and $|\Phi|$ is the number of automorphisms found by nauty including the identity mapping.

Domain	Algorithm	$\|B\|$	$\|\Gamma\|$	Iter	Exec time	$V(b_0)$	ϵ
Tiger	PBVI	19	5	89	0.07 s	6.40	0.01
	Symm-PBVI	10	5	89	0.05 s	6.40	
Tiger-grid	PBVI	590	532	88	359.69 s	0.80	0.03
	Symm-PBVI	300	529	85	196.09 s	0.80	
2-city-ticketing ($p_{\text{err}} = 0$)	PBVI	51	5	167	157.80 s	8.74	0.02
	Symm-PBVI	17	5	168	57.60 s	8.74	
2-city-ticketing ($p_{\text{err}} = 0.1$)	PBVI	104	9	167	546.04 s	7.76	0.02
	Symm-PBVI	30	10	167	201.97 s	7.73	
3-city-ticketing ($p_{\text{err}} = 0$)	PBVI	261	37	91	43094.32 s	8.08	1.00
	Symm-PBVI	36	42	91	9395.06 s	8.08	
3-city-ticketing ($p_{\text{err}} = 0.1$)	PBVI	275	39	91	43286.92 s	6.95	1.00
	Symm-PBVI	30	133	91	16791.17 s	6.94	

Figure 11: Performance comparisons of the PBVI algorithm with automorphisms. Symm-PBVI is the PBVI algorithm exploiting the automorphisms, *i.e.*, symmetric belief collection and symmetric backup. $|B|$ is the number of belief states given to the algorithms, $|\Gamma|$ is the number of α-vectors comprising the policy, Iter is the number of iterations until convergence, $V(b_0)$ is the average return of the policy starting from initial belief b_0, and ϵ is the convergence criteria of each algorithm for running until $\max_{b \in B} |V^{(n)}(b) - V^{(n-1)}(b)| \leq \epsilon$. All $V(b_0)$'s are within the 95% confidence interval of the optimal.

of PBVI slightly differs from the original version in that the original PBVI *interleaves* the belief state exploration and the value iteration, rather than fixing the belief states in the onset of execution. We also gathered the belief states simply using breadth-first traversal instead of stochastic simulation. This was to analyze the efficiency of the symmetric backup isolated from the effects of symmetric belief state exploration. Figure 11 shows the results of the experiments. In summary, automorphisms help significantly improve the performance of PBVI in running time without sacrificing the quality of policy.

7.2. POSG Experiments

There are no well-known benchmark domains for general POSGs, but there is a wealth of benchmark domains for DEC-POMDPs. Hence, we report the results on our symmetry exploitation in MADP for DEC-POMDPs only: Dec-Tiger [19], Grid-Small [1], and Box-Pushing [27]. By focusing on DEC-POMDPs, we can also rule out

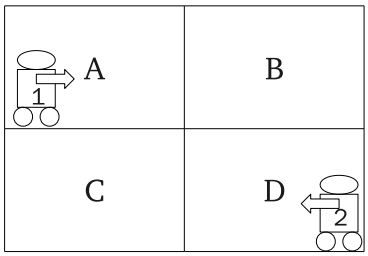

Figure 12: Grid-Small environment.

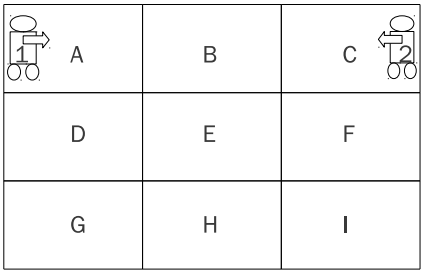

Figure 13: Grid-Small3x3 environment.

issues such as equilibrium selection problem in general-sum games.

The Dec-Tiger domain is a multi-agent extension of the well-known Tiger domain, which has been introduced in Section 4.3. The main difference is that the agents suffer less (gain more) by coordinating their actions - *e.g.*, the penalty is more severe for one agent unilaterally opening the door that leads to the tiger, than for both opening the door to the tiger.

The standard Grid-Small domain is set in a 2-by-2 grid world, where the two agents, $i = 1$ and $i = 2$, have to spend as much time as possible on the same grid cell. There are a total of 16 states (each grid cell either has an agent or not), five actions per agent ($a_{i,\text{UP}}$, $a_{i,\text{DOWN}}$, $a_{i,\text{RIGHT}}$, $a_{i,\text{LEFT}}$, $a_{i,\text{STAY}}$), and two observations per agent, denoted $z_{i,\text{LEFT}}$ and $z_{i,\text{RIGHT}}$, that indicate whether the agent senses a wall to its left or right, respectively. The 16 states are encoded as s_{XY}, where X and Y can take any one of $\{A, B, C, D\}$ given in Figure 12. The X indicates the cell in which agent 1 resides, and Y for agent 2, *e.g.*, s_{AD} is given in Figure 12. An extended version of Grid-Small is played in a 3-by-3 grid world. There are a total of 81 states, where the grid cells can take any one of $\{A, B, C, D, E, F, G, H, I\}$. The action set remains the same as the 2-by-2 case. There is an additional observation for not sensing a wall on either side, and is denoted as $z_{i,\text{NOTHING}}$ for agent i. A visual representation of the state s_{AC} is given in Figure 13.

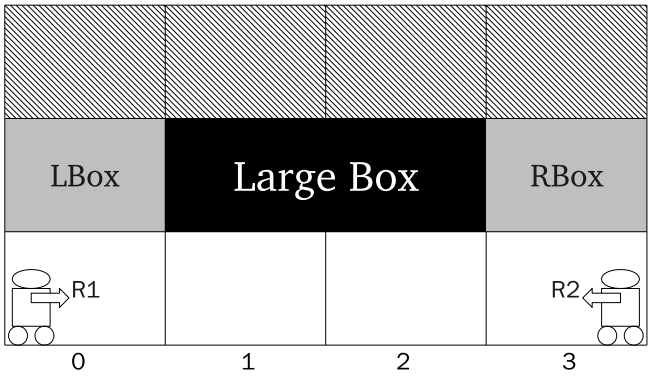

Figure 14: Initial configuration of the Box-Pushing domain. Immediately above the robots are the left and right small gray boxes, next to the black large box. The hatched region in the grid is the goal region.

The Box-Pushing domain requires the two agents, $i = 1$ or 2, to push two small boxes and one large box to a goal state. The large box is too heavy for a single agent to move, so the two must coordinate their actions in order to jointly push the large box. There are four actions per robot ($a_{i,\text{LEFT}}$, $a_{i,\text{RIGHT}}$, $a_{i,\text{MOVE}}$, $a_{i,\text{STAY}}$), five observations per robot ($z_{i,\text{SMALL}}$, $z_{i,\text{LARGE}}$, $z_{i,\text{WALL}}$, $z_{i,\text{EMPTY}}$, $z_{i,\text{OTHER}}$), and 100 states, four of which are goal states. The robots can either choose to place the two small boxes individually into the goal state and receive a small reward, or cooperatively push the large box and receive a greater reward. The initial state of the Box-Pushing domain is depicted in Figure 14. In this domain, two robots R1 and R2 start facing each other in a 3-by-4 grid. Notice that the location of R1 is always left to that of R2. This is because, in order for R1 to be left to R2, it must first move upwards. But since both robots have boxes above them, moving upward will cause the box to be positioned in the goal region, terminating the domain. This also accounts for the fact that the column coordinates, labeled 0 to 3, suffice to describe the positions of R1 and R2, since it is impossible for either robot to be in the above two rows of the grid without having the domain terminated. Thus, we adopt an alpha-numeric encoding to denote a particular non-goal state. All non-goal states will be of the form s_{XXYY}, where the first two Xs will be the column coordinates for R1 and R2 in that order, and the last two Ys take values from $\{r, l, u, d\}$ indicating the robot is facing right, left, up, or down, respectively. *e.g.*, s_{03rl} depicts the initial state given in Figure 14. The four goal states correspond to: the left small box being in the goal region (s_{LBox}), and the right small box being in the goal region (s_{RBox}), and both small boxes being in the goal region (s_{LRBox}), and the large box being in the goal region (s_{LargeBox}).

Prior to executing the symmetric MADP algorithm, we ran nauty on the graph encoding of each DEC-POMDP domain. The automorphisms in Dec-Tiger are presented in Figure 15. The automorphisms discovered included one trivial automorphism - the

Symm. Type	State	Action	Obs.
Inter-Agent	Identity mapping	$a_{1,\text{LISTEN}} \leftrightarrow a_{2,\text{LISTEN}}$ $a_{1,\text{LEFT}} \leftrightarrow a_{2,\text{LEFT}}$ $a_{1,\text{RIGHT}} \leftrightarrow a_{2,\text{RIGHT}}$	$z_{0,\text{LEFT}} \leftrightarrow z_{1,\text{LEFT}}$ $z_{1,\text{RIGHT}} \leftrightarrow z_{2,\text{RIGHT}}$
	$s_{\text{LEFT}} \leftrightarrow s_{\text{RIGHT}}$	$a_{1,\text{LISTEN}} \leftrightarrow a_{2,\text{LISTEN}}$ $a_{1,\text{LEFT}} \leftrightarrow a_{2,\text{RIGHT}}$ $a_{1,\text{RIGHT}} \leftrightarrow a_{2,\text{LEFT}}$	$z_{1,\text{LEFT}} \leftrightarrow z_{2,\text{RIGHT}}$ $z_{1,\text{RIGHT}} \leftrightarrow z_{2,\text{LEFT}}$
Intra-Agent	$s_{\text{LEFT}} \leftrightarrow s_{\text{RIGHT}}$	$a_{1,\text{LEFT}} \leftrightarrow a_{1,\text{RIGHT}}$ $a_{2,\text{LEFT}} \leftrightarrow a_{2,\text{RIGHT}}$ $a_{1,\text{LISTEN}} \leftrightarrow a_{1,\text{LISTEN}}$ $a_{2,\text{LISTEN}} \leftrightarrow a_{2,\text{LISTEN}}$	$z_{1,\text{LEFT}} \leftrightarrow z_{1,\text{RIGHT}}$ $z_{2,\text{LEFT}} \leftrightarrow z_{2,\text{RIGHT}}$

Figure 15: Non-trivial automorphisms in Dec-Tiger. The notation $X \leftrightarrow Y$ indicates that X is symmetric to Y.

identity mapping. There are three non-trivial automorphisms, two of them being inter-agent. The inter-agent and intra-agent automorphisms of Grid-Small domain are shown in Figure 17 and Figure 18, respectively in the Appendix. This domain contains eight automorphisms, including the identity mapping. Of the seven non-trivial automorphisms, four are inter-agent. Similarly for Grid-Small3x3, there are seven non-trivial automorphisms. These are shown in Figure 19 and Figure 21 in the Appendix as well. For the Box-Pushing domain, the only non-trivial automorphism is an inter-agent automorphism, as shown in Figure 23 in the Appendix. One notable symmetry of this domain is the interchange of the two states indicating the left and right small boxes being in the goal region (s_{LBox} and s_{Rbox}). In addition, the two agents and their corresponding actions and observations are swapped as well.

After computing the symmetries, we compared our proposed algorithm to the MADP algorithm on each domain. We measured the memory usage by counting the number of value vectors created at the end of each iteration. We also counted the number of LP invocations at each horizon. As can be seen in Figure 16, both algorithms were able to complete three and two horizons for the former two domains and the Box-pushing domain, respectively. For the Grid-Small3x3 domain, MADP could not complete horizon three, whereas Symm-MADP could. The running time for all symmetry-exploiting algorithms include the time taken to compute the symmetries using nauty, which explains why Symm-MADP takes slightly longer to complete the first time horizon in some cases. A separate field for nauty execution time is omitted, as it was negligible (less than 2.5 seconds) compared to the overall running time. Notice that even with the exploitation of symmetries, proceeding beyond the horizon attained by MADP is still spatially constrained. For the Dec-Tiger domain, value vectors alone take 70GB of memory by the end of value computation for horizon 4, even with full symmetry exploitation. Such a tendency is due to the fact that memory usage experiences exponential increase while symmetry only helps by a linear factor at best. However, this issue can be addressed by various approximate algorithms that bound the memory usage, and experimenting with their symmetric versions will be left as a future work.

Earlier horizons do not exhibit much of the benefit of the symmetries because very

Domain	Algorithm	$T=1$			$T=2$			$T=3$		
		#LP	Time	$\|\mathcal{V}\|$	#LP	Time	$\|\mathcal{V}\|$	#LP	Time	$\|\mathcal{V}\|$
Dec-Tiger	Symm-MADP	4	1s	4	42	1s	207	1022	326s	86175
	MADP	6	0s	9	84	1s	810	2371	1215s	344250
Grid-Small	Symm-MADP	2	1s	1	6	0s	11	124	24s	2631
	MADP	8	0s	1	10	1s	50	410	65s	20000
Grid-Small3x3	Symm-MADP	1	3s	1	5	1s	25	189	172800s	99809
	MADP	4	1s	1	5	1s	50		N.A.	
Box-Pushing	Symm-MADP	10	2s	8	156	1271s	768		N.A.	
	MADP	12	1s	16	290	3505s	1536			

Figure 16: Performance comparisons on domains with and without symmetry exploitation. #LP is the number of LP invocations and $\mathcal{V}$ is the set of value vectors produced at the end of each iteration. The first row of each domain shows the results with symmetry exploitation and the second row shows the results without symmetry. All time records are rounded up to the nearest second.

few policy trees are generated. However, towards the last horizon, we can see the effect of symmetry exploitation. While the number of value vectors reduced is approximately proportional to that of the symmetries present in the domain, the number of LP invocations and the execution time do not necessarily follow this trend. This is due to (1) the existence of many self-symmetric policy trees that do not contribute to multiple removal and LP avoidance, and (2) differing LP sizes, by which the LP solver's execution time varies.

The size of LP is an important factor that influences the execution time. The size is governed by how many policy trees were created from exhaustive backup and the domain size itself. For example, the Box-Pushing domain utilizes relatively larger LPs up to 12800 constraints, thereby amplifying the effect of symmetries. Since LP solvers usually take a high-order polynomial amount of time, reducing a linear number of variables or constraints in LPs will attain super-linear improvement in time.

8. Conclusion

We have presented a graph-theoretical framework for computing and exploiting symmetries for POMDPs and POSGs. In addition, we have shown in the experiments that the actual running time and space are significantly reduced by exploiting symmetries.

The computation of the symmetries were done by first encoding the problems into appropriate graph structures. The automorphisms of such graphs are then mapped back to the problem domain to represent the symmetries of the problem. In doing so, we have also provided a theoretical result that relates the computational complexity of symmetry computation to that of graph isomorphism computation, *i.e.*, the class GI-complete. Additionally, we have extended the concept of symmetry to a multi-agent setting, introducing POSG symmetries. Because of its multi-agent nature, symmetries in POSGs yield various implications in the area of game theory. We presented some game-theoretic properties that are exhibited in the presence of symmetries.

Our algorithms that exploit the symmetries are presented as well. These algorithms are modifications of previous well-known algorithms PBVI and MADP for POMDPs

and POSGs, respectively. Although we have demonstrated the efficiency of symmetry exploitation only using PBVI and MADP, the idea can be readily extended to other algorithms. For example, symmetries can have an impact on solution techniques that use heuristic search such as MAA* [33], or Q-value functions for DEC-POMDPs [22]. Another interesting area of application would be to apply symmetries to a finite controller representation of policies [1].

While symmetry exploitation greatly reduces computational and spatial burden on solving POMDPs and POSGs, it is limited by the fact that not all problems come with symmetries. One promising direction of research would be to compute approximate symmetries, along with the theoretical error bound.

Acknowledgment

We are indebted to the anonymous reviewers for their helpful comments in improving this article. This work was supported by Korea Research Foundation Grant KRF-D00527, and by Defense Acquisition Program Administration and Agency for Defense Development of Korea under contract UD080042AD.

References

[1] Amato, C., Zilberstein, S., 2008. Heuristic policy iteration for infinite-horizon decentralized POMDPs. In: Proceedings of the AAMAS 2008 Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains. pp. 1–15.

[2] Bernstein, D. S., Amato, C., Hansen, E. A., Zilberstein, S., 2009. Policy iteration for decentralized control of Markov decision processes. Journal of Artificial Intelligence Research 34, 89–132.

[3] Bernstein, D. S., Givan, R., Immerman, N., Zilberstein, S., 2002. The complexity of decentralized control of Markov decision processes. Mathematics of Operations Research 27 (4), 819–840.

[4] Bernstein, D. S., Hansen, E. A., Zilberstein, S., 2005. Bounded policy iteration for decentralized POMDPs. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence. pp. 1287–1292.

[5] Bödi, R., Herr, K., Joswig, M., 2010. Algorithms for highly symmetric linear and integer programs. Tech. Rep. arXiv:1012.4941.

[6] Booth, K. S., Colbourn, C. J., 1979. Problems polynomially equivalent to graph isomorphism. Tech. Rep. CS-77-04, University of Waterloo.

[7] Cheng, S.-F., Reeves, D. M., Vorobeychik, Y., Wellman, M. P., 2004. Notes on equilibria in symmetric games. In: Proceedings of AAMAS 2004 Workshop on Game-Theoretic and Decision Theoretic Agents. pp. 23–28.

[8] Dean, T., Givan, R., 1997. Model minimization in Markov decision processes. In: Proceedings of the 14th National Conference on Artificial Intelligence. pp. 106–111.

[9] Doshi, F., Roy, N., 2008. The permutable POMDP: fast solutions to POMDPs for preference elicitation. In: Proceedings of the 7th International Conference on Autonomous Agents and Multi-Agent Systems. pp. 493–500.

[10] Givan, R., Dean, T., Greig, M., 2003. Equivalence notions and model minimization in Markov decision processes. Artificial Intelligence 147 (1–2), 163–223.

[11] Hansen, E. A., Bernstein, D. S., Zilberstein, S., 2004. Dynamic programming for partially observable stochastic games. In: Proceedings of the 19th National Conference on Artificial Intelligence. pp. 709–715.

[12] Kaelbling, L. P., Littman, M. L., Cassandra, A. R., 1998. Planning and acting in partially observable stochastic domains. Artificial Intelligence 101 (1–2), 99–134.

[13] Kim, K.-E., 2008. Exploiting symmetries in POMDPs for point-based algorithms. In: Proceedings of the 23rd AAAI Conference on Artificial Intelligence. pp. 1043–1048.

[14] Kim, Y., Kim, K.-E., 2010. Point-based policy iteration for decentralized POMDPs. In: Proceedings of the 11th Pacific Rim International Conference on Artificial Intelligence. pp. 614–619.

[15] Kumar, A., Zilberstein, S., 2010. Point-based backup for decentralized POMDPs: Complexity and new algorithms. In: Proceedings of the 9th International Conference on Autonomous Agents and Multi-Agent Systems. pp. 1315–1322.

[16] Littman, M. L., Cassandra, A. R., Kaelbling, L. P., 1995. Learning policies for partially observable environments: Scaling up. In: Proceedings of the 12th International Conference on Machine Learning. pp. 362–370.

[17] Madani, O., Hanks, S., Condon, A., 2003. On the undecidability of probabilistic planning and related stochastic optimization problems. Artificial Intelligence 147 (1-2), 5–34.

[18] McKay, B. D., 2007. Nauty user's guide, version 2.4 (online document).
URL `http://cs.anu.edu.au/~bdm/nauty/nug.pdf`

[19] Nair, R., Tambe, M., Yokoo, M., Pynadath, D., Marsella, S., 2003. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence. pp. 705–711.

[20] Narayanamurthy, S. M., Ravindran, B., 2008. On the hardness of finding symmetries in Markov decision processes. In: Proceedings of the 25th International Conference on Machine Learning. pp. 688–695.

[21] Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V. V., 2007. Algorithmic Game Theory. Cambridge University Press, New York, NY, USA.

[22] Oliehoek, F. A., Vlassis, N., 2007. Q-value functions for decentralized POMDPs. In: Proceedings of the 6th International Conference on Autonomous Agents and Multi-Agent Systems. pp. 838–845.

[23] Papadimitriou, C. H., Tsitsiklis, J. N., 1987. The complexity of Markov decision processes. Mathematics of Operations Research 12 (3), 441–450.

[24] Pineau, J., Gordon, G., Thrun, S., 2003. Policy-contingent abstraction for robust robot control. In: Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence. pp. 477–484.

[25] Pineau, J., Gordon, G., Thrun, S., 2006. Anytime point-based approximation for large POMDPs. Journal of Artificial Intelligence Research 27, 335–380.

[26] Ravindran, B., Barto, A. G., 2001. Symmetries and model minimization in Markov decision processes. Tech. Rep. CMPSCI 01-43, University of Massachusetts, Amherst.

[27] Seuken, S., Zilberstein, S., 2007. Improved memory-bounded dynamic programming for decentralized POMDPs. In: Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence. pp. 344–351.

[28] Seuken, S., Zilberstein, S., 2007. Memory-bounded dynamic programming for DEC-POMDPs. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence. pp. 2009–2015.

[29] Shani, G., Brafman, R. I., Shimony, S. E., 2007. Forward search value iteration for POMDPs. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence. pp. 2617–2624.

[30] Smith, T., Simmons, R., 2004. Heuristic search value iteration for POMDPs. In: Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence. pp. 520–527.

[31] Sondik, E. J., 1971. The optimal control of partially observable Markov decision processes. Ph.D. thesis, Department of Electrical Engineering, Stanford University.

[32] Spaan, M. T. J., Vlassis, N., 2005. Perseus: Randomized point-based value iteration for POMDPs. Journal of Artificial Intelligence Research 24, 195–220.

[33] Szer, D., Charpillet, F., Zilberstein, S., 2005. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In: Proceedings of 21st Conference on Uncertainty in Artificial Intelligence. pp. 576–583.

[34] White, C., 1991. A survey of solution techniques for the partially observed Markov decision process. Annals of Operations Research 32 (1–4), 215–230.

[35] Williams, J. D., Poupart, P., Young, S., 2005. Factored partially observable Markov decision processes for dialogue management. In: Proceedings of the IJCAI 2005 Workshop on Knowledge and Reasoning in Practical Dialogue Systems. pp. 76–82.

[36] Wolfe, A. P., 2006. POMDP homomorphisms. In: Proceedings of the NIPS 2006 Workshop on Grounding Perception, Knowledge and Cognition in Sensory-Motor Experience.

[37] Wu, F., Zilberstein, S., Chen, X., 2010. Point-based policy generation for decentralized POMDPs. In: Proceedings of the 9th International Conference on Autonomous Agents and Multi-Agent Systems. pp. 1307–1314.

Appendix

The figures in this appendix show the automorphisms in Grid-Small, Grid-Small3x3, and Box-Pushing, accompanying the results in the experiments section.

Symm. Type	State	Action	Obs.
Inter-Agent	$s_{AB} \leftrightarrow s_{BA}$ $s_{AC} \leftrightarrow s_{CA}$ $s_{AD} \leftrightarrow s_{DA}$ $s_{BC} \leftrightarrow s_{CB}$ $s_{BD} \leftrightarrow s_{DB}$ $s_{CD} \leftrightarrow s_{DC}$	$a_{1,\text{UP}} \leftrightarrow a_{2,\text{UP}}$ $a_{1,\text{DOWN}} \leftrightarrow a_{2,\text{DOWN}}$ $a_{1,\text{LEFT}} \leftrightarrow a_{2,\text{LEFT}}$ $a_{1,\text{RIGHT}} \leftrightarrow a_{2,\text{RIGHT}}$ $a_{1,\text{STAY}} \leftrightarrow a_{2,\text{STAY}}$	$z_{1,\text{LEFT}} \leftrightarrow z_{2,\text{LEFT}}$ $z_{1,\text{RIGHT}} \leftrightarrow z_{2,\text{RIGHT}}$
	$s_{AA} \leftrightarrow s_{CC}$ $s_{AB} \leftrightarrow s_{DC}$ $s_{AD} \leftrightarrow s_{BC}$ $s_{BA} \leftrightarrow s_{CD}$ $s_{BB} \leftrightarrow s_{DD}$ $s_{CB} \leftrightarrow s_{DA}$	$a_{1,\text{UP}} \leftrightarrow a_{2,\text{DOWN}}$ $a_{1,\text{DOWN}} \leftrightarrow a_{2,\text{UP}}$ $a_{1,\text{LEFT}} \leftrightarrow a_{2,\text{LEFT}}$ $a_{1,\text{RIGHT}} \leftrightarrow a_{2,\text{RIGHT}}$ $a_{1,\text{STAY}} \leftrightarrow a_{2,\text{STAY}}$	$z_{1,\text{LEFT}} \leftrightarrow z_{2,\text{LEFT}}$ $z_{1,\text{RIGHT}} \leftrightarrow z_{2,\text{RIGHT}}$
	$s_{AA} \leftrightarrow s_{BB}$ $s_{AC} \leftrightarrow s_{DB}$ $s_{AD} \leftrightarrow s_{CB}$ $s_{BC} \leftrightarrow s_{DA}$ $s_{BD} \leftrightarrow s_{CA}$ $s_{CC} \leftrightarrow s_{DD}$	$a_{1,\text{UP}} \leftrightarrow a_{2,\text{UP}}$ $a_{1,\text{DOWN}} \leftrightarrow a_{2,\text{DOWN}}$ $a_{1,\text{LEFT}} \leftrightarrow a_{2,\text{RIGHT}}$ $a_{1,\text{RIGHT}} \leftrightarrow a_{2,\text{LEFT}}$ $a_{1,\text{STAY}} \leftrightarrow a_{2,\text{STAY}}$	$z_{1,\text{LEFT}} \leftrightarrow z_{2,\text{RIGHT}}$ $z_{1,\text{RIGHT}} \leftrightarrow z_{2,\text{LEFT}}$
	$s_{AA} \leftrightarrow s_{DD}$ $s_{AB} \leftrightarrow s_{CD}$ $s_{AC} \leftrightarrow s_{BD}$ $s_{BA} \leftrightarrow s_{DC}$ $s_{BB} \leftrightarrow s_{CC}$ $s_{CA} \leftrightarrow s_{DB}$	$a_{1,\text{UP}} \leftrightarrow a_{2,\text{DOWN}}$ $a_{1,\text{DOWN}} \leftrightarrow a_{2,\text{UP}}$ $a_{1,\text{LEFT}} \leftrightarrow a_{2,\text{RIGHT}}$ $a_{1,\text{RIGHT}} \leftrightarrow a_{2,\text{LEFT}}$ $a_{1,\text{STAY}} \leftrightarrow a_{2,\text{STAY}}$	$z_{1,\text{LEFT}} \leftrightarrow z_{2,\text{RIGHT}}$ $z_{1,\text{RIGHT}} \leftrightarrow z_{2,\text{LEFT}}$

Figure 17: Non-trivial inter-agent automorphisms in Grid-Small.

Symm. Type	State	Action	Obs.
Intra-Agent	$s_{AA} \leftrightarrow s_{CC}$ $s_{AB} \leftrightarrow s_{CD}$ $s_{AC} \leftrightarrow s_{CA}$ $s_{AD} \leftrightarrow s_{CB}$ $s_{BA} \leftrightarrow s_{DC}$ $s_{BB} \leftrightarrow s_{DD}$ $s_{BC} \leftrightarrow s_{DA}$ $s_{BD} \leftrightarrow s_{DB}$	$a_{1,\text{UP}} \leftrightarrow a_{1,\text{DOWN}}$ $a_{2,\text{UP}} \leftrightarrow a_{2,\text{DOWN}}$	Identity mapping
	$s_{AA} \leftrightarrow s_{BB}$ $s_{AB} \leftrightarrow s_{BA}$ $s_{AC} \leftrightarrow s_{BD}$ $s_{AD} \leftrightarrow s_{BC}$ $s_{CA} \leftrightarrow s_{DB}$ $s_{CB} \leftrightarrow s_{DA}$ $s_{CC} \leftrightarrow s_{DD}$ $s_{CD} \leftrightarrow s_{DC}$	$a_{1,\text{LEFT}} \leftrightarrow a_{1,\text{RIGHT}}$ $a_{2,\text{LEFT}} \leftrightarrow a_{2,\text{RIGHT}}$	$z_{1,\text{LEFT}} \leftrightarrow z_{1,\text{RIGHT}}$ $z_{2,\text{LEFT}} \leftrightarrow z_{2,\text{RIGHT}}$
	$s_{AA} \leftrightarrow s_{DD}$ $s_{AB} \leftrightarrow s_{DC}$ $s_{AC} \leftrightarrow s_{DB}$ $s_{AD} \leftrightarrow s_{DA}$ $s_{BA} \leftrightarrow s_{CD}$ $s_{BB} \leftrightarrow s_{CC}$ $s_{BC} \leftrightarrow s_{CB}$ $s_{BD} \leftrightarrow s_{CA}$	$a_{1,\text{UP}} \leftrightarrow a_{1,\text{DOWN}}$ $a_{2,\text{UP}} \leftrightarrow a_{2,\text{DOWN}}$ $a_{1,\text{LEFT}} \leftrightarrow a_{1,\text{RIGHT}}$ $a_{2,\text{LEFT}} \leftrightarrow a_{2,\text{RIGHT}}$	$z_{1,\text{LEFT}} \leftrightarrow z_{1,\text{RIGHT}}$ $z_{2,\text{LEFT}} \leftrightarrow z_{2,\text{RIGHT}}$

Figure 18: Non-trivial intra-agent automorphisms in Grid-Small.

Symm. Type	State	Action	Obs.
Inter-Agent	$s_{AB} \leftrightarrow s_{BA}, s_{AC} \leftrightarrow s_{CA}$ $s_{AD} \leftrightarrow s_{DA}, s_{AE} \leftrightarrow s_{EA}$ $s_{AF} \leftrightarrow s_{FA}, s_{AG} \leftrightarrow s_{GA}$ $s_{AH} \leftrightarrow s_{HA}, s_{AI} \leftrightarrow s_{IA}$ $s_{BC} \leftrightarrow s_{CB}, s_{BD} \leftrightarrow s_{DB}$ $s_{BE} \leftrightarrow s_{EB}, s_{BF} \leftrightarrow s_{FB}$ $s_{BG} \leftrightarrow s_{GB}, s_{BH} \leftrightarrow s_{HB}$ $s_{BI} \leftrightarrow s_{IB}, s_{CD} \leftrightarrow s_{DC}$ $s_{CE} \leftrightarrow s_{EC}, s_{CF} \leftrightarrow s_{FC}$ $s_{CG} \leftrightarrow s_{GC}, s_{CH} \leftrightarrow s_{HC}$ $s_{CI} \leftrightarrow s_{IC}, s_{DE} \leftrightarrow s_{ED}$ $s_{DF} \leftrightarrow s_{FD}, s_{DG} \leftrightarrow s_{GD}$ $s_{DH} \leftrightarrow s_{HD}, s_{DI} \leftrightarrow s_{ID}$ $s_{EF} \leftrightarrow s_{FE}, s_{EG} \leftrightarrow s_{GE}$ $s_{EH} \leftrightarrow s_{HE}, s_{EI} \leftrightarrow s_{IE}$ $s_{FG} \leftrightarrow s_{GF}, s_{FH} \leftrightarrow s_{HF}$ $s_{FI} \leftrightarrow s_{IF}, s_{GH} \leftrightarrow s_{HG}$ $s_{GI} \leftrightarrow s_{IG}, s_{HI} \leftrightarrow s_{IH}$	$a_{1,\text{UP}} \leftrightarrow a_{2,\text{UP}}$ $a_{1,\text{DOWN}} \leftrightarrow a_{2,\text{DOWN}}$ $a_{1,\text{LEFT}} \leftrightarrow a_{2,\text{LEFT}}$ $a_{1,\text{RIGHT}} \leftrightarrow a_{2,\text{RIGHT}}$ $a_{1,\text{STAY}} \leftrightarrow a_{2,\text{STAY}}$	$z_{1,\text{LEFT}} \leftrightarrow z_{2,\text{LEFT}}$ $z_{1,\text{RIGHT}} \leftrightarrow z_{2,\text{RIGHT}}$ $z_{1,\text{NOTHING}} \leftrightarrow z_{2,\text{NOTHING}}$
	$s_{AA} \leftrightarrow s_{CC}, s_{AB} \leftrightarrow s_{BC}$ $s_{AD} \leftrightarrow s_{FC}, s_{AE} \leftrightarrow s_{EC}$ $s_{AF} \leftrightarrow s_{DC}, s_{AG} \leftrightarrow s_{IC}$ $s_{AH} \leftrightarrow s_{HC}, s_{AI} \leftrightarrow s_{GC}$ $s_{BA} \leftrightarrow s_{CB}, s_{BD} \leftrightarrow s_{FB}$ $s_{BE} \leftrightarrow s_{EB}, s_{BF} \leftrightarrow s_{DB}$ $s_{BG} \leftrightarrow s_{IB}, s_{BH} \leftrightarrow s_{HB}$ $s_{BI} \leftrightarrow s_{GB}, s_{CD} \leftrightarrow s_{FA}$ $s_{CE} \leftrightarrow s_{EA}, s_{CF} \leftrightarrow s_{DA}$ $s_{CG} \leftrightarrow s_{IA}, s_{CH} \leftrightarrow s_{HA}$ $s_{CI} \leftrightarrow s_{GA}, s_{DD} \leftrightarrow s_{FF}$ $s_{DE} \leftrightarrow s_{EF}, s_{DG} \leftrightarrow s_{IF}$ $s_{DH} \leftrightarrow s_{HF}, s_{DI} \leftrightarrow s_{GF}$ $s_{ED} \leftrightarrow s_{FE}, s_{EG} \leftrightarrow s_{IE}$ $s_{EH} \leftrightarrow s_{HE}, s_{EI} \leftrightarrow s_{GE}$ $s_{FG} \leftrightarrow s_{ID}, s_{FH} \leftrightarrow s_{HD}$ $s_{FI} \leftrightarrow s_{GD}, s_{GG} \leftrightarrow s_{II}$ $s_{GH} \leftrightarrow s_{HI}, s_{HG} \leftrightarrow s_{IH}$	$a_{1,\text{UP}} \leftrightarrow a_{2,\text{UP}}$ $a_{1,\text{DOWN}} \leftrightarrow a_{2,\text{DOWN}}$ $a_{1,\text{LEFT}} \leftrightarrow a_{2,\text{LEFT}}$ $a_{1,\text{RIGHT}} \leftrightarrow a_{2,\text{RIGHT}}$ $a_{1,\text{STAY}} \leftrightarrow a_{2,\text{STAY}}$	$z_{1,\text{LEFT}} \leftrightarrow z_{2,\text{RIGHT}}$ $z_{1,\text{RIGHT}} \leftrightarrow z_{2,\text{RIGHT}}$ $z_{1,\text{NOTHING}} \leftrightarrow z_{2,\text{NOTHING}}$

Figure 19: Non-trivial inter-agent automorphism in Grid-Small3x3.

Symm. Type	State	Action	Obs.
Inter (contd.)	$s_{AA} \leftrightarrow s_{GG}, s_{AB} \leftrightarrow s_{HG}$ $s_{AC} \leftrightarrow s_{IG}, s_{AD} \leftrightarrow s_{DG}$ $s_{AE} \leftrightarrow s_{EG}, s_{AF} \leftrightarrow s_{FG}$ $s_{AH} \leftrightarrow s_{BG}, s_{AI} \leftrightarrow s_{CG}$ $s_{BA} \leftrightarrow s_{GH}, s_{BB} \leftrightarrow s_{HH}$ $s_{BC} \leftrightarrow s_{IH}, s_{BD} \leftrightarrow s_{DH}$ $s_{BE} \leftrightarrow s_{EH}, s_{BF} \leftrightarrow s_{FH}$ $s_{BI} \leftrightarrow s_{CH}, s_{CA} \leftrightarrow s_{GI}$ $s_{CB} \leftrightarrow s_{HI}, s_{CC} \leftrightarrow s_{II}$ $s_{CD} \leftrightarrow s_{DI}, s_{CE} \leftrightarrow s_{EI}$ $s_{CF} \leftrightarrow s_{FI}, s_{DA} \leftrightarrow s_{GD}$ $s_{DB} \leftrightarrow s_{HD}, s_{DC} \leftrightarrow s_{ID}$ $s_{DE} \leftrightarrow s_{ED}, s_{DF} \leftrightarrow s_{FD}$ $s_{EA} \leftrightarrow s_{GE}, s_{EB} \leftrightarrow s_{HE}$ $s_{EC} \leftrightarrow s_{IE}, s_{EF} \leftrightarrow s_{FE}$ $s_{FA} \leftrightarrow s_{GF}, s_{FB} \leftrightarrow s_{HF}$ $s_{FC} \leftrightarrow s_{IF}, s_{GB} \leftrightarrow s_{HA}$ $s_{GC} \leftrightarrow s_{IA}, s_{HC} \leftrightarrow s_{IB}$	$a_{1,\text{UP}} \leftrightarrow a_{2,\text{DOWN}}$ $a_{1,\text{DOWN}} \leftrightarrow a_{2,\text{UP}}$ $a_{1,\text{LEFT}} \leftrightarrow a_{2,\text{LEFT}}$ $a_{1,\text{RIGHT}} \leftrightarrow a_{2,\text{RIGHT}}$ $a_{1,\text{STAY}} \leftrightarrow a_{2,\text{STAY}}$	$z_{1,\text{LEFT}} \leftrightarrow z_{2,\text{LEFT}}$ $z_{1,\text{RIGHT}} \leftrightarrow z_{2,\text{RIGHT}}$ $z_{1,\text{NOTHING}} \leftrightarrow z_{2,\text{NOTHING}}$
	$s_{AA} \leftrightarrow s_{II}, s_{AB} \leftrightarrow s_{HI}$ $s_{AC} \leftrightarrow s_{GI}, s_{AD} \leftrightarrow s_{FI}$ $s_{AE} \leftrightarrow s_{EI}, s_{AF} \leftrightarrow s_{DI}$ $s_{AG} \leftrightarrow s_{CI}, s_{AH} \leftrightarrow s_{BI}$ $s_{BA} \leftrightarrow s_{IH}, s_{BB} \leftrightarrow s_{HH}$ $s_{BC} \leftrightarrow s_{GH}, s_{BD} \leftrightarrow s_{FH}$ $s_{BE} \leftrightarrow s_{EH}, s_{BF} \leftrightarrow s_{DH}$ $s_{BG} \leftrightarrow s_{CH}, s_{CA} \leftrightarrow s_{IG}$ $s_{CB} \leftrightarrow s_{HG}, s_{CC} \leftrightarrow s_{GG}$ $s_{CD} \leftrightarrow s_{FG}, s_{CE} \leftrightarrow s_{EG}$ $s_{CF} \leftrightarrow s_{DG}, s_{DA} \leftrightarrow s_{IF}$ $s_{DB} \leftrightarrow s_{HF}, s_{DC} \leftrightarrow s_{GF}$ $s_{DD} \leftrightarrow s_{FF}, s_{DE} \leftrightarrow s_{EF}$ $s_{EA} \leftrightarrow s_{IE}, s_{EB} \leftrightarrow s_{HE}$ $s_{EC} \leftrightarrow s_{GE}, s_{ED} \leftrightarrow s_{FE}$ $s_{FA} \leftrightarrow s_{ID}, s_{FB} \leftrightarrow s_{HD}$ $s_{FC} \leftrightarrow s_{GD}, s_{GA} \leftrightarrow s_{IC}$ $s_{GB} \leftrightarrow s_{HC}, s_{HA} \leftrightarrow s_{IB}$	$a_{1,\text{UP}} \leftrightarrow a_{2,\text{DOWN}}$ $a_{1,\text{DOWN}} \leftrightarrow a_{2,\text{UP}}$ $a_{1,\text{LEFT}} \leftrightarrow a_{2,\text{RIGHT}}$ $a_{1,\text{RIGHT}} \leftrightarrow a_{2,\text{LEFT}}$ $a_{1,\text{STAY}} \leftrightarrow a_{2,\text{STAY}}$	$z_{1,\text{LEFT}} \leftrightarrow z_{2,\text{RIGHT}}$ $z_{1,\text{RIGHT}} \leftrightarrow z_{2,\text{LEFT}}$ $z_{1,\text{NOTHING}} \leftrightarrow z_{2,\text{NOTHING}}$

Figure 20: Non-trivial inter-agent automorphism in Grid-Small3x3, continued.

Symm. Type	State	Action	Obs.
Intra-Agent	$s_{AA} \leftrightarrow s_{CC}, s_{AB} \leftrightarrow s_{CB}$ $s_{AC} \leftrightarrow s_{CA}, s_{AD} \leftrightarrow s_{CF}$ $s_{AE} \leftrightarrow s_{CE}, s_{AF} \leftrightarrow s_{CD}$ $s_{AG} \leftrightarrow s_{CI}, s_{AH} \leftrightarrow s_{CH}$ $s_{AI} \leftrightarrow s_{CG}, s_{BA} \leftrightarrow s_{BC}$ $s_{BD} \leftrightarrow s_{BF}, s_{BG} \leftrightarrow s_{BI}$ $s_{DA} \leftrightarrow s_{FC}, s_{DB} \leftrightarrow s_{FB}$ $s_{DC} \leftrightarrow s_{FA}, s_{DD} \leftrightarrow s_{FF}$ $s_{DE} \leftrightarrow s_{FE}, s_{DF} \leftrightarrow s_{FD}$ $s_{DG} \leftrightarrow s_{FI}, s_{DH} \leftrightarrow s_{FH}$ $s_{DI} \leftrightarrow s_{FG}, s_{EA} \leftrightarrow s_{EC}$ $s_{ED} \leftrightarrow s_{EF}, s_{EG} \leftrightarrow s_{EI}$ $s_{GA} \leftrightarrow s_{IC}, s_{GB} \leftrightarrow s_{IB}$ $s_{GC} \leftrightarrow s_{IA}, s_{GD} \leftrightarrow s_{IF}$ $s_{GE} \leftrightarrow s_{IE}, s_{GF} \leftrightarrow s_{ID}$ $s_{GG} \leftrightarrow s_{II}, s_{GH} \leftrightarrow s_{IH}$ $s_{GI} \leftrightarrow s_{IG}, s_{HA} \leftrightarrow s_{HC}$ $s_{HD} \leftrightarrow s_{HF}, s_{HG} \leftrightarrow s_{HI}$	$a_{1,\text{LEFT}} \leftrightarrow a_{1,\text{RIGHT}}$ $a_{2,\text{LEFT}} \leftrightarrow a_{2,\text{RIGHT}}$	$z_{1,\text{LEFT}} \leftrightarrow z_{1,\text{RIGHT}}$ $z_{2,\text{LEFT}} \leftrightarrow z_{2,\text{RIGHT}}$
	$s_{AA} \leftrightarrow s_{GG}, s_{AB} \leftrightarrow s_{GH}$ $s_{AC} \leftrightarrow s_{GI}, s_{AD} \leftrightarrow s_{GD}$ $s_{AE} \leftrightarrow s_{GE}, s_{AF} \leftrightarrow s_{GF}$ $s_{AG} \leftrightarrow s_{GA}, s_{AH} \leftrightarrow s_{GB}$ $s_{AI} \leftrightarrow s_{GC}, s_{BA} \leftrightarrow s_{HG}$ $s_{BB} \leftrightarrow s_{HH}, s_{BC} \leftrightarrow s_{HI}$ $s_{BD} \leftrightarrow s_{HD}, s_{BE} \leftrightarrow s_{HE}$ $s_{BF} \leftrightarrow s_{HF}, s_{BG} \leftrightarrow s_{HA}$ $s_{BH} \leftrightarrow s_{HB}, s_{BI} \leftrightarrow s_{HC}$ $s_{CA} \leftrightarrow s_{IG}, s_{CB} \leftrightarrow s_{IH}$ $s_{CC} \leftrightarrow s_{II}, s_{CD} \leftrightarrow s_{ID}$ $s_{CE} \leftrightarrow s_{IE}, s_{CF} \leftrightarrow s_{IF}$ $s_{CG} \leftrightarrow s_{IA}, s_{CH} \leftrightarrow s_{IB}$ $s_{CI} \leftrightarrow s_{IC}, s_{DA} \leftrightarrow s_{DG}$ $s_{DB} \leftrightarrow s_{DH}, s_{DC} \leftrightarrow s_{DI}$ $s_{EA} \leftrightarrow s_{EG}, s_{EB} \leftrightarrow s_{EH}$ $s_{EC} \leftrightarrow s_{EI}, s_{FA} \leftrightarrow s_{FG}$ $s_{FB} \leftrightarrow s_{FH}, s_{FC} \leftrightarrow s_{FI}$	$a_{1,\text{UP}} \leftrightarrow a_{1,\text{DOWN}}$ $a_{2,\text{UP}} \leftrightarrow a_{2,\text{DOWN}}$	Identity Mapping

Figure 21: Non-trivial intra-agent automorphism in Grid-Small3x3.

Symm. Type	State	Action	Obs.
Intra (contd.)	$s_{AA} \leftrightarrow s_{II}, s_{AB} \leftrightarrow s_{IH}$		
	$s_{AC} \leftrightarrow s_{IG}, s_{AD} \leftrightarrow s_{IF}$		
	$s_{AE} \leftrightarrow s_{IE}, s_{AF} \leftrightarrow s_{ID}$		
	$s_{AG} \leftrightarrow s_{IC}, s_{AH} \leftrightarrow s_{IB}$		
	$s_{AI} \leftrightarrow s_{IA}, s_{BA} \leftrightarrow s_{HI}$		
	$s_{BB} \leftrightarrow s_{HH}, s_{BC} \leftrightarrow s_{HG}$		
	$s_{BD} \leftrightarrow s_{HF}, s_{BE} \leftrightarrow s_{HE}$		
	$s_{BF} \leftrightarrow s_{HD}, s_{BG} \leftrightarrow s_{HC}$		
	$s_{BH} \leftrightarrow s_{HB}, s_{BI} \leftrightarrow s_{HA}$	$a_{1,\text{UP}} \leftrightarrow a_{1,\text{DOWN}}$	
	$s_{CA} \leftrightarrow s_{GI}, s_{CB} \leftrightarrow s_{GH}$	$a_{2,\text{UP}} \leftrightarrow a_{2,\text{DOWN}}$	$z_{1,\text{LEFT}} \leftrightarrow z_{1,\text{RIGHT}}$
	$s_{CC} \leftrightarrow s_{GG}, s_{CD} \leftrightarrow s_{GF}$	$a_{1,\text{LEFT}} \leftrightarrow a_{1,\text{RIGHT}}$	$z_{2,\text{LEFT}} \leftrightarrow z_{2,\text{RIGHT}}$
	$s_{CE} \leftrightarrow s_{GE}, s_{CF} \leftrightarrow s_{GD}$	$a_{2,\text{LEFT}} \leftrightarrow a_{2,\text{RIGHT}}$	
	$s_{CG} \leftrightarrow s_{GC}, s_{CH} \leftrightarrow s_{GB}$		
	$s_{CI} \leftrightarrow s_{GA}, s_{DA} \leftrightarrow s_{FI}$		
	$s_{DB} \leftrightarrow s_{FH}, s_{DC} \leftrightarrow s_{FG}$		
	$s_{DD} \leftrightarrow s_{FF}, s_{DE} \leftrightarrow s_{FE}$		
	$s_{DF} \leftrightarrow s_{FD}, s_{DG} \leftrightarrow s_{FC}$		
	$s_{DH} \leftrightarrow s_{FB}, s_{DI} \leftrightarrow s_{FA}$		
	$s_{EA} \leftrightarrow s_{EI}, s_{EB} \leftrightarrow s_{EH}$		
	$s_{EC} \leftrightarrow s_{EG}, s_{ED} \leftrightarrow s_{EF}$		

Figure 22: Non-trivial intra-agent automorphism in Grid-Small3x3.

State	Action	Obs.
$s_{\text{LBox}} \leftrightarrow s_{\text{RBox}}$, $s_{01uu} \leftrightarrow s_{23uu}$ $s_{01ud} \leftrightarrow s_{23du}$, $s_{01ul} \leftrightarrow s_{23ru}$ $s_{01ur} \leftrightarrow s_{23lu}$, $s_{01du} \leftrightarrow s_{23ud}$ $s_{01dd} \leftrightarrow s_{23dd}$, $s_{01dl} \leftrightarrow s_{23rd}$ $s_{01dr} \leftrightarrow s_{23ld}$, $s_{01lu} \leftrightarrow s_{23ur}$ $s_{01ld} \leftrightarrow s_{23dr}$, $s_{01ll} \leftrightarrow s_{23rr}$ $s_{01lr} \leftrightarrow s_{23lr}$, $s_{01ru} \leftrightarrow s_{23ul}$ $s_{01rd} \leftrightarrow s_{23dl}$, $s_{01rl} \leftrightarrow s_{23rl}$ $s_{01rr} \leftrightarrow s_{23ll}$, $s_{02uu} \leftrightarrow s_{13uu}$ $s_{02ud} \leftrightarrow s_{13du}$, $s_{02ul} \leftrightarrow s_{13ru}$ $s_{02ur} \leftrightarrow s_{13lu}$, $s_{02du} \leftrightarrow s_{13ud}$ $s_{02dd} \leftrightarrow s_{13dd}$, $s_{02dl} \leftrightarrow s_{13rd}$ $s_{02dr} \leftrightarrow s_{13ld}$, $s_{02lu} \leftrightarrow s_{13ur}$ $s_{02ld} \leftrightarrow s_{13dr}$, $s_{02ll} \leftrightarrow s_{13rr}$ $s_{02lr} \leftrightarrow s_{13lr}$, $s_{02ru} \leftrightarrow s_{13ul}$ $s_{02rd} \leftrightarrow s_{13dl}$, $s_{02rl} \leftrightarrow s_{13rl}$ $s_{02rr} \leftrightarrow s_{13ll}$, $s_{03ud} \leftrightarrow s_{03du}$ $s_{03ul} \leftrightarrow s_{03ru}$, $s_{03ur} \leftrightarrow s_{03lu}$ $s_{03dl} \leftrightarrow s_{03rd}$, $s_{03dr} \leftrightarrow s_{03ld}$ $s_{03ll} \leftrightarrow s_{03rr}$, $s_{12ud} \leftrightarrow s_{12du}$ $s_{12ul} \leftrightarrow s_{12ru}$, $s_{12ur} \leftrightarrow s_{12lu}$ $s_{12dl} \leftrightarrow s_{12rd}$, $s_{12dr} \leftrightarrow s_{12ld}$ $s_{12ll} \leftrightarrow s_{12rr}$	$a_{1,\text{LEFT}} \leftrightarrow a_{2,\text{LEFT}}$ $a_{1,\text{RIGHT}} \leftrightarrow a_{2,\text{RIGHT}}$ $a_{1,\text{MOVE}} \leftrightarrow a_{2,\text{MOVE}}$ $a_{1,\text{STAY}} \leftrightarrow a_{2,\text{STAY}}$	$z_{1,\text{EMPTY}} \leftrightarrow z_{2,\text{EMPTY}}$ $z_{1,\text{WALL}} \leftrightarrow z_{2,\text{WALL}}$ $z_{1,\text{OTHER}} \leftrightarrow z_{2,\text{OTHER}}$ $z_{1,\text{SMALL}} \leftrightarrow z_{2,\text{SMALL}}$ $z_{1,\text{LARGE}} \leftrightarrow z_{2,\text{LARGE}}$

Figure 23: Non-trivial automorphism in Box-Pushing. It is an inter-agent automorphism.