# Common Subspace Transfer for Reinforcement Learning Tasks

Haitham Bou Ammar*
Institute of Applied Research
Ravensburg-Weingarten University of Applied Sciences, Germany
bouammha@hs-weingarten.de

Matthew E. Taylor
Department of Computer Science
Lafayette College, USA
taylorm@lafayette.edu

## ABSTRACT

Agents in reinforcement learning tasks may learn slowly in large or complex tasks — transfer learning is one technique to speed up learning by providing an informative prior. How to best enable transfer between tasks with different state representations and/or actions is currently an open question. This paper introduces the concept of a *common task subspace*, which is used to autonomously learn how two tasks are related. Experiments in two different nonlinear domains empirically show that a *learned inter-state mapping* can successfully be used by fitted value iteration, to (1) improving the performance of a policy learned with a fixed number of samples, and (2) reducing the time required to converge to a (near-) optimal policy with unlimited samples.

## Categories and Subject Descriptors

I.1.2 [**Algorithms**]: Design and Analysis

## General Terms

Transfer Learning

## Keywords

Transfer Learning, Reinforcement Learning, Common Task-Subspace, Inter-State mapping

## 1. INTRODUCTION

Reinforcement learning [9] (RL) is a popular framework that allows agents to learn how to solve complex sequential-action tasks with minimal feedback. Unfortunately, amount of experience or time required for an RL agent to learn a high-quality policy may grow exponentially with the number of dimensions in the input (state) or output (action) space. Transfer learning [10] (TL) attempts to decrease the amount of time or data required for learning a complex (target) task by providing an informative prior, learned on a simpler (source) task.

At a high level, there are two types of algorithms for TL in RL tasks. The first broad category of algorithms transfer high-level knowledge, such as partial policies, rules, advice, or important features for learning. The second is to transfer low-level knowledge, such as action-value functions or individual state transition data. Our approach deals with the transfer of suggested state/action pairs between different, but related, tasks.

As discussed later in Section 3.4, the source task can potentially differ from the target task in many ways. If the tasks have different representations of state or action spaces, some type of mapping between the tasks is required. While there have been a number of successes in using such a mapping, it typically is hand-coded, and may require substantial human knowledge [13, 10]. This paper introduces a novel construct, a *common task subspace*, and shows that 1) an inter-state mapping can be learned, provided such a subspace through task state transition mappings, and 2) this inter-state mapping can significantly improve learning by transferring state/action data from one task to another based on the similarity of transitions in both tasks.

*The author is also affiliated with the Instituite of Neural Information Processing at the Ulm University, Germany.

This paper provides a proof-of-concept for our method, using fitted value iteration with locally weighted regression in two experiments. The first experiment shows successful transfer from a single mass system into a double mass system. The second experiment uses a policy learned on the simple inverted pendulum task to improve learning on the cartpole swing-up problem. Our results show:

1. an inter-state mapping can be learned from data collected in the source and target tasks;
2. this inter-state mapping can effectively transfer information from a source task to a target task, even if the state representations and actions differ;
3. an agent that uses transferred information can learn a higher quality policy in the target task, relative to not using this information, when keeping the number of samples in the target task fixed; and
4. an agent using information transferred from a source task can learn an optimal policy faster in the target task, relative to not using this information, when it has access to an unlimited number of target task samples.

The rest of the paper proceeds as follows. Related work is presented next, in Section 2. Background information is presented in Section 3. Section 4 describes how an inter-state mapping can be learned between two tasks by leveraging a distance-minimization algorithm. In Section 5, we show how the learned mapping can be used to transfer information between a source task and target task. Experiments in Section 6 evaluate the entire system on two pairs of tasks. Section 7 concludes with a discussion of future work.

## 2. RELATED WORK

There has been a significant amount of work done in recent years on transfer learning in RL domains [10]. This section outlines the most related work (summarized in three classes) and contrast it with this paper.

The first class of papers, providing motivation for this work, focus on using a hand-coded mapping between tasks with different state variables and actions. For instance, [13] transfers advice, and [11] transfers Q-values — both methods assume that a mapping between the state and action variables in the two tasks has been provided. Another approach is to frame different tasks as having a shared *agent space* [5], so that no mapping is explicitly needed, but this requires the agent acting in both tasks to share the same actions and the human must map new sensors back into the agent space. The primary contrast with these authors' work and ours is that we are interested in *learning* a mapping between states and actions in pairs of tasks, rather than assuming that it is provided or unnecessary.

The second approach is to assume that a mapping between tasks is not known, but that a high-level analysis can discover this mapping. For instance, [7] assume that a *quantitative dynamic Bayes network* has been provided for each task. Their work uses a graph mapping technique to efficiently find a mapping between tasks. Other work [6] analyzes full information games, and shows that information can be transferred between games by analyzing rule graphs constructed from the (known) transition function. In both cases, no data needs to be sampled from the environment, as the transition function can be analyzed (in terms of a network or rule graph, respectively). Our methods, rather than relying on analysis of the Markov Decision Processes (MDPs), instead are data-driven methods, using supervised learning techniques to find an accurate mapping.

A third approach involves learning a mapping between tasks using data gathered while agents interact with the environment. [8] supply an agent with a series of possible state transformations and a description of how actions are related in a pair of tasks. Over time the agent can learn the correct mapping by balancing exploration of the different transformations and exploiting the transformation thought to be best. In contrast to this method, our framework does not assume that the agent knows how the actions are related between the two tasks, nor does it rely on finding the correct mapping via exploration. Other work [12] learns both the state and action mapping simultaneously by gathering data in both the source task and the target task. They then use a classifier to find the most consistent mapping. However, this approach is computationally expensive and scales exponentially with the number of state variables and actions in the two tasks. In contrast, our approach will scale much better with higher dimensional tasks, assuming that a smaller task specific subspace can be found.

Finally, unlike all other existing methods (to the best of our knowledge), we assume differences among all the variables of MDPs describing the tasks and focus on learning an inter-state mapping, rather than a state-variable mapping. Our framework can also map different actions depending on the state. For instance, it could be that in some parts of the target task, action $a_{1,target}$ in the target task is most similar to action $a_{1,source}$ in the source task, while in other parts of the target task, action $a_{1,target}$ is most similar to action $a_{2,source}$. Since our framework relies on state transition similarities in both the target and source task, then it allows such a flexibility for the action choices in certain regions of the state space, while other existing algorithms do not.

## 3. BACKGROUND

This section provides the reader with a short background in reinforcement learning, transfer learning, locally weighted regression for function approximation and the learning methods used in this paper.

### 3.1 Reinforcement Learning

In an RL problem, an agent must decide how to sequentially select actions in order to maximize its long term expected reward [9]. Such problems are typically formalized as Markov decision processes (MDPs). An MDP is defined by $\langle S, A, P, R, \gamma \rangle$, where $S$ is the (potentially infinite) set of states, $A$ is the set of all possible actions that the agent may execute, $P : S \times A \to S$ is a state transition probability function describing the transition dynamics, $R : S \to \mathbb{R}$ is the reward function measuring the performance of the agent, and $\gamma \in [0, 1)$ is the discount factor. A policy $\pi : S \to A$ is defined as a mapping from a state to an action. The goal of an RL agent is to improve its policy, potentially reaching the optimal policy $\pi^*$ that maximizes the discounted total long term reward:

$$V^*(s) = \max_{\pi} E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | s = s_0, \pi\right]$$

### 3.2 Fitted Value Iteration

When operating in a continuous state space, the value function cannot be enumerated in a table [3]. Instead, some sort of function approximation must be used. The *fitted value iteration* (FVI) algorithm [3], as shown in Algorithm 1, is one approach to the problem of approximating a continuous function. The key idea of FVI is to approximate the value function after sampling a finite number of states using a parametric or nonparametric combination of some feature vector of the states. The value function, estimating the long-term value of a state, is

$$V(s) = \Psi^T \Phi(s) \tag{1}$$

where $\Psi^T$ is a vector of parameters to be fitted and $\Phi(s)$ is an appropriate feature vector mapping of the states. For each state in the finite sample and for each action $a \in A$, Algorithm 1 determines a quantity $y^{(i)}$ which is an approximation of the value function. Then it solves a linear regression problem to fit the $\Psi$ values making $V(s)$ as close as possible to $y^{(i)}$.[1]

### 3.3 Locally Weighted Regression

*Locally weighted regression* [1] (LWR) is a supervised learning algorithm used in function approximation where local models are fitted near query points. LWR is a *lazy* or *memory-based learning* method, where generalization is delayed until a query is made. In LWR, a weighted least squares criteria is used to fit local models. Such an approximation method is suited to our problem because state-action pairs are collected offline, as described in Section 5.1.

### 3.4 Transfer Learning in RL Tasks

In transfer learning, there typically exists a source and a target task, where the goal is to increase the performance and to reduce the learning times in the target task agent [10]. This is done by allowing an agent in a target task to reuse knowledge and behaviors acquired by an agent in one or more source tasks. In our transfer learning framework, we assume that there are two different but related tasks: a source and a target. We define both tasks as MDPs, where information is transferred from the source task (*MDP*$_1$) into the target task (*MDP*$_2$).

*MDP*$_1$ is defined by the tuple $(S_1, A_1, P_1, R_1, \gamma_1)$, while *MDP*$_2$ by $(S_2, A_2, P_2, R_2, \gamma_2)$, where $S_i \in R^{d_i}$, $A_i \in R^{q_i}$, $P_i$, $R_i$ : $S_i \to R$ and $\gamma_i$ for $i \in \{1, 2\}$ represent the state space, action space, transition probability and the discount factor of each of the

[1]In case of stochastic MDPs then $q(a)$ on line 7 is found by averaging over a number of successor states.

**Algorithm 1** Fitted Value Iteration for deterministic MDPs

```
1: Randomly sample $m$ states from the MDP
2: $\Psi \leftarrow 0$
3: $n \leftarrow$ the number of available actions in $A$
4: repeat
5:   for $i = 1 \rightarrow m$ do
6:     for all $a \in A$ do
7:       $q(a) \leftarrow R(s^{(i)}) + \gamma V(s^{(j)\prime})$
8:     $y^{(i)} \leftarrow \max q(a)$
9:   $\Psi \leftarrow \arg\min_{\Psi} \sum_{i=1}^{m} (y^{(i)} - \Psi^T \Phi(s^{(i)}))^2$
10: until $\Psi$ Converges
```

MDP respectively. In this paper, we assume that the source task can be easily learned and that an optimal policy, $\pi_1^*$, has already been found.[2] We note that our methods do not require similarities between any given pairs of source task / target task constituents. In other words, the source and target task can have differences in state spaces, action spaces, transition probabilities, reward functions, and/or discount factors. In Section 6, we show positive results when transferring between tasks that have different state spaces, action spaces, transition probabilities, and reward functions.

### 3.5 Inter-State Mapping

In order to enable transfer between tasks with different state and action spaces, some type of inter-state mapping, $\chi$, must be used.

The inter-state mapping, $\chi : S_2 \rightarrow S_1$, is a function mapping the state space of *MDP*$_2$ into *MDP*$_1$. It describes the relationship between the state space representations among the different but related MDPs by finding a label $s_1 \in S_1$, to an input $s_2 \in S_2$. For attaining such an inter-State mapping a supervised learning algorithm should be used. The major problem for any function approximator is the missing correspondence between the inputs, being states in $S_2$ to the outputs being states in $S_1$. We approach this problem by finding this correspondence between the inputs and the labels in a *common task-subspace* as described in Section 4.

Such a function is essential to our transfer framework since it is used to transfer knowledge from a source task agent into a target task agent, which acts in a different state space, with a different state representation (as described in Section 5.1).

## 4. LEARNING AN INTER-STATE MAPPING

At a high-level, our transfer framework can be decomposed into three major phases. In the first phase, the function $\chi$ is learned, mapping the states from *MDP*$_2$ into *MDP*$_1$. As discussed in this section, $\chi$ is learned by collecting transitions from the source task and target task and identifying correspondences. The second phase finds an initial policy for task two, $\pi_{tr}$ in *MDP*$_2$, by identifying actions in the target task that are most similar to actions selected in the source task by $\pi_1^*$ (see Section 5.1). The third phase uses samples gathered by $\pi_{tr}$ as an initialization for fitted value iteration, rather than using randomly selected samples, finding an optimal policy $\pi_2{}^*$ of *MDP*$_2$ (see Section 5.2).

We define a *common task subspace*, $S_c$, as a subspace that describes shared characteristics between the tasks *MDP*$_1$ and *MDP*$_2$. Generally, $S_c$ has a lower dimensionality than $S_1$ or $S_2$ and is determined by common state semantics shared between the two tasks. This subspace is described via the control problem's definition or is user defined. In many cases, manually defining such a common task subspace is relatively easy. In the case of control problems, the subspace construction can be influenced by the particular goal or goals an agent must achieve in a task. As an illustration, consider the problem of transfer between agents with two different robotic arms, each of which has acts in a different dimensionality space (i.e., has a different description of state because of different sensors and or degrees of freedom). In this case, $S_c$ can be defined as the position and orientation of the end effector in both robots. Thus, even with such a nonlinear continuous MDPs setting, attaining a common task space requires less effort than trying to manually encode the action and state variables mappings.

$S_c$ is used to determine the correspondence between state successor state pairs of *MDP*$_1$ and *MDP*$_2$, which in turn will generate data used to approximate $\chi$. Given that the two tasks are related through some common task subspace $S_c \in \mathbb{R}^{d_c}$, we proceed by learning a function $\chi : S_2 \rightarrow S_1$, mapping the two state spaces of *MDP*$_1$ and *MDP*$_2$ together. As discussed in Section 5.1, $\chi$ alone is capable of transferring policies from *MDP*$_1$ to *MDP*$_2$ by effectively finding a good prior for the agent in *MDP*$_2$.

We now explain how $\chi$ is learned. We take as input (1) $n_1$ state successor state patterns of the $d_1$ dimensional state space $S_1$, $\langle s_1, s_1' \rangle$ (gathered from interactions with the source task), (2) $n_2$ state successor state patterns of the $d_2$ dimensional state space $S_2$, $\langle s_2, s_2' \rangle$ (gathered from interactions with the target task), and (3) a common task subspace $S_c$ with dimensions $d_c \leq \min\{d_1, d_2\}$. Algorithm 2 proceeds by projecting each of the above patterns into $S_c$, attaining $n_1$ patterns of the form $\langle s_{c,1}^{(i)}, s_{c,1}^{\prime(i)} \rangle$, were the subscript $\{c, 1\}$ denotes mapping states from $S_1$ into states in $S_c$, for $i = \{1, 2, \ldots, n_1\}$, corresponding to the projected $S_1$ states (line 2 of Algorithm 2). Additionally, $n_2$ patterns of $\langle s_{c,2}^{(j)}, s_{c,2}^{\prime(j)} \rangle$ are found on line 4 of Algorithm 2, where the subscript $\{c, 2\}$ represents the notion of state space $S_2$ states in $S_c$ and $j = \{1, 2, \ldots, n_2\}$, corresponding to the projected $S_2$ states. The algorithm next calculates a minimum distance on the $n_1$ and $n_2$ patterns (lines 6–8). Once a correspondence between the projected states in $S_c$ has been found, full states rather than subspace states are required to train $\chi$. These are found by trying all the combinations in $S_1$ and $S_2$, lines 10–12, generating the recommended $s_{c,1}$ and $s_{c,2}$ (further discussed in Section 4.2). The algorithm collects these combinations (line 12) so that $\chi$ represents a best fit mapping between $S_2$ and $S_1$ via $S_c$.

### 4.1 Problem: Mapping Unrelated States

At this stage two potential problems arise. The first is that it is possible that states in $S_2$ are mapped into states in $S_1$, even when they are not related. This is a common problem in transfer learning (related to the problem of *negative transfer* [10]) which we cannot solve, but work to avoid by considering the distance between successor states.

Consider patterns in the target task, $\langle s_2, s_2' \rangle$, and a pattern in the source task, $\langle s_1, s_1' \rangle$. Using Algorithm 2, lines 2 and 4, we find that $f_2$ and $f_1$ maps each of the successor states into the common sub-space as $\langle s_{c,2}, s_{c,2}' \rangle$ and $\langle s_{c,1}, s_{c,1}' \rangle$ respectively. If the distance d, as measured by $||\langle s_{c,1}, s_{c,1}' \rangle, \langle s_{c,2}, s_{c,2}' \rangle||_2$, is greater than some threshold parameter (line 9), it suggests this mapping is suspect because the initial state successor state pair, $\langle s_2, s_2' \rangle$, has a poor correspondence with the source task pattern, potentially harming the agent's performance in *MDP*$_2$.[3] This state may not be the best choice for a prior in the target task — only states with small

[2]The framework is not limiting to having an optimal policy — we believe suboptimal policies could also be used successfully — but we focus on optimal policies for clarity of exposition.

[3]Even if the two tasks are closely related this could occur due to a large difference in the action spaces of the two tasks.

**Algorithm 2** Learn an Inter-State Mapping

```
Require: $n_1$ random samples of $\langle s_1^{(i)}, s_1'^{(i)}\rangle_{i=1}^{n_1}$; $n_2$ random samples of $\langle s_2^{(k)}, s_2'^{(k)}\rangle_{j=1}^{n_2}$; $f_1$ and $f_2$ representing the functions projecting $S_1$ and $S_2$ into $S_c$, respectively; and threshold $\beta_1$
1: for $i = 1 \to n_1$ do
2:   $\langle s_{c,1}^{(i)}, s_{c,1}'^{(i)}\rangle \leftarrow f_1\langle s_1^{(i)}, s_1'^{(i)}\rangle$
3: for $j = 1 \to n_2$ do
4:   $\langle s_{c,2}^{(j)}, s_{c,2}'^{(j)}\rangle \leftarrow f_2\langle s_2^{(j)}, s_2'^{(j)}\rangle$
5: for $k = 1 \to n_2$ do
6:   for $l = 1 \to n_1$ do
7:     $d^{(l)} \leftarrow ||\langle s_{c,1}^{(l)}, s_{c,1}'^{(l)}\rangle - \langle s_{c,2}^{(k)}, s_{c,2}'^{(k)}\rangle||_2$
8:   Calculate $l^* \leftarrow \arg\min_l d^{(l)}$
9:   if $d_{best}^{(l^*)} \leq \beta_1$ then
10:    $s_{c,1}^{(k)} \leftarrow$ all combinations of $s_1$
11:    $s_{c,2}^{(l^*)} \leftarrow$ the combinations of $s_2$
12:    Collect all combinations of the latter $s_2$ and $s_1$ as inputs and outputs, respectively, to approximate $\chi$
13:  else
14:    Do Nothing {ignore current sample}
15: Approximate $\chi$
```

distances are used as inputs and outputs for the supervised learning algorithm.

## 4.2 Problem: Non-injective Mapping

The second potential problem is that the function $\chi$ must map all state variables from the target task into the source task. However, the correspondence between the inputs, states in $S_2$, and the outputs, states in $S_1$, was found in the common state subspace $S_c$. The projection functions, $f_1$ and $f_2$, from $S_1$ and $S_2$ respectively, are not-injective. Thus, there may be a problem when attempting to fully recover the initial data points in $S_1$ and $S_2$, corresponding to $s_{c,1}$ and $s_{c,2}$, which is critical when approximating $\chi$.

We approach this problem by verifying all possible states in $s_1 \in S_1$ and $s_2 \in S_2$ corresponding to the intended $s_{c,1}$ and $s_{c,2}$ respectively. We then consider all combinations of the initial states, on line 12, that were mapped together using Algorithm 2, as inputs and outputs. By that, the authors have avoided the need for an inverse mapping $f_1^{-1}$ and $f_2^{-1}$ to recover the original states in $S_1$ and $S_2$. Once the correspondence between the patterns of $S_1$ and $S_2$ has been determined, a supervised learning scheme attains $\chi$. LWR was used (line 15 of Algorithm 2) to approximate $\chi$, which is used in turn to determine the transferred policy, $\pi_{tr}$, as described in the following section.

## 5. POLICY TRANSFER AND RL IMPROVEMENT

To transfer between agents with differences in the action spaces some type of a mapping representing the relations between the allowed actions of the source and target agent should be conducted. In finding a mapping of the action spaces between the tasks, there exists a major problem. The problem relates to the difference in dimensions between the two action spaces. Solving this problem could not be approached as done for the state space case in Section 3.5, since it is not trivial at all to determine some common action space shared between the tasks to be projected to so to find the inputs and labels which in turn would be used to map the action spaces together[4].

**Algorithm 3** Collect State-action Pairs

```
Require: $m$ random initial $s_2$ states, optimal policy of the first system $\pi_1^*$, probability transition functions of the two systems $P_1(s_1, a_1)$ and $P_2(s_2, a_2)$, the action space of system two $A_2$, and distance threshold $\beta_2$
1: Set $q_2$ to be the size of $A_2$
2: for $i = 1 \to m$ do
3:   ${s_1}^{(i)} \leftarrow \chi({s_2}^{(i)})$
4:   $a_1^{(i)} \leftarrow \pi_1^*(s_1^{(i)})$
5:   Attain $s_1'^{(i)} \sim P_1({s_1}^{(i)}, {a_1}^{(i)})$ sampled according to the state transition probability $P_1$
6:   for $k = 1 \to q_2$ do
7:     Attain $s_2'^{(k)} \sim P_2({s_2}^{(i)}, {a_2}^{(k)})$ sampled according to the state transition probability $P_2$
8:     Attain the corresponding $s_{1,c}'^{(k)} \leftarrow \chi(s_2'^{(k)})$ using the inter-state mapping $\chi$
9:     $d^{(k)} \leftarrow ||s_1'^{(i)} - s_{1,c}'^{(k)}||_2$
10:  $d_{best}^{(i)} \leftarrow \min_k(d^{(k)})$
11:  $j \leftarrow \arg\min_k d^{(k)}$
12:  if $d_{best}^{(i)} \leq \beta_2$ then
13:    Collect the following pattern $(s_2^{(i)}, a_2^{(j)})$ as one training pattern to approximate $\pi_2$
14:  else
15:    Do Nothing {Ignore this sample}
16: Using collected patterns, approximate $\pi_{tr}$
```

Rather than approaching this problem explicitly and conducting a mapping between the action spaces of the tasks, we perform an implicit mapping using the inter-state mapping learned in Section 3.5.

The inter-state mapping, $\chi$, will enable transfer from *MDP*$_1$ to *MDP*$_2$. This transfer is based on a similarity measure between state successor states in both MDPs, in the sense that only state transitions that relatively have acceptable distance measures are taken into account. Then, the action producing such a successor state in *MDP*$_2$ is held as the best action. This section will further detail the above scheme and explain how $\chi$ is used to conduct a policy transfer between the two MDPs.

## 5.1 Policy Transfer Scheme

The inter-state mapping, as learned in the previous section, is capable of providing the agent in the target task with an informative prior. Finding the transferred policy, $\pi_{tr}$, is done in two phases. First, state-action pairs are collected in the source task, according to $\pi_1^*$ (see Algorithm 3). Second, $\pi_{tr}$ is constructed from the collected samples, and the learned inter-state mapping.

Algorithm 3 needs to be able to generate successor states for both MDPs, lines 5–7. Thus, it is not necessary for Algorithm 3 to have access to a transition model or simulator, where agents in both tasks can generate next states by taking actions.

Algorithm 3 finds an action, $a_2 \in A_2$, for a state $s_2 \in S_2$, by using the inter-state mapping, $\chi$, and a user-defined threshold, $\beta_2$. Using $\chi$, the algorithm maps each of the $m$ random states, $s_2^{(1)}$–$s_2^{(m)}$, to corresponding states, $s_1^{(1)}$–$s_1^{(m)}$. It then selects on action, $a_1$, for a state in $S_1$, according to the optimal policy of *MDP*$_1$,

[4]This is in addition to the problem of determining an inverse mapping for $\chi$, since we need to approximate a starting policy in the target task.

**Algorithm 4** Fitted Value Iteration Algorithm + Transfer

```
1: Starting from random initial states, sample $f$ states according to $\pi_{tr}$
2: $\Psi \leftarrow 0$
3: $n_2 \leftarrow$ the size of the action space $A_2$
4: repeat
5:     for $i = 1 \rightarrow f$ do
6:         for all $a_2 \in A_2$ do
7:             $q(a_2) \leftarrow R(s^{(i)}) + \gamma V(s^{(j)'})$
8:         $y^{(i)} \leftarrow \max_{a_2 \in A_2} q(a_2)$
9:     $\Psi \leftarrow \arg\min_\Psi \sum_{i=1}^{f} \left(y^{(i)} - \Psi^T \Phi(s^{(i)})\right)^2$
10:    Greedily sample new $f$ states according to the fitted $\Psi$ values representing $\pi_{fit} = \arg\max_a E_{s' \sim P_{sa}}[\Psi^T \Phi(s')]$
11: until $\Psi$ converges
12: Represent $\pi_2^* = \arg\max_a E_{s' \sim P_{sa}}[\Psi^T \Phi(s')]$
```

and transients into the optimal $s'_1$ state according to the probability transition function $P_1(s_1, a_1)$.

The algorithm examines all possible actions in $A_2$ from the given initial state $s_2^{(i)}$ to transient to $q_2$ different subsequent states $s'_2$ (see line 6 – 7 of Algorithm 3). Then for each $s'_2$, $\chi$ is used again to find the corresponding $s'_1$ denoted by $s'_{1,c}$ in the algorithm, line 8. At this stage, a minimum distance search between the attained $s'_{1,c}$ and the one recommended by $\pi_1^*$ is conducted. If the distance is below the user defined threshold $\beta_2$ then the action $a_2$ corresponding to the minimum distance is chosen to be the best action for that random initial state $s_2$. This sequence is repeated for the $m$ different random initial states of $S_2$, resulting in a *data set of state-action pairs in the target task*, guided by $\pi_1^*$.

This data set is used to approximate $\pi_{tr}$, done via LWR in our experiments, and this policy will be used as a starting policy by the target task agent.

## 5.2 Improving the Transferred Policy

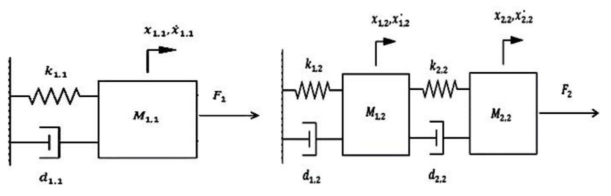
The policy $\pi_{tr}$ serves as an initial policy for the *MDP*$_2$ agent — this section describes how the policy is improved via FVI, using an initial trajectory produced by $\pi_{tr}$.

We used a minor variant of FVI, where the value function is repeatedly approximated after fitting the $\Psi$ values. Starting from a small number of initial states, $f$, sampled through $\pi_{tr}$, we attempt to find an optimal policy $\pi_2^*$, by iteratively re-sampling using the fitted $\Psi$ values as needed.

Algorithm 4 works to find optimal values for the parameters to fit the value function (Equation 1) on a set number of samples, which were sampled using $\pi_{tr}$. Then, after each iteration of the repeat loop, Algorithm 4 samples a new set of states according to current policy represented by $\pi_{fit}$. The sampling / value fitting process is repeated until convergence, attaining an optimal policy. The difference between Algorithm 4 to the one described in Section 3.2, is that the initial samples are not gathered according to a random policy, but by following $\pi_{tr}$. Assuming that $\pi_{tr}$ is a good prior, this procedure will better focus exploration of the policy space.

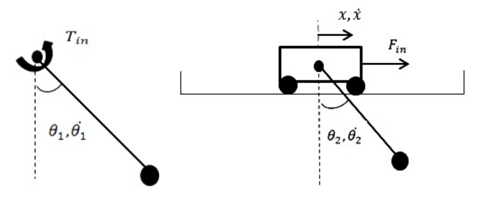
## 6. EXPERIMENTS

As a proof of concept, our algorithms were tested on two different systems. The first was the transfer from a single mass spring damper system to a double mass spring damper system, as shown in Figure 1. The second experiment transferred between the inverted pendulum task to the cartpole swing-up task [2] (see Figure 2). The following two sub-sections will discuss the details of the experiments and their results.



(a) Simple Mass System (b) Double Mass System

**Figure 1: The first experiment uses a policy for the single mass spring damper system in (a) to speed up learning a policy for the double mass spring damper system in (b).**



(a) Simple Pendulum (b) Cartpole swing-up

**Figure 2: The second experiment uses a policy for the inverted pendulum in (a) to speed up learning a policy for the benchmark cartpole swing-up task in (b).**

The values of the discount factor $\gamma$, used in Algorithms 1 and 4 were fixed to 0.8 while those of $\beta_1$ and $\beta_2$, used in algorithms 2 and 3, were fixed at 0.9 and 1.5, respectively. In fact, we found that varying the values of $\beta_1$ and $\beta_2$ did not significantly affect the performance of the algorithms, suggesting that our algorithms are robust to changes in these parameters.[5]

## 6.1 Single to Double Mass

For our first experiment, we transferred a policy between the systems shown in Figure 1. Detailed descriptions of the tasks' dynamics can be found elsewhere [4]. $S_1$ is described by the $\{x_{1,1}, \dot{x}_{1,1}\}$ variables, representing the position and the velocity of the mass $M_{1,1}$. $S_2 = \{x_{1,2}, \dot{x}_{1,2}, x_{2,2}, \dot{x}_{2,2}\}$, representing the position and the velocity of $M_{1,2}$ and $M_{2,2}$.

A reward of $+1$ is given to the agent of system one if the position of the mass $M_{1,1}$ is 1 and $-1$ otherwise. On the other hand, a reward of $+10$ is given to the agent of system two if the position and the velocity of the mass $M_{1,2}$ is 1 and 0 respectively, and otherwise a reward of $-10$ is given. The action spaces of the two systems are $A_1 = \{-15, 0, 15\}$ and $A_2 = \{-15, -10, 0, +10, -15\}$, describing the force of the controller in Newtons. The agent's goal is to bring the mass of system two, $M_{1,2}$, to the state $s_2 = \{1, 0\}$, which corresponds to a position of 1 ($x_{1,2} = 1$) and a velocity of zero ($\dot{x}_{1,2} = 0$). In our transfer learning setting, the agent relies on an initial policy delivered from the controller of the system *MDP*$_1$ and improves on it. In the source task, FVI found a policy to bring the mass $M_{1,1}$ to the $s_1 = \{1, 0\}$ goal state.

### *6.1.1 Common Task Subspace*

[5]We believe that carefully setting $\beta_1$ and $\beta_2$ may only be necessary when the source and target tasks are very dissimilar.

In both systems the control goal is to settle the first mass so that it reaches location $x = 1$ with zero velocity. Thus, the common task subspace $S_c$ is described via the variables $x$ and $\dot{x}$ for mass #1 in both systems.

### 6.1.2 Source Task: Single Mass System

The FVI algorithm was used to learn an optimal policy, $\pi_1^*$, for the first mass system. A parametric representation of the value function was used:

$$V(s) = \Psi^T \Phi(s)$$

$$V(s) = \begin{pmatrix} \psi_1 & \psi_2 & \psi_3 & \psi_4 & \psi_5 \end{pmatrix} \begin{pmatrix} {x_{1,1}}^2 \\ x_{1,1} \\ \dot{x}_{1,1}^2 \\ \dot{x}_{1,1} \\ 1 \end{pmatrix}$$

The second variant of Algorithm 1, described via Algorithm 4 but starting from random samples (source task), was able to converge to the optimal parametric values approximating the value function on a single core of a dual core 2.2 GHz processor in about 18 minutes, after starting with 5000 random initial samples. The resulting controller, represented as values in $\Psi$, was able, in 0.3 seconds, to control the first mass system in its intended final state: $s_1 = \{1, 0\}$.

### 6.1.3 Target Task: Double Mass System

To test the efficacy of our learned $\chi$ by Algorithm 2 and transfer method using Algorithms 3 and 4, we varied the values for $n_1$ and $n_2$ from 1000–8000, which corresponds to the number of samples used in the target task.[6] Algorithm 1 was run with these different sets of samples, which were in turn used to generate policies for the target task. The performance of these policies in the target task, after convergence, are shown in Figure 3, and are compared to using random initial samples (i.e., no transfer).

The results in Figure 3 show that FVI performs better when initialized with a small number of states sampled from $\pi_{tr}$ than when the states are generated by a random policy. Further, results confirm that as the number of samples increase, both transfer and non-transfer learning methods converge to the (same) optimal policy.

**Conclusion 1**: $\pi_{tr}$, which uses the learned $\chi$, allows an agent to achieve a higher performance with a fixed number of sampled target task states compared to a random scheme.

Finally, Algorithm 4 was used to attain the optimal policy $\pi_2^*$ when supplied with 7000 initial points, where the points were sampled randomly and from $\pi_{tr}$. The convergence time to attain an optimal policy starting from the initial states generated through $\pi_{tr}$ was approximately 4.5 times less than that starting from randomly sampled initial states.

**Conclusion 2**: $\pi_{tr}$ allows an agent to converge to an optimal policy faster by intelligently sampling the initial states for FVI that are improved on.

## 6.2 Inverted Pendulum to the Cartpole Swing-up

For the second experiment, we transfered between the systems shown in Figure 2. A detailed description of the task's dynamics can be found elsewhere [2]. $S_1$ is described by the $\theta_1$ and $\dot{\theta}_1$ variables representing the angle and angular speed of the inverted

[6]This corresponds to roughly 10–175 states ignored in Algorithm 2, line 14.

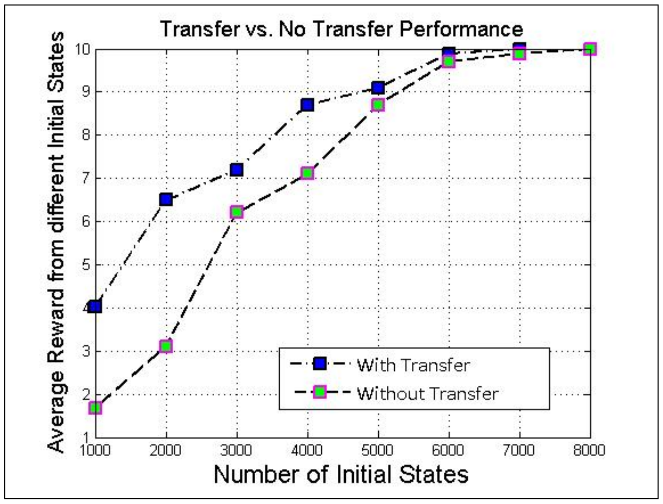


**Figure 3: This graph compares the performance of converged policies on the double mass system, as measured over $1000$ independent samples of random start states in the target task measured over independent $500$ trials. The $x$-axis shows the number of target task states used by FVI and the $y$-axis shows the average reward after FVI has converged (without resampling the states).**

pendulum respectively. $S_2$ is described by $\theta_2$, $\dot{\theta}_2$, $x$, and $\dot{x}$ representing the angle, angular speed, position, and velocity of the cartpole, respectively.

The reward of system one (inverted pendulum) was defined as $R_{sys_1} = \cos(\theta_1)$ while that of system two (cartpole swing up) was $R_{sys_2} = 10\cos(\theta_2)$. The action spaces of the two systems are $A_1 = \{-15, -1, 0, 1, 15\}$ and $A_2 = \{-10, 10\}$, describing the allowed torques, in Newton meters, and forces, in Newtons, respectively. The cart is able to move between $-2.5 \leq x \leq 2.5$. The agent's goal in the source task is to bring the pendulum of system two to state $s_2 = \{0, 0\}$, which corresponds to an angle of 0 ($\theta_2 = 0$) and an angular velocity ($\dot{\theta}_2 = 0$). In our transfer learning setting, the agent relies on an initial policy delivered from the controller of the first system and improves on it. In the source task, FVI found a policy to bring the pendulum to the state $s_1 = \{0, 0\}$.

### 6.2.1 Common Task Subspace

In both systems the control goal is to settle the pendulums in the $\{0, 0\}$ upright state corresponding to an angle of zero and an angular velocity of zero. Thus, the common task subspace $S_c$ is described via the variables $\theta$ and $\dot{\theta}$ of both systems.

### 6.2.2 Source Task: Simple Pendulum

The FVI algorithm was used to learn an optimal policy, $\pi_1^*$. As shown in Equation 1, a parametric representation of the value function was used:

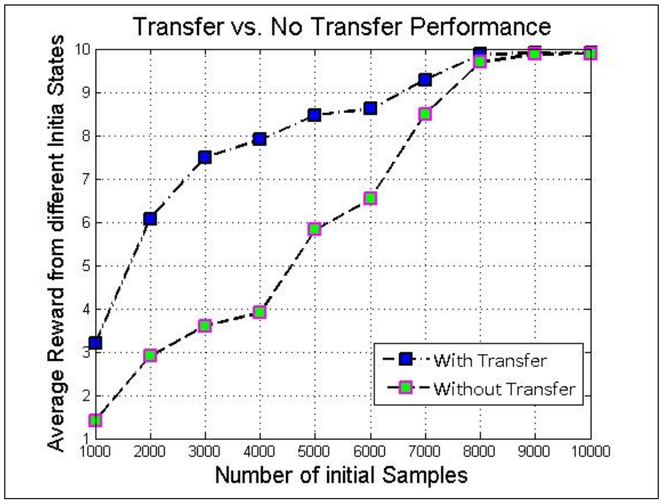


**Figure 4: This figure compares the performance of the cartpole swing-up task, measured by the averaged reward, vs. different numbers of initial starting states. Starting states can be sampled via the transfer policy (from the inverted pendulum task) or randomly.**

$$V(s) = \Psi^T \Phi(s)$$

$$V(s) = \begin{pmatrix} \psi_1 \; \psi_2 \; \psi_3 \; \psi_4 \; \psi_5 \end{pmatrix} \begin{pmatrix} \theta_1^2 \\ \theta_1 \\ \dot{\theta}_1^{\,2} \\ \dot{\theta}_1 \\ 1 \end{pmatrix}$$

The second variant of Algorithm 1 described via Algorithm 4, was able to converge to the optimal parametric values approximating the value function when on a single core of a dual core 2.2 GHz processor in about 23 minutes after starting from 5000 random initial samples. Then the controller was able, in 0.2 sec, to control the inverted pendulum in its intended final state $s_1 = \{0, 0\}$.

### 6.2.3 *Target Task: Cartpole Swing-up*

To test the efficacy of our learned $\chi$ using Algorithm 2 and transfer method using Algorithms 3 and 4, we varied the values for $n_1$ and $n_2$ from 1000 – 10000, which corresponded to the number of samples in the target task.[7] Algorithm 1 was run with these different sets of samples, which were in turn used to generate policies for the target task. The performance of these policies in the target task, after convergence, are shown in Figure 4, and are compared to the random scheme (i.e., no transfer).

The results in Figure 4 show that FVI performs better when initialized with a small number of states sampled from $\pi_{tr}$ than when the states are generated by a random policy. Further, the results confirm that as the number of samples increase, both transfer and non-transfer learning methods converge to the (same) optimal policy.

Finally, Algorithm 4 was used to attain the optimal policy $\pi_2^*$ when supplied with 7000 initial points, where the points were sampled randomly and from $\pi_{tr}$. The convergence time to attain an optimal policy starting from the initial states generated through $\pi_{tr}$ was approximately a factor of 6.3 less than that starting from randomly sampled initial states.

[7]This corresponds to roughly 18 – 250 states ignored in Algorithm 2, line 14.

**Table 1: Experiment Results Summary**

| | Double Mass System | | | |
|---|---|---|---|---|
| | No TL | | With TL | |
| Transitions | Reward | Time | Reward | Time |
| 1000 | 1.7 | 6.5 | **3.9** | **4.5** |
| 5000 | 8.7 | 27 | **9.1** | **9.5** |
| 10000 | 9.9 | 43 | 9.9 | **11.8** |
| | Cartpole Swing-up | | | |
| | No TL | | With TL | |
| Transitions | Reward | Time | Reward | Time |
| 1000 | 1.4 | 10 | **3.1** | **7** |
| 5000 | 6.09 | 32 | **8.4** | **15** |
| 10000 | 9.9 | 160 | 9.9 | **27** |

These results, summarized in Table 1, confirm the conclusions made in Section 6.1.3. The performance, as measured by the final average reward, was higher when using TL than when using randomly selected states. Furthermore, FVI was able to find an optimal policy in fewer minutes, denoted by $T$ in the table, when using TL than when using randomly selected initial states.

## 7. CONCLUSIONS & FUTURE WORK

We have presented a novel transfer learning approach based on the presence of a common subspace relating two tasks together. The overall high level scheme shown in Figure 5 emphasizes that our frame work constitutes of three major phases.

The first is the determination of the inter-state mapping $\chi$, relating the state spaces of the tasks, using a common task subspace, $S_c$, as described in Section 4. It relies on distance measures among state successor state pairs in both task to achieve the goal of finding a correspondence between the state spaces of the two tasks and then conducts a function approximation technique to attain $\chi$.

The second, is the determination of starting policy in the target task, $\pi_{tr}$, based on similarity transition measures between the two related tasks as presented in Section 5.1. This is achieved by mapping state successor states pairs in the target task back to corresponding pairs in the source task and then conducting a search to the most similar transition recommended by the optimal policy of the source task. The action in the target task with the closest similarity to that in the source task accompanied with the intended initial state is collected as in a data set to approximate a good prior in the target task.

Since $\pi_{tr}$ is a good starting prior for the agent in the target task to start from it needed improvement. The last point constitutes the third phase of our framework, as shown in Figure 5, which was conducted using FVI and detailed in Section 5. Here, the states recommended by $\pi_{tr}$ are used as an initial trajectory to start from and improve on.

In our approach, the common subspace was determined manually which was a good trade-off over the determination of the inter-task mapping manually. Such a space is relatively easy to design by a human just from knowing the control problem goal.

Results show that our algorithm was able to surpass ordinary fitted value iteration algorithms by attaining higher reward with fewer

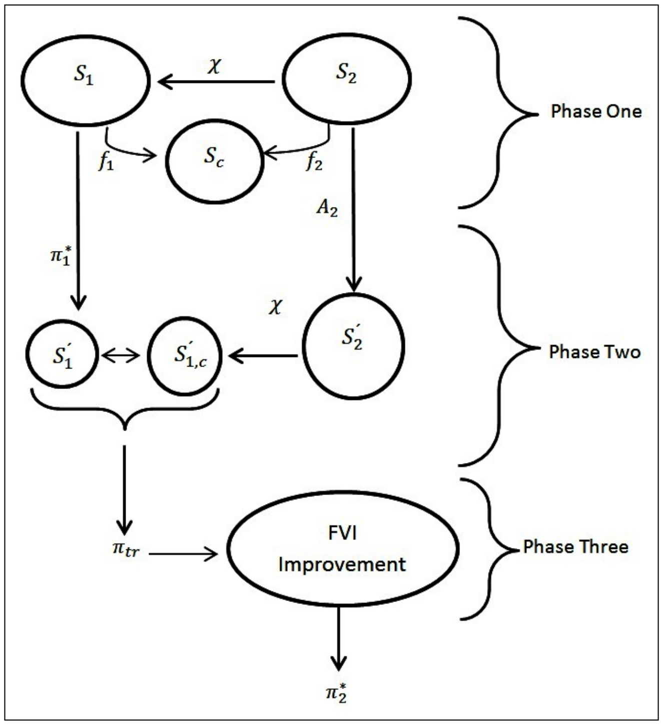


**Figure 5: This figure represents the overall scheme of our framework constituting of three major phases.**

initial states. Additionally, our results showed significant time reductions when attempting to find optimal policies in the target task, relative to the normal FVI algorithms.

Our future work will involve three major goals. The first is to extend our algorithms to operate in stochastic model-free MDP settings. The second is to determine the common subspace automatically in both the action and state spaces. Various ideas could be used to achieve such a goal, one of which could be a dimensionality reduction scheme constrained by the common characteristics shared by the different tasks. The third is to test our transfer method with multiple algorithms including policy iteration, Sarsa($\lambda$) and Q-learning.

## 8. ACKNOWLEDGMENTS

This work was supported by the Collaborative Center of Applied Research on Service Robotics (ZAFH Service Robotics). The authors would like to thank Wolfgang Ertel for the discussions.

## References

Atkeson, Christopher G., Moore, Andrew W., and Schaal, Stefan. Locally weighted learning. *A.I. Rev.*, 11(1-5):11–73, 1997.

Barto, Andrew G., Sutton, Richard S., and Anderson, Charles W. *Neuronlike adaptive elements that can solve difficult learning control problems*, pp. 81–93. IEEE Press, 1990.

Busoniu, Lucian, Babuska, Robert, Schutter, Bart De, and Ernst, Damien. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2010.

Charles M. Close, Dean K. Fredrick and Newel, Jonathan C. *Modeling and Analysis of Dynamic Systems*. John Wiley & Sons, Inc., Third Avenue, NY, USA, 3rd edition, 2002.

Konidaris, George and Barto, Andrew. Autonomous shaping: Knowledge transfer in reinforcement learning. In *ICML*, 2006.

Kuhlmann, Gregory and Stone, Peter. Graph-based domain mapping for transfer learning in general games. In *ECML*, 2007.

Liu, Yaxin and Stone, Peter. Value-function-based transfer for reinforcement learning using structure mapping. In *AAAI*, July 2006.

Soni, Vishal and Singh, Satinder. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *AAAI*, July 2006.

Sutton, Richard S. and Barto, Andrew G. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 9(5): 1054–1054, 1998.

Taylor, Matthew E. and Stone, Peter. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

Taylor, Matthew E. Stone, Peter, and Liu, Yaxin. Transfer learning via inter-task mappings for temporal difference learning. *J. of Machine Learning Research*, 8(1):2125–2167, 2007.

Taylor, Matthew E., Jong, Nicholas K., and Stone, Peter. Transferring instances for model-based reinforcement learning. In *ECML*, 2008.

Torrey, Lisa, Walker, Trevor, Shavlik, Jude W., and Maclin, Richard. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *ECML*, 2005.