Evolving Intertask Mappings for Transfer in Reinforcement Learning

Minh Hua
Department of Computer Science
The Johns Hopkins University
Baltimore, MD
mhua2@jhu.edu

John W. Sheppard
Gianforte School of Computing
Montana State University
Bozeman, MT
john.sheppard@montana.edu

***Abstract*—Recently, there has been a focus on using transfer learning to reduce the sample complexity in reinforcement learning. One component that enables transfer is an intertask mapping that relates a pair of tasks. Automatic methods attempt to learn task relationships either by evaluating all possible mappings in a brute force manner, or by using techniques such as neural networks to represent the mapping. However, brute force methods do not scale well in problems since there is an exponential number of possible mappings, and automatic methods that use complex representations generate mappings that are not always interpretable. In this paper, we describe a population-based algorithm that generates intertask mappings in a tractable amount of time. The idea is to use an explicit representation of an intertask mapping, and to combine an evolutionary algorithm with an offline evaluation scheme to search for the optimal mapping. Experiments on two transfer learning problems show that our approach is capable of finding highly-fit mappings and searching a space that is infeasible for a brute force approach. Furthermore, agents that learn using the mappings found by our approach are able to reach a performance target faster than agents that learn without transfer.**

I. Introduction

In reinforcement learning (RL), an agent explores an environment through trial and error to learn a sequence of actions that maximize a reward function [1]. Spurred by recent advances in computing capability and deep learning performance, contemporary RL approaches learn robust function approximators using deep neural networks to solve a variety of challenging tasks such as game playing and robotic control. Despite these successes, RL agents still face difficulties when they begin learning *tabula rasa*. Specifically, agents must spend a considerable amount of time exploring the environment in practical problems where the environment dynamics are unknown. Furthermore, complex environments with partial observability, complex state and action spaces, or sparse feedback complicate an agent's ability to collect a sufficient number of learning samples. Also, sample complexity must be minimized in domains where there is risk associated with sample collection, such as autonomous vehicles.

A promising approach to addressing the aforementioned problems is transfer learning (TL), which leverages knowledge from one or more source domains to facilitate learning in a target domain [2]. The key insight behind TL is that knowledge learned in one domain may be usable in a similar domain. This insight is not novel and has been studied extensively by psychologists for many years. At a minimum, transferring knowledge can spur learning in the target domain, and in some extreme cases, TL might even enable learning tasks that were otherwise unlearnable. TL has historically achieved success in supervised learning domains, while its applicability to RL is still being actively explored [3], [4].

For reinforcement learning, TL is usually done in the context of a Markov Decision Process (MDP), and approaches are categorized by their goals with transfer, the assumptions they make regarding task similarity, and the types of knowledge they transfer (e.g., expert experiences, transition samples, value functions, entire policies, etc.). The source and target tasks might exhibit differences in the action space, state variables, transition function, or reward structure. Thus, to transfer effectively, the agent must learn a relation, or mapping, between the source task and the target task. Taylor *et al.* [5] defined the relation between two tasks as a pair of intertask mappings, denoted χ_S (state mapping) and χ_A (action mapping), a definition that we adopt.

While TL methods have historically performed well with mappings provided by a human, there are some limitations to this approach. For example, there might be insufficient domain knowledge to define such a mapping, the state and action spaces might contain an exponential (or even infinite) number of instances, or the agent might be fully autonomous. In this paper, we focus on the problem of learning the mapping automatically. Several approaches have been proposed to address this issue, but as we shall discuss later, more work can be done to address the learning algorithm's computational complexity and the interpretability of the learned mappings.

The main contribution of this paper is a novel population-based approach to generate explicit intertask mappings (which directly state the relation between two tasks). We also propose a simpler variant that uses random-mutation hill-climbing.

II. Problem Statement and Hypothesis

In addressing the intertask mapping problem, we assume that each source task and target task is a Markov Decision Process (MDP). That said, we treat one of the learning problems,

979-8-3503-1458-8/23/$31.00 ©2023 IEEE

Robot Soccer Keepaway, as a semi-Markov Decision Process (SMDP), where agents apply "macro-actions" that last for several time steps. We also assume that the agent can learn a near-optimal policy in the source task and can collect samples from both tasks. In addition, we assume that the Euclidean distance is an appropriate measure for the similarity between states, and state variables can be scaled so that they are weighted equally. Finally, we assume that the two MDPs exhibit similar reward structures. With these assumptions established, we can define the problem (adapted from [6]) as follows: Given two MDPs $\mathcal{M}_1 = \langle \mathcal{S}_1, \mathcal{A}_1, \mathcal{T}_1, \mathcal{R}_1, \gamma_1 \rangle$ and $\mathcal{M}_2 = \langle \mathcal{S}_2, \mathcal{A}_2, \mathcal{T}_2, \mathcal{R}_2, \gamma_2 \rangle$, where $\mathcal{S}_i, \mathcal{A}_i, \mathcal{T}_i, \mathcal{R}_i, \gamma_i$ correspond to the state space, action space, transition model, reward function, and discount factor of the i-th MDP, learn two mappings $\chi_S : \mathcal{S}_2 \to \mathcal{S}_1$ and $\chi_A : \mathcal{A}_2 \to \mathcal{A}_1$. Thus we learn mappings from the target to the source. When necessary, we also work with inverse mappings.

We compare three approaches to learning intertask mappings: a genetic algorithm, named *Genetic Algorithms for Mapping Evolution* (GAME), a random-mutation hill-climbing variant of GAME (GAME-RMHC), and a brute force algorithm (MASTER) [7].

We test five hypotheses:

- **H1**: GAME and GAME-RMHC will find intertask mappings with offline evaluation scores that are no worse than the mappings found by the MASTER, and GAME will find intertask mappings with offline evaluation scores that are higher than those found by GAME-RMHC.
- **H2**: GAME-RMHC will have the lowest computational complexity, followed by GAME and MASTER as determined by measuring the number of comparisons (a comparison is counted each time a mapping is compared against another, or a nontrivial operation is performed such as mutation or crossover) and fitness evaluations (FEs) performed before convergence (i.e., the point where the best fitness does not change).
- **H3**: an agent that uses an intertask mapping will take less time learning (measured by training episodes) to attain a performance threshold in the target task, relative to an agent that does not use this mapping. Note that, for one domain, we use simulator hours to measure training time.
- **H4**: an agent that uses the mapping learned by GAME will take *at most* as much time learning to attain a performance threshold in the target task as an agent that uses a mapping learned by the MASTER. Conversely, an agent that uses the mapping produced by GAME-RMHC will take *at least* as much time to attain a performance threshold in the target task as an agent that uses either the mapping learned by MASTER or GAME.
- **H5**: the mapping with the highest offline score will reduce learning time in the target task the most, and vice versa.

III. Related Work

A. Intertask Mappings

Most works focused on intertask mappings assume a single mapping over the entire state-action space. However, there is also interest in learning to select between multiple intertask mappings [8]. Taylor *et al.* [5] assumed that each action and state variable in the target task has a unique correspondence in the source task and used intuition to design the mappings manually, which were then applied to transfer different types of knowledge between tasks (e.g., the action-value function). Similarly, Torrey *et al.* [9] used an expert-designed mapping to transform a learned model from one task into advice for a new task. Furthermore, Taylor *et al.* [10] utilized intertask mappings to transfer source task instances to assist model-based RL algorithms like FITTED R-MAX with model approximation. Observing that a state variable in the target task may be related to more than one state variable in the source task, Cheng *et al.* [11] proposed a mapping that utilizes a linear combination of related state variables in the source task.

B. Autonomous Learning of Intertask Mappings

Directly related to our research, *Mapping Learning via Classification* assumes that state variables can be arranged into task-independent groupings provided by an expert [12]. Then, classifiers are trained on subsets of the state variables that define a particular object. Once trained, these classifiers define the state and action mappings. Although this approach shows promise, the need for expert knowledge in the form of state variable groupings precludes it from being fully autonomous.

In contrast, the *Modeling Approximate State Transitions by Exploiting Regression (MASTER)* algorithm enumerates all possible state and action mappings and evaluates them offline using a model of the state transition dynamics in the target task [7]. Samples are collected from the source and target tasks, and neural networks are trained to predict the successor state given a state and action pair in the target task. Then, every possible pair of state and action mappings is used to transform source samples into target samples. These samples are then evaluated using the learned transition models, and the mapping that produced samples that most closely match the approximated target task transition dynamics is picked as the best mapping. Although the results were promising, the authors observed that the algorithm's exhaustive search scales exponentially and suggested that a heuristic search is needed. Our work addresses this issue directly.

Subsequent work employs more advanced methods to represent the intertask mappings, or learns mappings over spaces different from the state or action space. Da Silva and Costa [13] used Object-Oriented MDPs to estimate mappings without the use of classifiers. To address scalability, Ammar *et al.* [14] used sparse coding, sparse projection learning, and sparse Gaussian processes to learn an intertask mapping over the state transition dynamics. Ammar *et al.* [15] used restricted Boltzmann machines to encode intertask mappings. However, this approach has been criticized for being computationally expensive. More recent work utilizes feedforward networks to represent intertask mappings implicitly [16]. Gupta *et al.* [17] used encoder-decoder neural networks to learn an agent-specific feature space that can be used to transfer skills. However, using a network representation diminishes the interpretability of the intertask mapping. In this paper, we are motivated to learn intertask mappings that *explicitly* state the relation between tasks.

C. Evolutionary Reinforcement Learning

Our proposed approach falls within the purview of Evolutionary Reinforcement Learning (ERL). Moriarty *et al.* [18] published a survey on Evolutionary Algorithms for Reinforcement Learning (EARL), a class of algorithms that applies an evolutionary approach to evolving optimal policies. The authors summarized several prominent EARL algorithms and discussed their strengths and limitations. That said, Bishop and Miikkulainen [19], used an evolutionary algorithm to search the space of feature subsets to rank features for online RL, which is much closer to our approach. There has also been work that applied population-based methods explicitly to TL in RL, but their focus was on multiagent settings for stock trading [20]. To the best of our knowledge, our work is the first to apply evolutionary algorithms to learn intertask mappings.

Due to the representation that we chose for our mappings, we can treat the learning problem as a combinatorial optimization problem (COP), where the task is to search for a combination of state and action mappings that maximizes the offline evaluation score. Our approach uses genetic operators proposed by Chu [21], which solved four NP-Hard COPs using a GA-based approach.

IV. Approach

A. Problem Domains Studied

The first source task we considered was two-dimensional Mountain Car (MC2D), where a car travels along the curve $\sin(3x)$ in the range $-1.2 \leq x \leq 0.6$. The state variables are the car's horizontal position x and its velocity $\dot{x}$. The possible actions are $\{$`Left`, `Neutral`, `Right`$\}$, which change the car's velocity by -0.001, 0, and 0.001, respectively. To simulate the force of gravity on the car, $-0.025(\cos(3x))$ is added to $\dot{x}$ at each time step. The car starts at the bottom of the hill, and the goal states are locations where $x \geq 0.5$.

The corresponding target task we considered was three-dimensional Mountain Car (MC3D), where the mountain's curve is extended to a surface defined by $\sin(3x) + \sin(3y)$. We added two state variables corresponding to the position y and the velocity $\dot{y}$. The possible actions are $\{$`Neutral`, `West`, `East`, `South`, `North`$\}$. `West` and `East` modify $\dot{x}$ by -0.001 and 0.001 respectively, while `South` and `North` modify $\dot{y}$ by -0.001 and 0.001 respectively. The goal states are locations with coordinates $x \geq 0.5$ and $y \geq 0.5$.

An episode in each task terminates when the car reaches the goal state or if 5,000 steps have been taken. Each step incurs a reward of -1 except when the car reaches the goal state, in which case the agent earns a reward of 0. We used the Mountain Car environment available in OpenAI GYM [22] for MC2D and implemented our own version of MC3D in GYM.

The second problem domain we considered was Robot Soccer Keepaway [23]. In Keepaway, a team of keepers must maintain possession of a ball within a limited region while another team of takers attempts to steal the ball or force it out of bounds. Each keeper is controlled by a separate agent while each taker follows a hand-coded strategy. Each agent is rewarded for the number of time steps that the ball remains in play after the agent acts. Each time step lasts for $100\,\mathrm{ms}$ in simulation time and the field's dimensions are $25\,\mathrm{m} \times 25\,\mathrm{m}$.

The Keepaway state space consists of several variables that represent the angles and distances between the current player, their teammates, and the takers. A keeper may only take an action when it possesses the ball; otherwise, the keepers run a hand-coded routine to try to get open for passing. Therefore, the state variables are focused on the keeper currently possessing the ball, K_1, who may hold the ball or pass it to one of its teammates. The action space is $\{\texttt{Hold}, \texttt{Pass}_i\}$, where $\texttt{Pass}_i$ passes the ball to the i-th nearest teammate. The closest teammate to K_1 is K_2, and so on. Similarly, T_1 is the closest taker to K_1, and so on. C denotes the center of the field. Besides the state variables that involve distances between two objects, there are two further types of state variables: $\min(dist(K_i, T_1), \ldots, dist(K_i, T_n))$ is the distance from the i-th nearest teammate to the nearest taker, while $\min(ang(K_i, K_1, T_1), \ldots, ang(K_i, K_1, T_n))$ is the angle of the passing lane between the keeper currently possessing the ball and the i-th closest teammate.

Our first source task for this domain is 3 vs. 2 Keepaway, where three keepers play against two takers. There are 13 state variables and three possible actions. More complex variants add more players, which increases the number of state variables as well as the number of possible actions. The corresponding target task we investigate is 4 vs. 3 Keepaway, which has 19 state variables and four actions. We used Aijun Bai's implementation of Keepaway as our testbed.[1]

B. The GAME Algorithm

Our approach incorporates the intertask mapping definition and the offline evaluation method proposed by Taylor *et al.* [7]. We also define the following notation. Let $S_{\text{source}} = \{s_{s,1}, s_{s,2}, \ldots, s_{s,n}\}$ be a set of n state variables in the source task. Let $S_{\text{target}} = \{s_{t,1}, s_{t,2}, \ldots, s_{t,m}\}$ be a set of m state variables in the target task. Let $A_{\text{source}} = \{a_{s,1}, a_{s,2}, \ldots, a_{s,u}\}$ be a set of u actions in the source task. Let $A_{\text{target}} = \{a_{t,1}, a_{t,2}, \ldots, a_{t,v}\}$ be a set of v actions in the target task.

The first step in our approach is to create an offline evaluation mechanism for the intertask mappings. We first train an agent on the source task for p episodes and collect observation tuples to create a dataset D_{source}. Then, we allow an agent to explore the target task for a small number of episodes $q \ll p$ and collect observation tuples to create a dataset D_{target}. For all sets of tasks, we used a Sarsa(λ) agent with CMAC tile coding function approximation. After creating D_{target}, we learn transition models of the target task $T_{a_{t,i}, s_{t,j}}(s) \mapsto s'_{t,j}$ using feedforward neural networks, where $i = 1, \ldots, v$ and $j = 1, \ldots, m$. Each network takes as input a vector of the current state variables and approximates the effect of one

[1]https://github.com/aijunbai/keepaway

target action $(a_{t,i})$ to predict the next value for one target state variable $(s'_{t,j})$. Thus, if the target task has v actions and m state variables, then we will require $v \times m$ neural networks.

After training the networks, we use them to evaluate the intertask mappings generated by the three algorithms. During each iteration, we use the inverse of the state and action mappings currently being evaluated to transform tuples from D_{source} into "mapped tuples," creating a dataset D'_{source}. Specifically, D'_{source} has as its columns the target task's current and next state variables and the current action, but the actual values for the state variables come from the source task and are populated according to the mapping. An action $a_s \in D_{\text{source}}$ becomes an action $a_t \in D'_{\text{source}}$ according to the action mapping. Then, for every tuple $(s, a, s') \in D'_{\text{source}}$, we calculate the mean squared error (MSE) between the predicted successor state $T_{a_{t,i}, s_{t,j}}(s)$ and the actual successor state $s'_{t,j}$.

We evaluate three different methods to generate intertask mappings automatically. First, we evaluate MASTER [7], which enumerates all possible intertask mappings. MASTER evaluates n^m different state mappings, and for each state mapping, MASTER evaluates $u \times v$ single action mappings using an offline evaluation scheme similar to the one explained above. Note that a single action mapping refers to a relation between one action in the source task and one action in the target task. A full action mapping (called an action mapping in this paper), however, refers to a set of v single action mappings that maps all the actions in the target task. There are $u \times v$ possible single action mappings but u^v possible combinations of single action mappings. Thus, each mapping has $u \times v$ different MSE values, and we score each state mapping using the average $1 - \text{MSE}$ values over all single action mappings.

Our population-based method is called *Genetic Algorithm for Mapping Evolution* (GAME) and uses a non-binary genetic algorithm (GA) to evolve a population of intertask mapping individuals, $\rho_k = \{\chi_{S,k}, \chi_{A,k}\}$. A state mapping $\chi_S : S_{\text{target}} \rightarrow S_{\text{source}}$ assigns each $s_{t,j} \in S_{\text{target}}$ to a corresponding $s_{s,i} \in S_{\text{source}}$. Thus, a state mapping is a list of length $|S_{\text{target}}|$, where the j-th element corresponds to the source task state variable mapped to the j-th target task state variable. An action mapping $\chi_A : A_{\text{target}} \rightarrow A_{\text{source}}$ is defined in a similar fashion. For example, assume that we order the MC3D state variables and actions as $\{x, \dot{x}, y, \dot{y}\}$ and $\{\texttt{Neutral}, \texttt{West}, \texttt{East}, \texttt{South}, \texttt{North}\}$. Then, a state mapping $\chi_S = \{x, \dot{x}, x, \dot{x}\}$ maps the state variables x and y in the target task to the state variable x in the source task and maps the state variables $\dot{x}$ and $\dot{y}$ in the target task to the state variable $\dot{x}$ in the source task. Similarly, an action mapping $\chi_A = \{\texttt{Left}, \texttt{Left}, \texttt{Left}, \texttt{Right}, \texttt{Right}\}$ maps the target actions `Neutral`, `West`, and `East` to the source action `Left` and maps the target actions `South`, `North` to the source action `Right`. In this sense, GAME formulates the problem of learning an intertask mapping as one of searching for the combination of state and action mappings that maximizes a fitness value.

The fitness of each individual ρ_k is defined by its ability to generate samples that match the approximated target task transition dynamics. Thus, the aforementioned offline evaluation scheme is used as the fitness function for GAME. After using the inverse of $\chi_{S,k}$ and $\chi_{A,k}$ to create D'_{source} and running the transition networks for prediction, we average the $v \times m$ individual $1 - \text{MSE}$ values to determine the fitness of the intertask mapping.

We initialize a population of mappings randomly. Each gene in a state (action) mapping chromosome is sampled uniformly from the set of source task state variables (actions). In addition, we use top-k elitism, a standard generational GA model, and tournament selection to control the selection pressure.

We evaluated GAME with the one-point and fusion crossover operators [21]. For each crossover operator, two individuals may cross their state mappings or action mappings, but not cross a state mapping with an action mapping. The fusion crossover operator produces only one offspring where, at each gene, the offspring inherits the value of the parents' genes if they are the same. Otherwise, the offspring inherits the gene from the higher-fitness parent with a higher probability. We are motivated by the fact that although we are not certain of the single state mapping that performed the best, the individual with the higher fitness should have a larger proportion of good single state mappings. Finally, we use uniform mutation with a constant mutation rate, where each gene in the state and action mapping chromosomes has a constant probability of mutating.

We also evaluated a random-mutation hill-climbing variant of GAME to see if the complexity of our population-based method is justified. GAME-RMHC uses the same mapping representation as GAME, but the population contains only one individual (i.e., it is effectively a (1+1)-evolution strategy). At each generation, GAME-RMHC randomly mutates one state mapping and one action mapping to produce one offspring, which replaces the parent if its fitness is higher.

We employed two transfer learning methods. For Mountain Car, we used Q-Value Reuse, which modifies the action-value computation in the target task with CMAC weights from the source task [7]. Given a target state s_t, a target action a_t, and an intertask mapping $\rho = \{\chi_S, \chi_A\}$, the action-value for the pair (s_t, a_t) is computed as $Q(s_t, a_t) = Q_{\text{target}}(s_t, a_t) + Q_{\text{source}}(\chi_S(s_t), \chi_A(a_t))$ where Q_{target} uses the CMAC weights of the MC3D agent and Q_{source} uses the CMAC weights of the MC2D agent. For Keepaway, the learned mappings are used to initialize the CMAC weights of an agent in the target task with weights from a trained agent in the source task [12]. For each weight in the source agent, we determine the source state variable and action that activate that weight. Then, we use the inverse intertask mapping to look up the corresponding target CMAC weight that is activated. We then copy the weights from the source agent to the corresponding weights in the target agent. Otherwise, the remaining weights in the target agent are initialized to zero. We randomly select one agent trained on 3 vs. 2 Keepaway as the source material for transfer.

C. Experimental Design

After the agent explores the target task environment for q episodes, we generated a dataset D_{target} of transition samples.

We split 20% of D_{target} into a test set and the remaining 80% into a training set. We then tuned the networks over the entire training set using grid search with five-fold cross validation.

For each Mountain Car agent, we used Sarsa(λ) and tuned the learning rate α, trace decay rate λ, discount factor γ, exploration rate ϵ, and the number of CMAC tilings. For each parameter setting, we evaluated the agent for 250 episodes. The best parameters found for the MC3D Sarsa(λ) agent were $\alpha = 0.75, \lambda = 0.99, \gamma = 1, \epsilon = 0.01$, and eight CMAC tilings. We used these parameter settings for the MC3D agent without transfer learning, while each transfer learning agent was re-tuned using the same parameter ranges. Due to time constraints, we were not able to tune the Keepaway agents. Recognizing this as a limitation, we consulted the literature and chose $\alpha = 0.1, \lambda = 0, \gamma = 1, \epsilon = 0.01$, and 32 tilings.

For GAME, we tuned the population size, crossover rate, mutation rate, crossover operator, tournament size, and k in top-k elitism. We did not tune GAME on Mountain Car because initial experiments demonstrated that the optimal state mapping was frequently found during the initialization process. To run GAME on Mountain Car, we used a population size of 10, a crossover rate of 0.8, a mutation rate of 0.14, the fusion crossover operator, a tournament size of 4, and top-1 elitism. Due to the complexity of Keepaway, we tuned GAME by running the algorithm with different parameter combinations for a maximum of 200 FEs. The best parameters found for GAME on Keepaway were a population size of 100, a crossover rate of 0.8, a mutation rate of 0.04, the fusion crossover operator, a tournament size of 10, and top-5 elitism.

To test **H1** and **H2**, we ran MASTER to obtain baseline results; however, this was only feasible for Mountain Car as Keepaway would have required us to evaluate $4 \times 3 \times 13^{19}$ intertask mappings. Then, we ran GAME and GAME-RMHC with the best parameters in 10 independent trials. For each trial, we ran GAME and GAME-RMHC for 240 FEs on Mountain Car and 2,000 FEs on Keepaway. We measured each approach's complexity through the number of comparisons and fitness evaluations performed before convergence.

To test **H3–H5**, we first trained agents in the source tasks and saved their weights. On MC2D, we trained the source agent and collected transitions for 100 episodes. For 3 vs. 2 Keepaway, we trained the source agent and collected samples for four simulator hours. Next, we trained agents without transfer on the target tasks until a performance threshold was reached for 30 independent trials. We chose to run 30 trials based on standard statistical guidance assuming the results are likely to follow a normal distribution. For MC3D, the performance threshold was to reach the goal state within an average of 1,000 steps over 10 episodes. We chose this threshold because the minimum number of steps required to solve MC3D usually falls within 1,000 steps. For 4 vs. 3 Keepaway, the performance threshold was the ability to keep the ball for an average of more than nine seconds over 1,000 episodes. We chose this threshold because it was close to the best performance achieved within the training budget without transfer.

TABLE I
AVERAGE NUMBER OF COMPARISONS AND FITNESS EVALUATIONS

	MC3D		**Keepaway**	
	Comparisons	FEs	Comparisons	FEs
GAME	2150.2	145.4	62 438.8	1840
GAME-RMHC	**126.6**	**63.3**	**3567.8**	**1783.9**
MASTER	240	240	$\Omega(12 \cdot 13^{19})$	

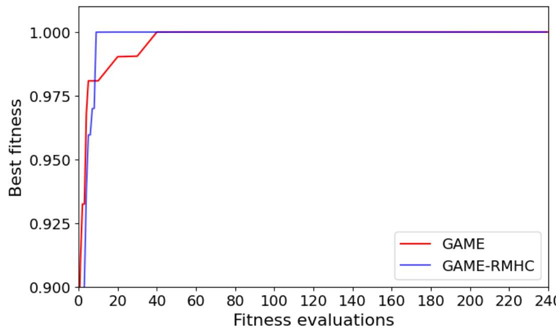

Fig. 1. Learning curves on MC3D for GAME and GAME-RMHC.

To train the transition networks, we collected transition samples for 50 episodes in MC3D and 1,000 episodes in 4 vs. 3 Keepaway. Then, we used the learned mappings to train agents with transfer. For each problem, we also evaluated an agent with a hand-coded mapping informed by intuition. We evaluated each transfer agent in 30 independent trials.

V. RESULTS

A. *Evolving Intertask Mappings*

Experiments evolving intertask mappings for Mountain Car demonstrate that GAME and GAME-RMHC are able to find mappings with scores that are no worse than the score of the mapping found by MASTER, supporting **H1**. Specifically,, GAME and GAME-RMHC both converged to the mapping with the highest offline evaluation score at 0.999949 in every trial. MASTER converged to the same state mapping as GAME, but this intertask mapping's fitness was slightly lower at 0.999946 due to its averaging over all single action mappings. Table I shows the average number of comparisons and FEs required by each algorithm before the best solution was found in each trial. On Mountain Car, GAME incurred the highest number of comparisons, exceeding even MASTER. GAME-RMHC, however, was the quickest to converge, requiring an average of only 63.3 FEs before the best solution was found, partially supporting **H2**. This result is confirmed by Figure 1, which shows the best fitness found by GAME and GAME-RMHC as a function of FEs.

In contrast, the Keepaway results do not support the second part of **H1** that the scores of the mappings found by GAME will be higher than the scores of the mappings found by GAME-RMHC. Specifically, supporting **H2**, GAME-RMHC converged faster than GAME (Figure 2) and converged to mappings with higher offline scores on all 10 Keepaway trials (see Tab. II). A Wilcoxon rank-sum test ($\alpha = 0.05$) confirms

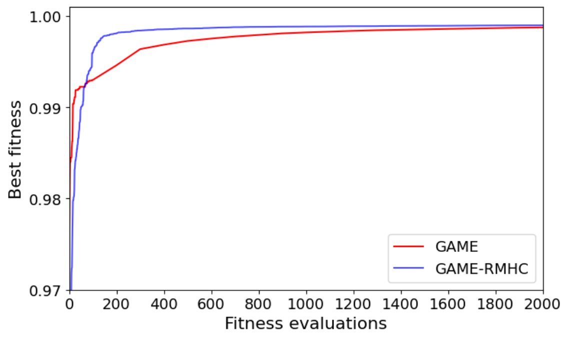

Fig. 2. Learning curves on 4 vs. 3 Keepaway for GAME and GAME-RMHC.

that the fitness scores of the mappings found by GAME-RMHC are higher than the fitness scores of the mappings found by GAME. In addition, GAME-RMHC had the lowest number of comparisons and FEs out of all three methods.

The hand-coded Mountain Car state mapping is $\{x_{\text{target}}, y_{\text{target}}\} \rightarrow x_{\text{source}}$ and $\{\dot{x}_{\text{target}}, \dot{y}_{\text{target}}\} \rightarrow \dot{x}_{\text{source}}$. There is no best hand-coded action mapping because the source actions Left and Right could each map to West, East, South, and North in the target task depending on the car's orientation. However, if the car was facing the goal state, the most intuitive action mapping would be $\texttt{Neutral}_{\text{target}} \rightarrow \texttt{Neutral}_{\text{source}}$, $\{\texttt{West}, \texttt{South}\} \rightarrow \texttt{Left}$, and $\{\texttt{East}, \texttt{North}\} \rightarrow \texttt{Right}$. Interestingly, the fitness for the hand-coded mapping is not optimal at 0.999944.

GAME, GAME-RMHC, and MASTER all converged to the hand-coded state mapping. Given this state mapping, the action mapping with the highest fitness maps $\{\texttt{Neutral}_{\text{target}}, \texttt{West}, \texttt{South}\} \rightarrow \texttt{Neutral}_{\text{source}}$ and $\{\texttt{East}, \texttt{North}\} \rightarrow \texttt{Right}$, which was found by GAME and GAME-RMHC. This action mapping is not perfect, but we observe that three of the target actions are mapped in the same way as the hand-coded action mapping. Given this result, we expect that an agent that uses this mapping might perform poorly during transfer learning due to the fact that the source action Left is not utilized and two target actions are not mapped correctly. However, that the fitness of the hand-coded action mapping was within 10^{-4} of this action mapping confirms our intuition that there is no optimal action mapping, which would have made it difficult for the offline evaluation scheme to identify the hand-coded action mapping as optimal.

The most intuitive 4 vs. 3 Keepaway state mapping assigns each target state variable to the source state variable with the most semantic similarity, e.g. $dist(K_1, C)_{\text{target}} \rightarrow dist(K_1, C)_{\text{source}}$. The only exceptions are the state variables that are novel to 4 vs. 3 Keepaway. We assigned these state variables to the most similar state variables in the source task. The mapping for the actions followed a similar process. This hand-coded mapping is also not optimal, having an offline fitness value of only 0.99427. The best mapping evolved by GAME had an offline evaluation score of 0.99886, and the best mapping evolved by GAME-RMHC had an offline score of 0.9991. Although these mappings have higher offline evaluation scores than the hand-coded mapping, they are more difficult to understand due to the seemingly random matching of the state variables. Due to the exponential number of possible state mappings on Keepaway, we could not feasibly run MASTER during this part of the experiment, which reinforces a weakness with its brute force approach.

Despite evolving a state mapping with a lower fitness score, five of the single state mappings evolved by GAME matched the hand-coded state mapping. That said, only two of the single state mappings evolved by GAME-RMHC matched the hand-coded state mapping. Furthermore, both GAME and GAME-RMHC erroneously mapped source state variables involving angles to target state variables involving distances, and vice versa. Finally, both GAME and GAME-RMHC mapped every target action to the source action $\texttt{Pass}_1$, which is reminiscent of a similar problem on MC3D where many target actions were mapped to the same source action. However, transfer learning experiments show that the agents that utilize the learned mappings perform better than the baseline agent.

B. *Transfer Learning*

Figure 3 graphs the learning curves for the MC3D agents with and without transfer learning. Each data point is averaged over 30 independent trials and represents a moving window of 10 episodes. The first agent to reach the threshold performance on MC3D was the agent using the hand-coded mapping. Despite the imperfect action mapping found by GAME and GAME-RMHC, however, a Wilcoxon rank-sum test ($\alpha = 0.05$) confirms that the agents that used the best mapping found by GAME and GAME-RMHC performed comparably to the agent using the transfer learning method employed by MASTER, supporting the first part of **H4**. Additional Wilcoxon rank-sum tests ($\alpha = 0.05$) show that the time-to-threshold of *all* transfer learning agents were less than the time-to-threshold of the agent without transfer, supporting **H3**.

Shifting the transfer learning curves by 100 episodes to account for the pre-training in the source task, we see that transfer learning provided significant benefit to all transfer learning agents. Although the Q-Value Reuse method employed by MASTER allowed it to match the performance of GAME on Mountain Car, it has a serious deficiency in that it requires the enumeration of every single action mapping, which might not be feasible in complex problem domains. Nevertheless, we predict that the mappings found by GAME and GAME-RMHC would have outperformed the mapping found by MASTER had the offline evaluation scheme been more accurate and instead assigned the highest fitness to the hand-coded action mapping.

Figure 4 shows the learning curves for the 4 vs. 3 Keepaway agents with and without transfer. Each data point is the average of 30 independent trials and represents a moving window of 1,000 episodes. The first set of keepers to hold the ball for more than nine seconds used the hand-coded mapping. Wilcoxon rank-sum tests ($\alpha = 0.05$) demonstrate that the agents that used the mappings evolved by GAME and GAME-RMHC performed worse than the agent that uses the hand-

TABLE II
FITNESS OF BEST KEEPAWAY MAPPINGS AVERAGED ACROSS 10 TRIALS

	1	2	3	4	5	6	7	8	9	10
GAME	0.99874	0.99875	0.99882	0.99884	0.99886	0.99884	0.99876	0.99866	0.99862	0.99866
GAME-RMHC	0.99892	0.99891	0.99908	0.99894	0.99903	0.99910	0.99897	0.99879	0.99902	0.99909

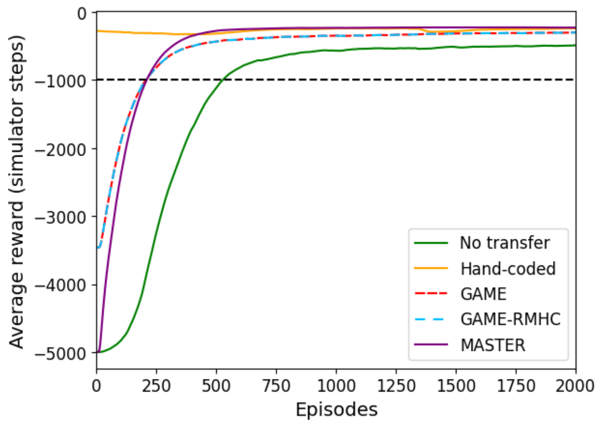

Fig. 3. Learning curves on MC3D with and without transfer.

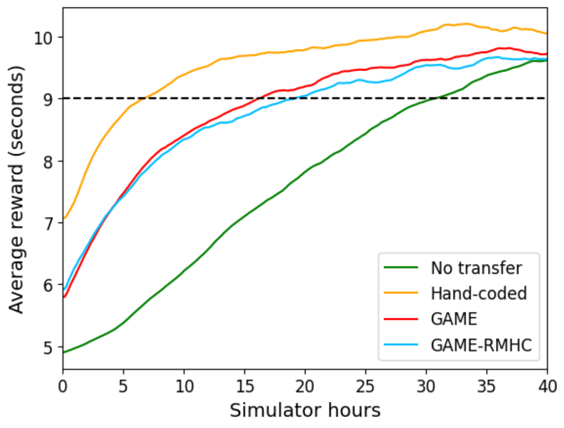

Fig. 4. Learning curves on 4 vs. 3 Keepaway with and without transfer.

coded mapping, but better than the agent without transfer learning. In addition, the agents that used the mapping evolved by GAME and GAME-RMHC performed comparably, which does not support the second part of **H4**. We also verified our transfer learning method by evaluating agents with weights sampled randomly from the source agent. These agents performed comparably to the agents without transfer. Shifting the transfer learning curves by four simulator hours to account for the pre-training in the source task illustrates that transfer learning yielded moderate benefits for the agents that used the mappings evolved by GAME and GAME-RMHC. These modest results stem from the fact that the action mappings found by GAME and GAME-RMHC once again ignored many source actions and mapped many target actions to the same source action. The fault lies with the offline evaluation scheme, which provided an inaccurate fitness signal. In fact, combining the state mapping found by GAME with the hand-coded action mapping resulted in a lower fitness of 0.997, to which GAME would not have converged.

VI. DISCUSSION

The results support our hypothesis that an agent that uses transfer will learn faster than an agent that does not use any transfer. On Mountain Car, the results support our hypothesis that an agent that uses the mappings evolved by GAME and GAME-RMHC will take at most as much time learning as an agent that uses the mapping found by MASTER. On Keepaway, the results show that an agent that uses the mapping evolved by GAME-RMHC takes as much time to reach the performance threshold as an agent that uses the mapping evolved by GAME, but does not support the hypothesis that the latter will outperform the former. Finally, the results do not support our hypothesis that the offline evaluation scheme will be correlated with time-to-threshold (**H5**).

Perhaps the most puzzling results were that the hand-coded mappings had the lowest offline scores, and the mappings with the highest offline scores mapped all the target actions to one single source action. First, in complex tasks like Keepaway, the offline evaluation scheme is not an accurate measure of how well an intertask mapping might perform, which makes it an unreliable fitness signal. Our experiments indicate that the offline evaluation scheme works better in tasks with semantically distinct state variables like Mountain Car.

On Keepaway, many of the state variables were either distances or angles, which made it hard to determine whether it is better to map $dist(K_1, K_2)_{\text{target}}$ to $dist(K_1, K_2)_{\text{source}}$, $dist(K_1, T_1)_{\text{source}}$, or $dist(K_1, C)_{\text{source}}$. One solution is to use an online evaluation scheme where each mapping's fitness is related to the reward received by an agent that uses this mapping. In this sense, GAME can identify the high-fitness mappings as promising candidates and the online evaluation scheme can then be used to search within this reduced pool of mappings. Alternatively, we can employ the state mapping found by GAME with algorithms like COMBREL, which uses multiple action mappings [8]. Additionally, we can incorporate a Quality-Diversity approach to facilitate the generation of diverse and high-quality mappings to combat the accumulation of homogeneous action mappings in the final population [24].

The transition networks most likely led to the hand-coded mappings' low scores and the phenomenon where every target action was mapped to one source action. On Keepaway, the hand-coded mapping's single state mapping with the lowest score at 0.98489 was for the prediction of $dist(K_1, K_3)$, despite the fact that the single state mapping correctly mapped

$dist(K_1, K_3)_{\text{target}} \rightarrow dist(K_1, K_3)_{\text{source}}$. In contrast, GAME and GAME-RMHC had scores of 0.9994 and 0.99949 respectively despite mapping $dist(K_1, K_3)_{\text{target}}$ to $dist(K_3, C)_{\text{source}}$. Since an intertask mapping is evaluated by how well the mapped dataset D'_{source} conforms to the transition neural networks, it is highly probable that the mapped datasets generated by GAME and GAME-RMHC better matched the original dataset on which the neural networks were trained. Similarly, if every target action is mapped to one source action, then only the samples in D_{source} containing that source action gets evaluated in D'_{source}. Thus, if the samples corresponding to a specific source action were biased to the networks' training data, then the same situation can occur.

VII. Conclusions

Our work demonstrated that a metaheuristic approach can generate explicit intertask mappings in a tractable amount of time. Our GA and hill-climbing algorithms both converged to highly-fit mappings and explored a search space that was too massive for the brute force algorithm. However, weaknesses with the offline evaluation method undermined our transfer learning results. Specifically, the fitness function introduced a disconnect between a mapping's fitness and its performance in transfer learning. We will address this disconnect in the future.

Although we have demonstrated that our approach scales with the size of the mapping search space, we acknowledge that we could have evaluated problems with even more state variables and actions. Preliminary experiments with transfer from 6 vs. 5 to 7 vs. 6 Keepaway suggest that GAME can scale to larger problems, but we were unable to run enough trials due to time constraints. We also plan to compare our transfer method to ones that use different intertask mapping representations. We only compared our approach to MASTER due to time constraints and to our knowledge, MASTER is the only other algorithm that generates explicit intertask mappings. However, we acknowledge the importance of considering methods beyond GAME to understand the full benefits of transfer methods that rely on explicit mappings. Finally, we will also study how GAME works with RL algorithms other than Sarsa(λ), and investigate alternative complexity measures.

To improve GAME, we will focus on algorithms that are more sensitive to the differences between state variables and can better handle domains with complex decision processes. We also plan to incorporate auxiliary information to improve the model learning process. For example, on Keepaway, knowing the ball's location at each time step might have helped to distinguish between holding and passing the ball. Finally, we could have transformed the MSE values into a fitness signal that is more informative than an average. Given GAME's ability to converge tractably to mappings with high fitness, with a fitness signal to reflect the benefit of using the mapping during learning more accurately, we believe that GAME will be a competitive intertask mapping learning algorithm.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, 2018.

[2] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big Data*, vol. 3, no. 1, pp. 1–40, 2016.

[3] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey." *Journal of Machine Learning Research*, vol. 10, no. 7, 2009.

[4] Z. Zhu, K. Lin, and J. Zhou, "Transfer learning in deep reinforcement learning: A survey," *arXiv:2009.07888*, 2020.

[5] M. E. Taylor, P. Stone, and Y. Liu, "Transfer learning via inter-task mappings for temporal difference learning." *Journal of Machine Learning Research*, vol. 8, no. 9, 2007.

[6] H. B. Ammar, "Automated transfer in reinforcement learning," PhD Dissertation, Maastricht University, Department of Advanced Computing Sciences, 2013.

[7] M. E. Taylor, G. Kuhlmann, and P. Stone, "Autonomous transfer for reinforcement learning." in *Proceedings of the Autonomous Agents and Multi-Agent Systems Conference*, 2008, pp. 283–290.

[8] A. Fachantidis, I. Partalas, M. E. Taylor, and I. Vlahavas, "Transfer learning with probabilistic mapping selection," *Adaptive Behavior*, vol. 23, no. 1, pp. 3–19, 2015.

[9] L. Torrey, T. Walker, J. Shavlik, and R. Maclin, "Using advice to transfer knowledge acquired in one reinforcement learning task to another," in *Proceedings of the European Conference on Machine Learning*. Springer, 2005, pp. 412–424.

[10] M. E. Taylor, N. K. Jong, and P. Stone, "Transferring instances for model-based reinforcement learning," in *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2008, pp. 488–505.

[11] Q. Cheng, X. Wang, and L. Shen, "Transfer learning via linear multi-variable mapping under reinforcement learning framework," in *Proceedings of the 36th Chinese Control Conference (CCC)*, 2017, pp. 8795–8799.

[12] M. E. Taylor, "Autonomous inter-task transfer in reinforcement learning domains," PhD Dissertation, The University of Texas at Austin, Department of Computer Sciences, 2008.

[13] F. L. Da Silva and A. H. R. Costa, "Towards zero-shot autonomous inter-task mapping through object-oriented task description," in *Workshop on Transfer in Reinforcement Learning (TiRL)*, 2017.

[14] H. B. Ammar, K. Tuyls, M. E. Taylor, K. Driessens, and G. Weiss, "Reinforcement learning transfer via sparse coding," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, vol. 1. International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 383–390.

[15] H. B. Ammar, D. C. Mocanu, M. E. Taylor, K. Driessens, K. Tuyls, and G. Weiss, "Automatically mapped transfer between reinforcement learning tasks via three-way restricted Boltzmann machines," in *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2013, pp. 449–464.

[16] Q. Cheng, X. Wang, and L. Shen, "An autonomous inter-task mapping learning method via artificial neural network for transfer learning," in *Proceedings of the IEEE International Conference on Robotics and Biomimetics*, 2017, pp. 768–773.

[17] A. Gupta, C. Devin, Y. Liu, P. Abbeel, and S. Levine, "Learning invariant feature spaces to transfer skills with reinforcement learning," *ICLR*, 2017.

[18] D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette, "Evolutionary algorithms for reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 11, pp. 241–276, 1999.

[19] J. Bishop and R. Miikkulainen, "Evolutionary feature evaluation for online reinforcement learning," in *Proceedings of the IEEE Conference on Computational Intelligence in Games*, 2013, pp. 1–8.

[20] B. Hirchoua, I. Mountasser, B. Ouhbi, and B. Frikh, "Evolutionary deep reinforcement learning environment: Transfer learning-based genetic algorithm," in *Proceedings of the 23rd International Conference on Information Integration and Web Intelligence*, 2021, pp. 242–249.

[21] P. C. H. Chu, "A genetic algorithm approach for combinatorial optimisation problems," PhD dissertation, Imperial College of Science, Technology, and Medicine, The Management School, London, United Kingdom, 1997.

[22] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," *arXiv:1606.01540*, 2016.

[23] P. Stone and R. S. Sutton, "Keepaway soccer: A machine learning test bed," in *Robot Soccer World Cup*. Springer, 2001, pp. 214–223.

[24] J. K. Pugh, L. B. Soros, and K. O. Stanley, "Quality diversity: A new frontier for evolutionary computation," *Frontiers in Robotics and AI*, p. 40, 2016.