To appear in *The Autonomous Agents and Multiagent Systems Conference* (AAMAS-07), Honolulu, Hawaii, USA, May 2007.

IFSA: Incremental Feature-Set Augmentation for Reinforcement Learning Tasks

Mazda Ahmadi Matthew E. Taylor Peter Stone
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188
{mazda, mtaylor, pstone}@cs.utexas.edu

ABSTRACT

Reinforcement learning is a popular and successful framework for many agent-related problems because only limited environmental feedback is necessary for learning. While many algorithms exist to learn effective policies in such problems, learning is often used to solve real world problems, which typically have large state spaces, and therefore suffer from the "curse of dimensionality." One effective method for speeding-up reinforcement learning algorithms is to leverage expert knowledge. In this paper, we propose a method for dynamically augmenting the agent's feature set in order to speed up value-function-based reinforcement learning. The domain expert divides the feature set into a series of subsets such that a novel problem concept can be learned from each successive subset. Domain knowledge is also used to order the feature subsets in order of their importance for learning. Our algorithm uses the ordered feature subsets to learn tasks significantly faster than if the entire feature set is used from the start. Incremental Feature-Set Augmentation (IFSA) is fully implemented and tested in three different domains: Gridworld, Blackjack and RoboCup Soccer Keepaway. All experiments show that IFSA can significantly speed up learning and motivates the applicability of this novel RL method.

Keywords

Reinforcement Learning

1. INTRODUCTION

Reinforcement learning [16] is a popular and successful framework for many agent-related problems because only limited environmental feedback is necessary for learning. Many algorithms exist to attempt to learn policies to take optimal sequential decisions to maximize expected reward and there have been many successful applications of reinforcement learning (e.g. Elevator Scheduling [2], Keepaway [14], and TDGammon [17]). However, as these methods scale up to more real world problems, which typically have very large state spaces, learning may be intractably slow.

One popular approach to solving reinforcement learning problems is value function approximation, where the agent attempts to learn the value for each state, i.e. the long term expected reward from entering a particular state, and then derives a policy from these values. The state space is the set of all possible states in an agent's environment. When the state space is very large, learning a value for each state can be intractable; if the state space is continuous, it is impossible. Function approximation techniques have been successfully used to speed up learning for problems with large or continuous state spaces, but solving large problems can be very hard given the "curse of dimensionality," and such methods may require extensive tuning.

We categorize efforts to advance the state of the art in reinforcement learning into three main categories:

- **General purpose algorithms:** Algorithms that are applicable to all, or a large subset of, the reinforcement learning problems. These algorithms do not explicitly use any expert knowledge (e.g. [1, 7]).
- **Case studies:** Specific problems that are solved with reinforcement learning. Domain-specific expert knowledge is typically utilized to speed up the learning process (e.g. [9, 17]).
- **General techniques for using domain knowledge:** Domain knowledge may be used to speed up reinforcement learning problems in at least two ways via hierarchy and abstraction [3] and by incorporating *advice* [8]. Hierarchical reinforcement learning (HRL) leverages a domain-dependant hierarchy designed by an expert to decompose the learning task. In advice taking agents, algorithms utilize advice from a domain expert to speed up the base reinforcement learning algorithm. In both cases it may be difficult to formulate the domain knowledge correctly, particularly in complex domains.

IFSA falls squarely in the third category as it is a general algorithm that leverages domain knowledge to speed up reinforcement learning. One of the main differences between IFSA and previous methods is that the knowledge the domain expert must supply to IFSA is significantly less detailed and may often be available from non-technical domain experts.

In a "real world" problem, the state space, by definition, contains an infinite number of possible state features from the world. The designer of a particular task must therefore pick the most relevant features (e.g. if the task is to travel to work in Boston, the weather report for London is not likely to be relevant) from which to construct a *feature-set*. In many (if not most) real world problems the feature-set can be broken into a series of subsets, where each subset can be used to learn a specific concept of the domain. Some concepts (feature-set subsets) may be more important than others. For example, consider the general task of navigation, where the goal is to find the best plan for getting to point B from point A. The agent must choose a path and a transportation method: *walking*, *taking a bus*, *taking a taxi*, or *driving*. The feature-set of the agent's state space includes:

1. The positions of A and B
2. Is it raining?
3. The type of shoe the agent is wearing
4. Does the agent have an umbrella?
5. The day of the week
6. The time

The agent can learn the concept of position and basic path planning when utilizing only the first feature. Features 2-4 can be used to learn how weather should change its policy. The last two features can be used to learn how to handle traffic. For the purposes of exposition, we assume that the concept of position is more important to the success of the agent than the rain concept.

A conventional approach to solving this type of reinforcement learning problem is to learn in a six dimensional state space with all features. The key insight of IFSA is that if an agent begins the learning process considering only a subset of features and adds new features while the learning progresses, the policy will improve faster than if the agent uses the entire feature-set from the beginning of training. Continuing our example, an agent can first learn a reasonable plan only using the first feature (position of A and B). After the agent has learned that basic navigation policy, it can then start using features about the time and day of the week to refine its policy, accounting for likely traffic situations. Finally, the agent can use the weather-related information to learn the optimal policy.

The order of feature-set subsets is important and should be determined by the domain expert based on their importance [1]. Results in the three different domains of Gridworld, Blackjack and RoboCup Keepaway show that the IFSA significantly speeds-up reinforcement learning.

The rest of the paper is organized as follows. Related work is presented in the next section. In Section 3 we briefly review the concept of value function reinforcement learning and Sarsa, the base learning algorithm we use. IFSA is specified in Section 4. Experimental evaluations in the three test domains are presented in Section 5 and Section 6 concludes.

2. RELATED WORK

Dietterich proposed using *MAXQ value-function decomposition* and provided a HRL algorithm which can be applied to the MAXQ decomposition [3]. However, the MAXQ decomposition must be performed by hand with expert domain knowledge. Sutton, Precup and Singh propose using temporal abstraction for reinforcement learning problems [15]. They define *options* as closed loop policies which consist of a series of primitive actions and run until a termination condition is reached. Options can then be used with different base learning algorithms, such as Q-learning [15], to speed up reinforcement learning. *Layered learning* [12] assumes that a domain expert provides a bottom-up, hierarchical task decomposition, and that learning occurs at each level. The main challenge of using HRL in real world applications is designing the hierarchy, which may be difficult or impossible. While there have been attempts to automatically build the hierarchies, the state of the art methods are still unable to learn complex hierarchies on very large problems. For example, Hengst decomposes a model-free MDP into a hierarchy of smaller MDPs [5]. However, he uses specific characteristics of state variables and it is not clear if the method is domain-independent. IFSA is designed so that performing the feature-set augmentation requires relatively less domain knowledge, though it still requires the hierarchy to be given.

Maclin and Shavlik use advice to speed up reinforcement learning [8], where advice is provided in a simple programming language and is given by the domain expert. They use a neural network for the utility function and update the neural network based on the received advice. Some work has been done by using natural language to provide advice (e.g. [6]) but a significant amount of domain knowledge is still required to make the advice usable by a learner. Instead of requiring an expert to know which actions should be performed in particular situations, our method requires only knowledge about (1) the concepts that need to be learned, (2) their relative importance, and (3) how these are related to particular state features.

3. PRELIMINARIES

This section provides a brief review of value function reinforcement learning and Sarsa [10, 11], a popular reinforcement learning algorithm. Note that IFSA may be used with any *value function* reinforcement learning algorithm but we focus on how it can be instantiated with Sarsa as this is the particular method utilized in our experiments.

In a typical reinforcement learning setting [16], an agent perceives state $s \in S$, takes some action $a \in A$, and then perceives the new state s'. Which state the agent arrives in is determined by the environment's transition function, $T(S \times A) \mapsto S$. Additionally, the agent receives a reward of r for arriving in state s', based on the reward function $R(S \times A) \mapsto \mathbb{R}$. The value function, $V^\pi(s)$, is defined as the average sum of rewards received when an agent starts in state s and follows policy π. $V^*(s)$ is the optimal policy, such that $\forall \pi, s$: $V^*(s) \geq V^\pi(s)$.

In value function reinforcement learning, the value function is learned (commonly with a *temporal difference method* [16]) and a policy is derived from V^π. Over time, V^π will approach V^* and the policy will improve. Given an optimal value function, the optimal action in any state can be computed:

$$\pi^*(s) = argmax_{a \in A}\{\sum_{s' \in S} Pr(s'|a,s)(V^*(s') + R(s'))\}$$

where $Pr(s'|a,s)$ is the probability that taking action a in state s result in the new state s'.

Sarsa is one popular temporal difference method which utilizes an action-value function to solve reinforcement learning problems. $Q^\pi(s,a)$ is defined as the average sum of the rewards when the agent starts from state s, takes action a, and follows policy π afterwards. Action-value functions are often used in lieu of value functions because determining the optimal policy, given an optimal action-value function, is easier than if only the value function is known.

Sarsa's update is based on its acronym: state, action, reward, state, action. The agent is in state s_t, takes action a_t, receives re-

[1]Many different orderings of adding the features could result in speeding up learning. However some ordering will fail to produce any speed up (see Section 5). For example, if the navigation agent is given only the current time as its first feature-set subset it could never hope to successfully navigate between positions A and B.

ward r_{t+1}, transitions to state s_{t+1}, and then selects action a_{t+1}. Sarsa then uses the following equation to update the Q function:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)],$$

where α is the learning rate and γ is a discount factor used to weight immediate rewards more heavily than future rewards.

4. IFSA

In reinforcement learning problems, the state space is typically composed of a set of features without regard for their relative importance. In this work, we take advantage of the fact that many reinforcement learning problems have subsets of features that can be used to learn different concepts about the task. These concepts may also be relatively more or less important to the overall performance of the learner. If concept A and B can be learned from feature sets Φ_A and Φ_B respectively, we say concept A is more important than concept B when the performance of the average policy learned using only Φ_A outperforms the average policy learned using Φ_B. We assume that in many cases a domain expert may easily recognize the more important concepts (as we illustrate in Section 5).

IFSA is a meta-learning method that defines what features the base learning method should use at any point in the learning process. Any value function reinforcement learning algorithm can thus be used as the base learner for IFSA. Additionally, IFSA can be used in conjunction with different speed up methods that do not manipulate the state space or feature set. IFSA is most appropriately applied to reinforcement learning problems and agents with the following three characteristics:

- Value function reinforcement learning is used as the base learning method.
- The task's state space is represented by a set of features and a conjunction of features is necessary to learn the optimal policy.
- There are subsets of features which may be used to learn sub-optimal policies and these subsets may be combined to learn better policies.

Many (if not most) reinforcement learning problems that are currently being used do have the above mentioned characteristics. The state space of almost all the real world complex problems (recall the navigation agent example) consist of a set of dependent features which can be prioritized based on their importance.

Assume that the state space (S) of the problem is represented by the following n features:

$$F = \{f_0, f_1, \cdots, f_{n-1}\}$$

The goal is to learn an optimal value function, $V^*(s)$ for each $s \in S$, which is equivalent to learning $V^*(F)$ for all possible values of features in F.[2] In many real world problems it is possible to find a set F' such that $F' \subset F$ and learning $V^*(F')$ first, and then using the learned function to initialize $V^*(F)$, is faster than simply learning $V^*(F)$ from scratch. If F' includes all the most important features then $V^*(F) \approx V^*(F')$. In that case, a significant speed up in the learning process can be achieved by starting the learning process with fewer features (F'). After the learning algorithm converges, learning with all features, F, can more easily find the optimal policy for the all features. IFSA relies on just this technique: it augments the feature set over time and utilizes the incrementally learned value functions to bootstrap learning until a value function is learned using the entire feature set.

Let $\Phi = (\phi_0, \phi_2, \cdots, \phi_{m-1} \subseteq F)$ be an ordered partitioning of the feature-set of the problem. The feature-sets utilized by IFSA will be the augmentation set $\{\phi_0, (\phi_0 \cup \phi_1), \ldots, (\phi_0 \cup \phi_1, \ldots, \phi_{m-1})\}$. Each ϕ_i consists of a set of features that the agent can use to learn a specific concept about the problem, where order is determined by importance. If $i > j$ then ϕ_i is more important than ϕ_j and should be considered first. Constructing Φ is thus where domain knowledge is directly leveraged.

Continuing the navigation example from the Introduction, Φ could be constructed as:

- **Basic set:** $\phi_0 = \{feature1\}$. Using the current and goal locations the agent can learn a basic travel plan.
- **Traffic factors:** $\phi_1 = \{feature5, feature6\}$. The agent can use the time and the day of the week to refine its basic travel plan to consider the traffic delays.
- **Rain factor:** $\phi_2 = \{feature2, feature3, feature4\}$. The agent can use these weather-related features to learn a better policy in inclement weather.

Once Φ is determined, the agent first attempts to learn a policy only using the features in ϕ_0. When the algorithm is *reasonably converged*, [3] the agent adds the features in ϕ_1 to the feature set is considering, augmenting its feature space. If the agent has learned value V_{ϕ_0} for a set of features ϕ_0, it initializes V_{ϕ_0,ϕ_1} with V_{ϕ_0} for all ϕ_0, ϕ_1s. Note that the effect of adding new features is essentially a splitting of old states into multiple new states that are all initialized identically, but can now learn different values. In this way, although the dimensionality of the feature space has changed, and therefore the domain of the value function is different, the learned value function is able to initialize V_{ϕ_0,ϕ_1}. Likewise, when the learning has reasonably converged using features $\{\phi_0 \cdots \phi_{i-1}\}$, the agent starts using ϕ_i, and initializes $V_{\phi_0 \cdots \phi_i}$ with $V_{\phi_0 \cdots \phi_{i-1}}$ for all $\phi_0 \cdots \phi_i$. The agent continues to add all feature sets in Φ until all the features are added. An overview of IFSA independent of domain or base learning method, is presented in Algorithm 1.

Algorithm 1 Pseudocode for IFSA algorithm

```
Let Φ = (φ_0, φ_2, ··· , φ_{m−1} ⊆ F) be an ordered augmentation of the feature-set of the problem.
for i=0 to m-1 do
    Learn V*(φ_0 ··· φ_i) until reasonably converged.
    for all φ_0 ··· φ_{i+1} do
        V*(φ_0 ··· φ_{i+1}) = V*(φ_0 ··· φ_i)
    end for
end for
```

In the navigation agent example, the agent first learns the path between two points, then by adding features ϕ_1 it learns different travel plans for different traffic situations. If the agent's goal is to travel from Austin to Houston, using ϕ_0 it quickly learns that just two paths (US-71 and US-290) are reasonable, and all other paths have low values. Also it learns that walking and taking a taxi are not reasonable. When the traffic feature is added, it can quickly learn the better path from the two in a specific traffic situation. Finally using ϕ_2 the agent adapts its policy for inclement weather so that it

[2] We actually attempt to learn $Q^*(s, a)$, but we focus on the closely related $V^*(s)$ for the purposes of exposition.

[3] A learning trial is considered *reasonably* converged when the change in the value of the optimal policy approaches zero (i.e. the slope of the learning curve is close to zero).

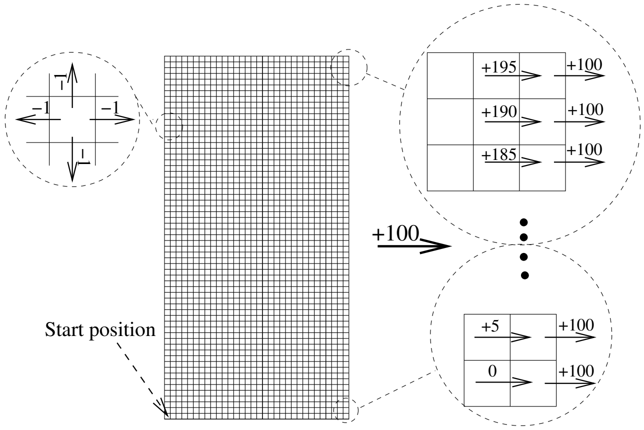

Figure 1: A diagram of the gridworld problem considered. Numbers on the arrows denote the reward for executing the associated move action.

learns not to attempt to walk across the city if it is raining and it is wearing expensive suede shoes.

5. EXPERIMENTAL RESULTS

Three different domains are used to evaluate IFSA to both demonstrate its applicability in multiple domains and to emphasize how Φ may be easily constructed in different domains. In all experiments IFSA is used to speed up learning when using Sarsa as a base learning algorithm. First we utilize a version of the gridworld problem. Second, a simplified version of Blackjack [16] is studied. Lastly, we examine a variant of Keepaway, a complex multiagent task that simulates robot soccer.

5.1 Gridworld

The first task we consider is situated in the gridworld domain. The grid is of size 20×40, the agent starts in the bottom-left grid cell, and the agent can move in the four cardinal directions. There is a reward of -1 for each attempted action. Additionally, moving right from any cell in the right most column results in a reward of $+100$, and moving right from a cell in the second most right column results in a reward equal to 5 times the row number that the agent is in (see Figure 1). In order to prevent infinite rewards, the "move left" action is disabled in the right most column. If the agent attempts a move that would take it off the top, bottom, or left sides of the grid, the action has no effect. An episode terminates when the agent exits the grid to the right.

The state space is composed of two features: the x-coordinate and y-coordinate of the agent. Since exiting the right side of the grid has the reward of $+100$, one could say that the x-coordinate is more important than the y-coordinate. However, our experiments show that constructing Φ with *either* ordering result in better performance than the unmodified Sarsa algorithm.

We run Sarsa with ϵ-greedy action selection. $\epsilon = 0.02$ and $\alpha = 0.5$ were chosen after initial experiments utilizing Sarsa in this domain (different values of ϵ in the range of $[0.01, 0.1]$ have been tested); the same parameter settings are used by Sarsa when it is used in conjunction with IFSA. Note that since the parameters have been selected to increase the learning speed of Sarsa, it may be a disadvantage for IFSA to use the same parameters.

We define reasonably converged in this domain to mean that there is no improvement to the policy in 4 consecutive episodes; when using IFSA with ϕ_0, after 4 consecutive episodes of no improvement, ϕ_1 is added to the agent's feature set. When the x-coordinate is used first the second feature is added after 29.9 ± 3.1 episodes on average. When the first feature is the y-coordinate, the second feature is added after 11.5 ± 3.5 episodes on average.

Three different Φ's are evaluated.

1. $\phi_0 = \{x-coordinate, y-coordinate\}$ (i.e. Sarsa without IFSA)
2. $\phi_0 = \{x-coordinate\}, \phi_1 = \{y-coordinate\}$ (IFSA)
3. $\phi_0 = \{y-coordinate\}, \phi_1 = \{x-coordinate\}$ (IFSA-rev, as it the reverse ordering from our initial intuition)

Experiments show that both IFSA and IFSA-rev significantly improve the speed of learning relative to Sarsa without IFSA. We show the learning curves for two levels of detail: episodes $0-100$ and episodes $0-1,000,000$. Even though the second feature is added quite early in the learning (before episode 30, on average), we find that the speed up has effects lasting over the length of the entire learning curve, each of which is an average of 100 independent trials.

In the graph of Figure 2, IFSA, IFSA-rev and unmodified-Sarsa are compared in the first 100 episodes. IFSA quickly learns a basic policy of taking only right actions, which yields a positive reward, while it takes roughly 100 episodes for the Sarsa algorithm to reach the same performance. Note that the inclusion of ϕ_1 does not result in any drop in the performance. On the other hand, although IFSA-rev eventually learns a policy with ϕ_0 and ϕ_1 that is superior to only using ϕ_0, there is a significant performance drop immediately after ϕ_1 is added around episode 9. However, IFSA-rev does eventually significantly outperform Sarsa, as shown on the next graph.

In the graph of Figure 3, the performance of the algorithms is compared for $100,000$ episodes where IFSA reaches the optimal policy. IFSA and IFSA-rev are each better based on different metrics (IFSA learns the optimal policy faster, but the total reward accrued by IFSA-rev in $100,000$ episodes is larger), but both are significantly better than Sarsa. A Student's t-test shows that the difference in performance between IFSA and Sarsa is statistically significant after episode $10,000$ ($p < 6 \times 10^{-5}$) and the difference between IFSA-rev and Sarsa is also statistically significant after episode 1000 ($p < 3 \times 10^{-8}$). Beyond the graph horizon, Sarsa converges to the optimal policy after about $1,100,000$ episodes, and IFSA-rev converges to the optimal after about $170,000$ episodes.

5.2 Blackjack

The Blackjack problem considered in this section is adopted from Sutton and Barto [16]. Blackjack is a popular card game played between a player and the dealer where the goal for the player is to have cards with the higher sum than the dealer without exceeding 21. All face cards count as 10 and aces can be counted as 11 or 1, whichever is more desirable. The game starts with the dealer and the player each dealt two cards, one of the dealer's cards is face up and the other is face down. If either the dealer or player has a sum of 21, they instantly win. Otherwise the player has two choices: ask for another card (*hit*) or stop (*stick*). The player can ask for more cards until he chooses to stop or the sum goes over 21 (a *bust*). If the player busts, he loses; if he stops then it is the dealer's turn. The dealer has a fixed strategy. The dealer must hit with a total less than 17 and must stick with 17 or more. If the dealer busts, the player wins. If both the dealer and player stop, the winner is determined by who has the higher total (if the total is equal, it is a draw). Note that some actions available in regular casino Blackjack

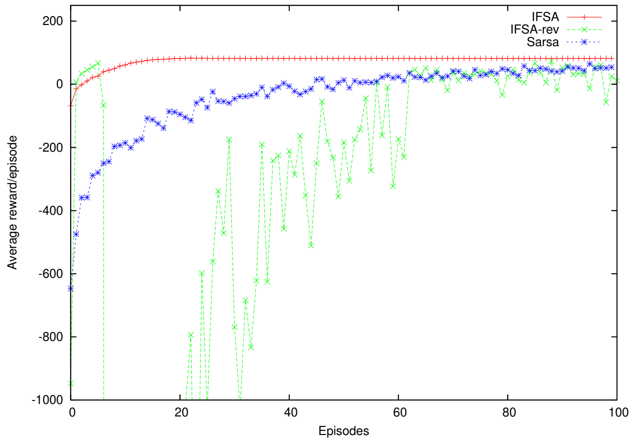

Figure 2: The initial performance of IFSA on the first 100 episodes in the Gridworld domain depends on the ordering of the state features.

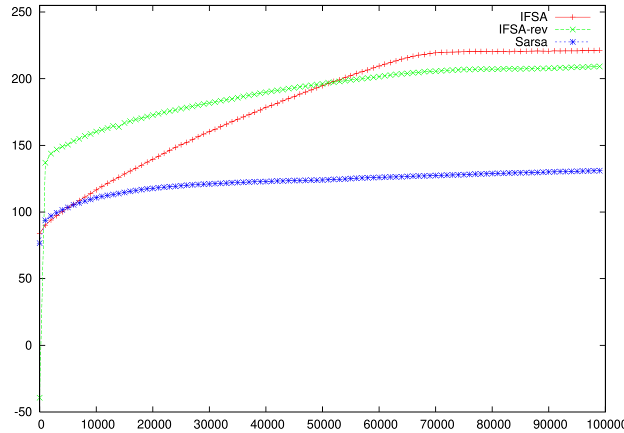

Figure 3: IFSA outperforms the base learning algorithm in Gridworld significantly, regardless of the ordering of the state features in Φ.

such as *split*, *double down*, and *insurance* are not available in this task.

The agent's actions are therefore *hit* and *stick*. Rewards are -1 for loss, 0 for a draw and $+1$ for a win. The states consist of the face up card of the dealer and the player's current hand. It is assumed that the dealt cards do not affect the distribution of future cards (i.e. cards are dealt from an infinite deck). The state space consists of three features:

1. *feature 0:* The player's current sum (12-21). Note that when the player has sum of less than 12 the optimal actions is always to hit.

2. *feature 1:* The value of the dealer's face-up card (1-10).

3. *feature 2:* A binary value which is true if the player holds an ace that can be counted as either 11 or 1 without exceeding a total of 21.

For the IFSA algorithm, $\phi_0 = \{feature0\}$, $\phi_1 = \{feature1\}$, and $\phi_2 = \{feature2\}$. The agent first learns a basic policy only using its own cards, it refines that policy considering what the dealer has, and then finally considers whether it has a usable ace. We again compare IFSA with unaugmented Sarsa learning. For both Sarsa and IFSA we use $\epsilon = 0.01$ and $\alpha = 0.01$, again chosen after initial experiments with unaugmented Sarsa showed these to be reasonable values.

In the experiments, ϕ_1 is added in episode 1000, and ϕ_2 is added at episode $3,000$. The two thresholds (1000 and 3000) are selected after informal experiments showed that the learning algorithms converged at approximately these two episode counts. To reduce the variance in performance due to chance, we pair learning trials of Sarsa and IFSA so that both trials in a pair have the same random seed.

For sake of clarity, the comparison between IFSA and unaugmented Sarsa are shown with two different levels of details: $10,000$ and $1,000,000$ episodes. Although the addition of features all occur within the first time-frame (at 1000 and 3000 episodes) the effects of IFSA are visible even after $1,000,000$ episodes.

In figure 4, the two algorithms are compared for the first $10,000$ episodes. The results are averaged over $10,000$ runs, and each point of the graph is the average reward from the 50 previous episodes. IFSA learns much faster only with ϕ_0 and continues to speed up learning with the addition of ϕ_1. Note that the addition of ϕ_1 does not result in any drop in the learning performance.

In Figure 5, the result of comparison between IFSA and Sarsa is shown for the first $1,000,000$ episodes. The learning curves are averaged over 100 runs and each data point is averaged in a $5,000$ episode sliding window. As shown in the graph, the advantage from IFSA is visible even after $1,000,000$ episodes. Beyond the graph horizon Sarsa and IFSA converge after about $5,000,000$ episodes. However they do not converge to the optimal policy because Sarsa does not achieve the optimal performance in this task.[4] (Recall that IFSA is a speed up method and is not expected to result in a better asymptotic policy). A Student's t-test shows the statistical significance of the advantage of IFSA compared to Sarsa in the first 75,000 episodes. After 75,000 episodes some of the differences are not statistically significant, although the mean for IFSA is always higher than that for Sarsa.

The above experiments show that IFSA significantly speeds-up learning compared to conventional Sarsa. We also experimented with different orders of adding features to the system. The order $(\phi_0, \phi_1, \phi_2) = (0-2-1)$ is slightly better than the more intuitive ordering $(\phi_0, \phi_1, \phi_2) = (0-1-2)$. Other orderings do not perform as well (see Figure 6). Four orderings of the features in IFSA outperform Sarsa. For both of the two orderings that underperform Sarsa, $\phi_0 = feature2$: the first feature considered is whether or not the player has an ace. A domain expert would easily identify this feature as less important, but if you play Blackjack *only* using information about whether you have an ace, the optimal policy performance is close to that of a random policy. On the other hand, if you only use information about the sum of the cards you hold or the dealer's card, it is possible to play much better than randomly.

5.3 Keepaway

As a 3rd test of the efficacy of IFSA, we consider the episodic multi-agent task of Keepaway in the RoboCup simulated soccer domain. In this work, we utilize agents based on version 0.6 of

[4] Even the policy that is considered optimal in [16] is shown to be sub-optimal [4].

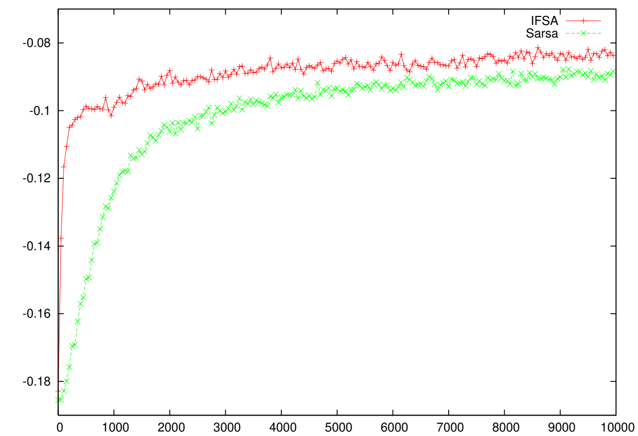

Figure 4: IFSA outperforms Sarsa for the first 10,000 episodes in the Blackjack domain because it is able to learn better policies much faster.

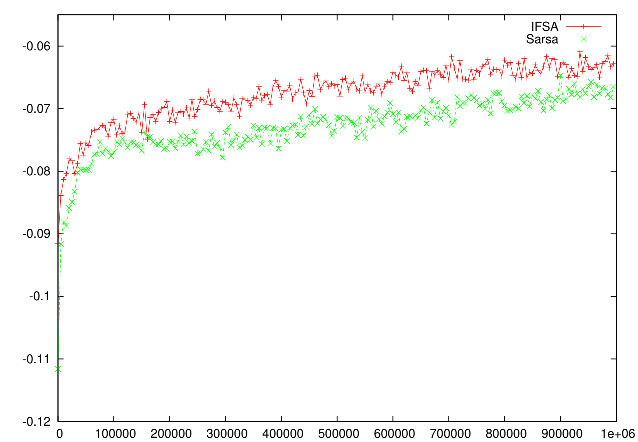

Figure 5: IFSA outperforms Sarsa for the first 1,000,000 episodes in the Blackjack domain.

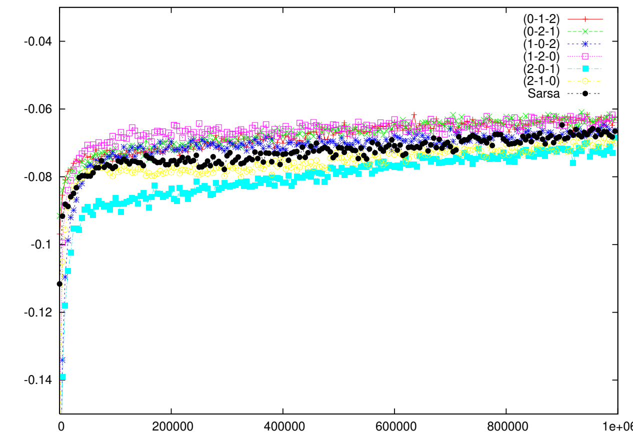

Figure 6: Comparing different orders of adding features for IFSA in the Blackjack domain.

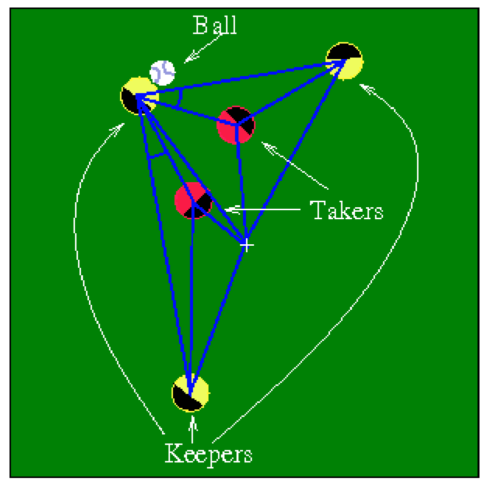

Figure 7: The 13 state variables used for learning with 3 keepers and 2 takers are made of 11 distances and 2 angles.

the benchmark players distributed by UT-Austin [13]. This section describes our task within the Keepaway domain and demonstrates how IFSA can be used to increase performance significantly.

Keepaway is a multiagent domain with noise in both sensors and actuators. It is considerably more complex than Gridworld or Blackjack in part due to its many continuous features, thus requiring function approximation. A set of *keepers* act to maintain possession of a ball while a second set of *takers* attempt to stop them. When the takers succeed in capturing the ball or kicking it out of bounds, the episode ends. Three keepers play against two takers in Figure 7. A keeper without the ball follows a fixed strategy, while a keeper in possession may choose between three macro actions. As keepers learn, they are able to maintain control of the ball within the specified area, increasing the average length of an episode. Takers follow a fixed strategy. Full details of the Keepaway domain can be found elsewhere [14].

In this work, we are concerned with tasks where different state features have different relative importance. We therefore modify the Keepaway task to that of *XOR Keepaway*, where one state feature is of lesser importance, but is necessary to achieve optimal performance. In doing so, we make the canonical 3 vs. 2 Keepaway task more difficult and show that IFSA is able to improve learning speeds on a more complex task.

The canonical 3 vs. 2 Keepaway formulation defines the state as being comprised of 13 state features and learning can succeed by simply treating all features independently [14]. Agents in the XOR Keepaway task use the same 13 state features but the task is modified so that optimal performance is only achieved when two of the state variables are used conjunctively. The two state features utilized are $dist(K_1, T_1)$ (the distance from keeper with ball to closest taker) and $dist(K_2, T)$ (the distance from closest teammate to any taker). Thus agents which do not represent $dist(K_1, T_1)$ and $dist(K_2, T)$ conjunctively are able to learn, but only to suboptimal performance levels.

The XOR Keepaway agents select from three actions: *Hold ball*, *Pass to closest teammate*, *Pass to furthest teammate*. However the task is defined to have four actions:

1. *Hold ball*: The keeper attempts to maintains possession of the ball.
2. *Good pass*: The keeper passes to its closest teammate and the takers do not move for one second.
3. *Bad pass*: The keeper passes directly to the closest taker, ending the episode.
4. *Pass to furthest teammate*: The keeper passes to the furthest teammate.

The details of the actions in the XOR Keepaway task are detailed below:

if Keeper attempts pass to closest teammate **then**
  if ($4m < d(K_1, T_1) < 6m$) XOR ($10m < d(K_2, T) < 12m$) **then**
    Execute *Good pass*
  else
    Execute *Bad pass*
  end if
else if Keeper attempts pass to furthest teammate **then**
  if ($7m < d(K_1, T_1) < 12m$) OR ($13m < d(K_2, T) < 16m$) **then**
    Execute *Bad pass*
  else
    Execute *Pass to furthest teammate*
  end if
else
  Execute *Hold ball*
end if

Thus Keepers must learn a policy based on both $dist(K_1, T_1)$ and $dist(K_2, T)$ conjunctively in order to avoid the *Bad pass* action.

To test IFSA, we run three sets of experiments in the XOR Keepaway domain with different representations:

1. *Independent Tiling*: 12 state features are tiled independently and $dist(K_2, T)$ is unused. We expect that agents using this representation will learn better policies initially but have lower asymptotic performance. Keepers will learn to keep the ball for longer periods of time but will not be able to differentiate between *Good pass* and *Bad pass*.
2. *Partially Conjunctive Tiling*: 11 state features are tiled independently, while $dist(K_1, T_1)$ and $dist(K_2, T)$ are tiled conjunctively. We expect that this representation will initially learn slower than the independent tiling representation because of the larger state space, but will be able to attain higher performance eventually because it is able to learn the XOR.
3. *IFSA Tiling*: Agents use the independent tiling representation for the first 25 simulated hours (ϕ_0 = 12 state features, excluding $dist(K_2, T)$). At 25 simulated hours, the representation is changed to partially conjunctive tiling(ϕ_1 = $dist(K_2, T)$) and weights are initialized via IFSA. We expect that this will yield the initial learning performance of the first representation while allowing us to learn a final policy equivalent to the second representation.

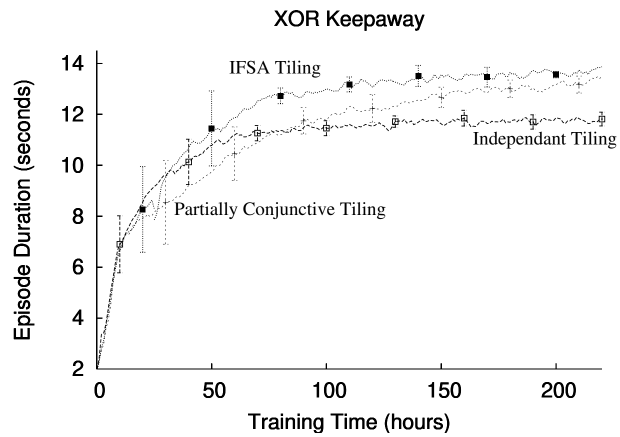

Figure 8: IFSA outperforms learning with just ϕ_0 (independent tiling) and learning with ϕ_0 and ϕ_1 (partially conjunctive tiling) from the start. 10 independent learning curves for each of the three setups are averaged and error bars show the standard deviation.

Figure 8 shows that IFSA can be effectively used in the XOR Keepaway task. Each of the three tests above are executed for 10 independent trials and their resulting learning curves are averaged. We use a Student's t-test to determine that agents using IFSA outperform the players using an independent tiling after 48 hours and outperform the players using the partially conjunctive tiling between 46 and 156 hours. Note that the final performance of players using IFSA and the partially conjunctive players is the same in the limit, as expected.

6. CONCLUSION AND FUTURE WORK

In this paper a general method for using domain knowledge to speed up value function reinforcement learning is presented. The proposed method (IFSA) is independent of the learning method used and utilizes a domain-dependant ordering of features to speed up learning. Each subset of features allow the learner to discover a concept and the subsets are ordered so that more important concepts are learned first. IFSA is implemented and evaluated in three different domains: Gridworld, Blackjack, and Keepaway. IFSA significantly speeds-up the learning by achieving higher performance with less training in all experiments.

In the future we intend to explore the possibility of automatically augmenting the feature-set by autonomously determining relevant feature subsets and their ordering for maximal speed up. IFSA could also be tested in additional environments, with different RL learning methods, and possibly in combination with other speed up methods listed in Section 2. Lastly, the idea that dividing a feature set appropriately into subsets could be used with policy search reinforcement learning methods that do not explicitly learn value functions.

Acknowledgement

This research was supported in part by DARPA grant HR0011-04-1-0035, NSF CAREER award IIS-0237699, NSF award EIA-0303609, and and ONR YIP award N00014-04-1-0545.

7. REFERENCES

[1] L. C. Baird and A. W. Moore. Gradient descent for general reinforcement learning. In M. J. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11, pages 968–974. The MIT Press, 1999.

[2] R. H. Crites and A. G. Barto. Improving elevator performance using reinforcement learning. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 1017–1023, Cambridge, MA, 1996. MIT Press.

[3] T. G. Dietterich. The MAXQ method for hierarchical reinforcement learning. In *Proceedings of the Fifteenth International Conference on Machine Learning*. Morgan Kaufmann, 1998.

[4] T. Edmunds. An optimum agent for the discrete triathlon. *In Reinforcement Learning Benchmarks and Bake-offs II A workshop at the 2005 NIPS conference*, 2005.

[5] B. Hengst. Discovering hierarchy in reinforcement learning with HEXQ. In *Proc. 19th International Conf. on Machine Learning*, pages 243–250, 2002.

[6] G. Kuhlmann, P. Stone, R. Mooney, and J. Shavlik. Guiding a reinforcement learner with natural language advice: Initial results in RoboCup soccer. In *The AAAI-2004 Workshop on Supervisory Control of Learning and Adaptive Systems*, July 2004.

[7] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[8] R. Maclin and J. W. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22:251–282, 1996.

[9] A. Y. Ng, H. J. Kim, M. I. Jordan, and S. Sastry. Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems 17*. MIT Press, 2004. To Appear.

[10] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG-RT 116, Engineering Department, Cambridge University, 1994.

[11] S. P. Singh and R. S. Sutton. Reinforcement learning with replaceing eligibility traces. *Machine Learning*, 22:123–158, 1996.

[12] P. Stone. *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*. MIT Press, 2000.

[13] P. Stone, G. Kuhlmann, M. E. Taylor, and Y. Liu. Keepaway soccer: From machine learning testbed to benchmark. In I. Noda, A. Jacoff, A. Bredenfeld, and Y. Takahashi, editors, *RoboCup-2005: Robot Soccer World Cup IX*, volume 4020, pages 93–105. Springer Verlag, Berlin, 2006.

[14] P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[15] R. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[16] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[17] G. Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.