Online Discovery of Feature Dependencies

Alborz Geramifard AGF@MIT.EDU
Finale Doshi-Velez FINALE@MIT.EDU
Joshua Redding JREDDING@MIT.EDU
Nicholas Roy NICKROY@MIT.EDU
Jonathan P. How JHOW@MIT.EDU
Massachusetts Institute of Technology, 77 Massachusetts Ave., Cambridge, MA 02139 USA

Abstract

Online representational expansion techniques have improved the learning speed of existing reinforcement learning (RL) algorithms in low dimensional domains, yet existing online expansion methods do not scale well to high dimensional problems. We conjecture that one of the main difficulties limiting this scaling is that features defined over the full-dimensional state space often generalize poorly. Hence, we introduce *incremental Feature Dependency Discovery* (iFDD) as a computationally-inexpensive method for representational expansion that can be combined with any online, value-based RL method that uses binary features. Unlike other online expansion techniques, iFDD creates new features in low dimensional subspaces of the full state space where feedback errors persist. We provide convergence and computational complexity guarantees for iFDD, as well as showing empirically that iFDD scales well to high dimensional multi-agent planning domains with hundreds of millions of state-action pairs.

1. Introduction

Large, combinatorial spaces in multi-agent domains pose significant challenges for RL agents. A standard approach for tractable learning in such domains is to map each state to a set of features, where each feature captures some property of the state (*e.g.,* Sutton, 1996). For example, a feature in a mission-planning scenario might correspond to an agent's fuel level. Function approximation on the feature space allows information to be shared across states with similar features (*i.e.,* generalization).

Linear approximators with binary features (*e.g.,* Albus, 1971) are a popular approximation choice as they are easy to train (Sutton, 1996; Geramifard et al., 2006) and are theoretically sound (Tsitsiklis & Van Roy, 1997). However, linear approximators assume that each feature contributes to the value function linearly independently of other features, which is not true for many cooperative multi-agent scenarios. For example, suppose we assign a feature to each (agent, location) pair, and the task requires multiple agents to approach distinct targets. A linear approximation of each state's value using only these features can reward agents for being at a target but cannot encode the advantage of two agents being at distinct targets. By adding conjunctions of existing features as new features (*e.g.,* agents A and B are at X and Y), these nonlinearities in the value function are encoded in the new representation; a linear approximator applied to this new representation will perform as well as a nonlinear approximator on the original features. Unfortunately, an initial set of n features will result in 2^n features in the new representation, rendering it computationally impractical.

While manually specifying important feature dependencies has been pursued in the literature (Albus, 1971; Sturtevant & White, 2006), learning dependencies online and expanding the representation automatically is becoming more popular as it simplifies the design process (*e.g.,* Munos & Moore, 2002; Ratitch & Precup, 2004; Whiteson et al., 2007). However, one of the main drawbacks of existing online expansion techniques is their inability to scale to high dimensional problems. A common property among these methods, as discussed in Sec. 5, is that new features correspond to a range of values along every dimension of the state space; features cannot ignore or constrain values in only a subset of dimensions. Thus, as the dimensionality of the state space grows, the degree of generalization of each feature (*i.e.,* the size of the set of states for which it is active) decays substantially. We conjecture that this property plays a major role in preventing existing techniques from scaling to higher dimensional domains.

To address this issue, we introduce *incremental Feature Dependency Discovery* (iFDD) as a general, model-free

Appearing in *Proceedings of the 28^{th} International Conference on Machine Learning*, Bellevue, WA, USA, 2011.

representational learning algorithm that expands the initial representation by creating new features which are defined in low dimensional subspaces of the full state space. These new features are created as feature dependencies and are introduced in regions where value function errors persist. This process eliminates improper generalization (*i.e.,* inappropriate sharing of weights among states) through time. Hence, given an initial set of binary features[1], we prove that iFDD reaches the best performance that any non-linear function approximation could achieve with those same features. We also show that the per-time-step computational complexity of iFDD scales with the number of *active* (*i.e.,* non-zero) features in each state, which can be kept sparse. Finally, compared to two state-of-the-art expansion methods, Sarsa using iFDD is shown to provide much faster learning in high dimensional problems.

2. Preliminaries

Markov Decision Processes A Markov Decision Process (MDP) (Sutton & Barto, 1998) is a tuple defined by $(\mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}, \gamma)$ where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P}^a_{ss'}$ is the probability of getting to state s' by taking action a in state s, $\mathcal{R}^a_{ss'}$ is the corresponding reward, and $\gamma \in [0, 1]$ is a discount factor that balances current and future rewards. A *trajectory* is a sequence $s_0, a_0, r_0, s_1, a_1, r_1, s_2, \ldots$, where the action a_t is chosen given a *policy* $\pi : \mathcal{S} \rightarrow \mathcal{A}$ mapping states to actions. Given a policy π, the value $Q^\pi(s, a)$ of each state-action pair is the expected sum of discounted rewards for an agent starting at s, doing a, and then following π:

$$Q^\pi(s, a) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s, a_0 = a \right]. \quad (1)$$

In discrete spaces, $Q^\pi(s, a)$ can be stored in a table. The optimal policy π^* maximizes the value function $V^\pi(s)$:

$$\begin{aligned} \forall s, \pi^*(s) &= \operatorname{argmax}_a Q^{\pi^*}(s, a), \\ V^{\pi^*}(s) &= V^*(s) = \max_a Q^{\pi^*}(s, a). \end{aligned}$$

Temporal Difference Learning The temporal difference (TD) error at time t is the difference between the current value for the state-action pair and an estimate based on the observed reward and the value for the next state-action pair:

$$\delta_t = r_t + \gamma Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t).$$

TD methods (Sutton, 1988) use the TD error δ_t as the gradient for reducing the error in the Q-function.

Linear Function Approximation A tabular representation of the action-value function Q is impractical for large state spaces, and a common approximation is to use a linear approximation of the form $Q^\pi(s, a) = \theta^T \phi(s, a)$. The feature representation $\phi : \mathcal{S} \times \mathcal{A} \rightarrow \Re^n$ maps states to a vector of features; the vector θ holds linear weights. Each element of the basis function $\phi(s)$ is called a *feature*; $\phi_f(s) = c$ denotes feature f has value c in state s.[2]

Of special interest is the set of basis functions ϕ where the output is a binary vector ($\phi : \mathcal{S} \times \mathcal{A} \rightarrow \{0, 1\}^n$). In this case, not only can the estimated values be computed efficiently (Buro, 1999), but the weights also indicate the importance of each binary property. Finally, if ϕ contains a unique feature that is active for each state-action pair then the function approximation reduces to a lookup table that assigns a separate value to each state-action pair (Eqn. 1).

3. Approach

We consider online value-based RL using linear function approximation given an initial set of binary features. The iFDD algorithm gradually captures nonlinearities within the linear approximation framework by introducing feature conjunctions as new binary features. We show in Sec. 4 that, especially in high-dimensional domains, gradually adding feature dependencies (*e.g.,* features corresponding to low dimensional subspaces of the full state space), encourages early generalization, which can speed up learning. The algorithm begins by building a linear approximation to the value function online using the initial set of binary features. It tracks the sum of absolute value of the approximation errors for all simultaneously activated feature pairs. We term the conjunction of each tracked feature pair as a *potential* feature and the cumulative approximation error associated with it as *relevance*. Once a potential feature's relevance exceeds a user-defined threshold, iFDD *discovers* that feature as a new binary feature, thus capturing the nonlinearity between the corresponding feature pair. We note that stochastic transitions may introduce some complications that we discuss in section 6. The algorithm proceeds in three steps:

(i) Identify *potential* features that can reduce the approximation error,

(ii) Track the relevance of each potential feature, and

(iii) Add potential features with relevance above a discovery threshold to the pool of features used for approximation.

Fig. 1 shows iFDD in progress. The circles represent initial features, while rectangles depict conjunctive features. The relevance of each potential feature f, ψ_f, is the filled part of the rectangle. The discovery threshold ξ, shown as the length of rectangles, is the only parameter of iFDD and

[1] Determining this initial set of binary features is important but is not the focus of this paper.

[2] For readability, we write $\phi(s)$ instead of $\phi(s, a)$, but ϕ always conditions on the action. When a new feature is discovered, it is added for all actions.

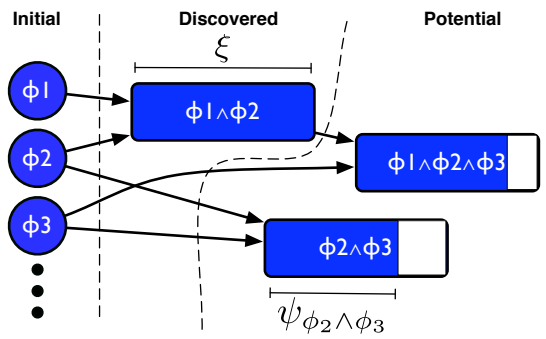


Figure 1. A snapshot of iFDD: Initial features are circles, conjunctive features are rectangles. The *relevance* ψ_f of a potential feature f is the filled part of the rectangle. Potential features are discovered if their relevance ψ reaches the discovery threshold ξ.

controls the rate of expansion. This parameter is domain-dependent and requires expert knowledge to set appropriately. However, intuitively lower values encourage faster expansion and improve the convergence to the best possible representation, while higher values slow down the expansion and allow for a better exploitation of generalization. While the ideal value for ξ will depend on the stochasticity of the environment, we found our empirical results to be fairly robust to the value of the discovery threshold.

We focus on iFDD integrated with TD learning, but any online, value-based RL method could supply the feedback error. Sec. 3.2 provides a proof that for the policy evaluation case the iFDD algorithm with TD learning will converge to the best possible function approximation given an initial set of binary features. We note that, if the initial features are such that no function approximation – linear or nonlinear – can satisfactorily approximate the underlying value function, then applying iFDD will not help. For example, if a key feature such as an agent's location is not included in the initial set of features, then the value function approximation will be poor even after applying iFDD.

3.1. Algorithm Details

The process begins with an initial set of binary features; let $\mathbf{F}$ be the current set of features used for the linear function approximation at any point in time. We use $\phi_f(s) = 1$ to indicate that feature $f \in \mathbf{F}$ is *active* in state s. After every interaction, we compute the local value function approximation error δ_t (*e.g.,* the TD error), the current feature vector $\phi(s_t)$, and update the weight vector θ (in the TD case, $\theta_{t+1} = \theta_t + \alpha_t \delta_t \phi(s_t)$, where α_t is the learning rate). Next, Algorithm 1 is applied to discover new features.

The first step in the discovery process (lines 1,2) identifies all conjunctions of active features as potential features.[3] Considering only conjunctive features is sufficient

Algorithm 1: Discover

Input: $\phi(s), \delta_t, \xi, \mathbf{F}, \psi$
Output: $\mathbf{F}, \psi$
1 **foreach** $(g, h) \in \{(i, j) | \phi_i(s)\phi_j(s) = 1\}$ **do**
2 $f \leftarrow g \wedge h$
3 **if** $f \notin \mathbf{F}$ **then**
4 $\psi_f \leftarrow \psi_f + |\delta_t|$
5 **if** $\psi_f > \xi$ **then**
6 $\mathbf{F} \leftarrow \mathbf{F} \cup f$

Algorithm 2: Generate Feature Vector (ϕ)

Input: $\phi^0(s), \mathbf{F}$
Output: $\phi(s)$
1 $\phi(s) \leftarrow \vec{0}$
2 *activeInitialFeatures* $\leftarrow \{i | \phi_i^0(s) = 1\}$
3 *Candidates* $\leftarrow \wp$(*activeInitialFeatures*) *sorted
4 **while** *activeInitialFeatures* $\neq \emptyset$ **do**
5 $f \leftarrow$ *Candidates*.next()
6 **if** $f \in \mathbf{F}$ **then**
7 *activeInitialFeatures* $\leftarrow$ *activeInitialFeatures* $- f$
8 $\phi_f(s) \leftarrow 1$
9 **return** $\phi(s)$

for iFDD to converge to the best approximation possible given the initial feature set; conjunctive features also remain sparse and thus keep the per-time-step computation low. The relevance ψ_f of each potential feature $f = g \wedge h$ is then incremented by the absolute approximation error $|\delta_t|$ (line 4). If the relevance ψ_f of a feature f exceeds the discovery threshold ξ, then feature f is added to the set $\mathbf{F}$ and used for future approximation (lines 5,6).

The computational complexity of iFDD can be reduced through a sparse summary of all active features. Note that if feature $f = g \wedge h$ is active, then features g and h must also be active. Thus, we can greedily consider the features composed of the largest conjunction sets until all active initial features have been included to create a sparse set of features that provides a summary of all active features.[4] For example, if initial features g and h are active in state s and feature $f = g \wedge h$ has been discovered, then we set the $\phi_f(s) = 1$ and $\phi_g(s), \phi_h(s) = 0$ since g and h are covered by f. Algorithm 2 describes the above process more formally: given the initial feature vector, $\phi^0(s)$, candidate features are found by identifying the active initial features and calculating its power set ($\wp$) sorted by set sizes (lines 2,3). The loop (line 4) keeps activating candidate features that exist in the feature set $\mathbf{F}$ until all active initial features are covered (lines 5-8). In the beginning, when no feature dependencies have been discovered, this function simply outputs the initial features.

[3] Conjunctions are stored in a "flat" representation, so there is only one conjunctive feature $a \wedge b \wedge c$ for the conjunction of features $a \wedge (b \wedge c)$ and $(a \wedge b) \wedge c$.

[4] Finding the minimum covering set is NP-complete but greedy selection gives the best polynomial time approximation.

Using the sparse summary also can help speed up the learning process. Suppose there are two features g and h that, when jointly active, result in high approximation errors. However, if one of them is active, then the approximation error is relatively low. Let f be the discovered feature $f = g \wedge h$. In our sparse summary, when g and h are both active in the initial representation, we set $\phi_f = 1$ and $\phi_g, \phi_h = 0$. If only one is active, then $\phi_f = 0$. Only non-zero features contribute to the value function approximation, so the learning update rule updates θ_f only if both $g = 1$ and $h = 1$. Otherwise, θ_f remains unchanged when θ_g or θ_h are updated. By separating the learning process for the states in which the feature conjunction $f = g \wedge h$ is true from states in which only one of the features g or h is true, we can improve our value function approximation estimates for the more specific cases without affecting the generalization in other states. The iFDD algorithm initializes the coefficient for a new feature f as $\theta_f = \theta_g + \theta_h$, so that the value function approximation remains unchanged when first adding a feature.

3.2. Properties and Performance Guarantees

The main virtue of iFDD is the way it expands the feature representation. Unlike other representation-expanding techniques (*e.g.,* Ratitch & Precup, 2004; Whiteson et al., 2007), iFDD increases the dimensionality of the space over which features are defined gradually as opposed to utilizing the full-dimensional state space. In this section, we show that our approach asymptotically leads to the best performance possible given the initial features: we first show that iFDD does not stop expanding the representation unless the representation is perfect or it is fully expanded. Next, we bound the asymptotic approximation error with respect to the true value function.

While iFDD can be used with any online, value-based RL method using binary features, we focus our analysis on iFDD combined with TD (iFDD-TD). Let s^i denote the i^{th} state. The feature function $\phi(s)$ outputs which features are active in state s; we represent all these output vectors in $\mathbf{\Phi}_{|\mathcal{S}|\times n}$, where the ith row corresponds to $\phi(s^i)^T$. As feature conjunctions are added as new features, the output dimensionality of $\phi(s)$ grows and adds columns to $\mathbf{\Phi}$.

Completeness of Feature Expansion To show that iFDD-TD will find the best representation possible given the initial set of features, we first argue that the iFDD-TD will either find a perfect representation or will add all possible conjunctive features:

Theorem 3.1 *Given initial features and a fixed policy π that turns the underlying MDP into an ergodic Markov chain, iFDD-TD is guaranteed to discover all possible feature conjunctions or converge to a point where the TD error is identically zero with probability one.*

Proof Suppose iFDD-TD has found ϕ as its final representation which neither sets the TD error zero everywhere nor includes all possible feature conjunctions. These properties imply that there is at least one state s that has at least two active features, g and h, where $g \wedge h$ has not been explored (if no state has more than one active feature, then the feature discovery algorithm would have no combined features to propose and the representation would have been fully expanded). The absolute sum of TD errors for this state s and this feature pair g and h after some time T_0 is $\sum_{t=T_0}^{\infty} |\delta_t| \mathcal{I}(s_t = s)$, where $\mathcal{I}$ indicates whether the agent was in state s at time t. By the ergodicity of the underlying Markov chain, we know that state s will be visited infinitely many times.

Since the value function approximation is not perfect, we know that there exists some state s' for which the TD error $\delta(s')$ is nonzero. We assumed that the Markov chain induced by the policy was ergodic; ergodicity implies that there exists a path of finite length from state s' to state s. Thus, over time, the TD error at state s' will propagate to state s. The only way for feature $f = g \wedge h$ to not be added is if the sum $\sum_{t=T_0}^{\infty} |\delta_t| \mathcal{I}(s_t = s)$ converges to some non-zero value that is less than the discovery threshold ξ.

We now argue that the sum $\sum_{t=T_0}^{\infty} |\delta_t| \mathcal{I}(s_t = s)$ diverges. Since the policy is fixed, we can consider the value function $V(s)$ instead of the action-values $Q(s, a)$. Let $V_\infty(s)$ be the converged fixed-point value function that would result from this learning process and $V_t(s)$ be the value function at time t. Let $\epsilon(s, t) = V_t(s) - V_\infty(s)$. Then we can write the absolute TD error at time t as

$$\begin{aligned} |\delta_t| &= |V_t(s) - r(s) - \gamma V_t(s')| \\ &= |V_\infty(s) + \epsilon(s,t) - r(s) - \gamma V_\infty(s') - \gamma\epsilon(s',t)| \\ &= |(V_\infty(s) - r(s) - \gamma V_\infty(s')) + (\epsilon(s,t) - \gamma\epsilon(s',t))| \end{aligned}$$

where, if the value function is not perfect, the first term $(V_\infty(s) - r(s) - \gamma V_\infty(s'))$ is some constant c (if the MDP is stochastic, the absolute TD error $|\delta_t| = |V_\infty(s) - r(s,a) - \gamma V_\infty(s')|$ will be nonzero simply because the successor state s' will vary). The second term $(\epsilon(s,t) - \gamma\epsilon(s',t))$ decreases as $V_t(s)$ converges toward $V_\infty(s)$; we can find a time $t = T_0$ such that $|\epsilon(s,t) - \gamma\epsilon(s',t)| < \frac{c}{2}$. Therefore, the sum of absolute TD errors $\sum_{t=T_0}^{\infty} |\delta_t| \mathcal{I}(s_t = s)$ is lower bounded by $\sum_{t=T_0}^{\infty} \frac{c}{2} \mathcal{I}(s_t = s)$ and diverges. ■

Corollary 3.2 *If at each step of iFDD-TD the policy changes but still induces an ergodic Markov chain (*e.g., *via ϵ-greedy or Boltzmann exploration), then iFDD-TD will explore all reachable features or converge to a point where the TD error is identically zero with probability one.*

Asymptotic Quality of Approximation Thm. 3.1 implies that iFDD-TD will converge to a fixed point where the TD error is zero everywhereor the feature space is fully

explored. Using the bound derived in Tsitsiklis and Van Roy (1997), we can bound the asymptotic approximation error of iFDD with respect to this final representation. Let $\mathbf{\Phi}_\infty$ be the feature matrix that includes all conjunctive features (including initial features), and $\mathbf{V}^*_{|\mathcal{S}|\times 1}$ be the vector representing the optimal value of all states.

Corollary 3.3 *With probability one, iFDD-TD converges to a weight vector θ and feature matrix $\mathbf{\Phi}$, where the approximated value function error, as originally shown by Tsitsiklis and Van Roy (1997) for a fixed set of linear bases, is bounded by:*

$$||\mathbf{\Phi}\theta - \mathbf{V}^*||_{\mathbf{D}} \leq \frac{1}{1-\gamma}||\mathbf{\Pi}\mathbf{V}^* - \mathbf{V}^*||_{\mathbf{D}},$$

where $\mathbf{D}_{|\mathcal{S}|\times|\mathcal{S}|}$ is a diagonal matrix with the stationary distribution along its diagonal, $\mathbf{\Pi} = \mathbf{\Phi}_\infty(\mathbf{\Phi}_\infty{}^T\mathbf{D}\mathbf{\Phi}_\infty)^{-1}\mathbf{\Phi}_\infty{}^T\mathbf{D}$, and $||.||$ stands for the weighted Euclidean norm.

Proof Thm. 3.1 states that iFDD-TD either finds a perfect representation with TD error zero everywhere (the approximation is exact) or exhaustively expands the whole representation. In the fully-expanded case, each state will have exactly one active feature, and thus the final feature matrix $\mathbf{\Phi}$ (excluding zero columns) will have full column rank. The representation reduces to the tabular case and we can apply Thm. 1 of Tsitsiklis and Van Roy's work (1997) to bound the error in the value function approximation. ■

Corollary 3.3 guarantees that our approach will achieve the best approximation for the value of policy π given the initial feature set.

In Sec. 4, we show empirically that our gradual approach – now combined with learning – learns more quickly than a full tabular approach (equivalent to starting out with a full set of "converged" conjunctions of basic features). Whether that value function is optimal depends on the initial choice of features; Corollary 3.3 states that we make the best possible use of given features asymptotically.

Maximum Features Explored Finally, we provide a bound on the maximum number of features iFDD-TD will explore. In general, if we begin with an initial feature set $\mathbf{F}$ with $|\mathbf{F}| = n$ elements, then the total number of possible features is the size of the power set of $\mathbf{F}$, $|\wp(\mathbf{F})| = 2^n$. In the specific case where initial features correspond to d independent variables that can each take q values – such as a continuous MDP discretized into q buckets for d independent dimensions – we can provide a tighter bound on the number of features to be explored because only one initial feature will be active for each dimension.

Remark Assume the initial set of features is defined for an MDP over d variables, each with domain size q. The maximum number of features explored (initial + discovered) using iFDD for such an MDP is $(q+1)^d - 1$.

Proof The number of feature conjunctions of size k is q^k. Hence the maximum number of features using Pascal's triangle amounts to:

$$\sum_{k=1}^{d}\binom{d}{k}q^d = \sum_{k=0}^{d}\binom{d}{k}q^d - 1 = (q+1)^d - 1. \blacksquare$$

A tabular representation for d variables uses q^d features, less than the $(q+1)^d - 1$ bound on features explored by iFDD. The difference occurs because the lookup table is the fringe of the feature tree expanded by iFDD process. While iFDD might explore more features than the tabular representation, we empirically find that iFDD often retains many fewer due to its nature of gradual expansion. Also because a minimal set of highest order clauses are active in any state, the asymptotic effective number of features used by iFDD is bounded by $|\mathcal{S}|$ (equal to the number of features in a tabular representation) unless a perfect representation is reached before discovering all possible features.

3.3. Computational Complexity

Given ϕ, let k_t be the maximum number of active features for any state and n_t be the total number of features in use at time t. The iFDD algorithm does not forget discovered features and ϕ uses a greedy set covering approach to form new feature vectors. Therefore, for all times $i < j \Rightarrow n_i \leq n_j, k_i \geq k_j$. Hence, $\forall t > 0, k_t \leq k_0$. The main loop of the $Discover$ function (Algorithm 1) requires k_t^2 operations. Using advanced hashing functions such as Fibonacci heaps, both $\mathbf{F}$ and ψ updates require $\mathcal{O}(1)$ operations. Hence the per-time-step complexity of Algorithm 1 is $\mathcal{O}(k_t^2) = \mathcal{O}(k_0^2)$. The outer loop of Algorithm 2 requires $\mathcal{O}(2^{k_0})$ operations in the worst case and each iteration through the loop involves $\mathcal{O}(1)$ lookup and $\mathcal{O}(k_0)$ set difference. Hence the total per-time-step complexity of evaluating the feature function ϕ is $\mathcal{O}(k_0 2^{k_0})$.

The computational complexity of both algorithms depends on k, the number of active features, and not n, the number of initial features. Thus, even if a method like tile coding, which may introduce large numbers of features, is used to create initial features, iFDD will still execute quickly.

4. Experimental Results

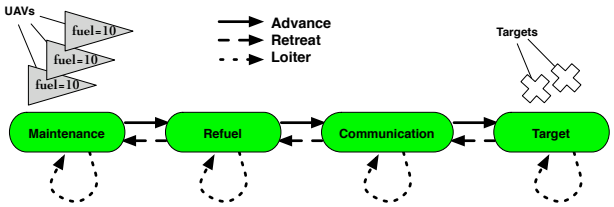
We compare the effectiveness of iFDD with Sarsa (see Sutton & Barto, 1998) against representations that (i) use only the initial features, (ii) use the full tabular representation, and (iii) use two state-of-the-art representation-expansion methods: adaptive tile coding (ATC), which cuts the space into finer regions through time (Whiteson et al., 2007), and sparse distributed memories (SDM), which creates overlapping sets of regions (Ratitch & Precup, 2004). All cases

used learning rates $\alpha_t = \frac{\alpha_0}{k_t}\frac{N_0+1}{N_0+\text{Episode \#}^{1.1}}$, where k_t was the number of active features at time t. For each algorithm and domain, we used the best α_0 from $\{0.01, 0.1, 1\}$ and N_0 from $\{100, 1000, 10^6\}$. During exploration, we used an ϵ-greedy policy with $\epsilon = 0.1$. Each algorithm was tested on each domain for 30 runs (60 for the rescue mission). iFDD was fairly robust with respect to the threshold, ψ, outperforming initial and tabular representations for most values.

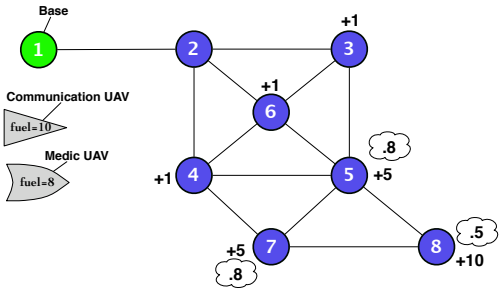
Inverted Pendulum Following Lagoudakis and Parr's work (2003), the system's state is the pendulum's angle and angular velocity, $(\theta, \dot{\theta})$, actions are three different torques, and episodes last up to 3000 steps. The initial features consisted of discretizing each dimension into 20 levels (for 120 features total). Fig. 2(a) plots the number of steps the pendulum remained balanced versus the total steps experienced (Sarsa using the original Gaussian representation is also plotted). In this relatively low-dimensional space, SDM and iFDD found good policies quickly, although SDM outperformed iFDD for the first test case after $10,000$ steps. Sarsa with the initial features never learned to balance the pendulum for more than $2,500$ steps, suggesting that the optimal value function was not linear in the initial features. The tabular representation reached near-optimal performance after about $60,000$ steps, while the Gaussian representation approach performed well after $40,000$ steps. ATC's initial advantage disappeared after many unhelpful splits that slowed generalization.

BlocksWorld The BlocksWorld task is to build a color-ordered 6-block tower starting with all the blocks originally on the table for a $+1$ reward. Each block-stacking attempt costs -0.01 and has a 30% chance of the block falling back onto the table. Episodes were capped at $1,000$ steps. Fig. 2(b) shows the return per episode. As expected, the initial representation does poorly because it cannot capture correlations between blocks. Our iFDD approach, on the other hand, discovered the necessary feature dependencies. The tabular representation could express the optimal policy, but its expressiveness hindered generalization: it needed three times the data as iFDD to achieve the same performance. Despite our optimization attempts, ATC and SDM both learned poorly in this larger, 36-dimensional domain.

Persistent Surveillance Fig. 3(a) shows an unmanned aerial vehicle (UAV) mission-planning task where three fuel-limited UAVs must provide continuous surveillance of two targets. At any time, the UAVs may be in maintenance, refuel, communication, or target states; deterministic actions allow them to stay in place or move to adjacent states. All actions except maintenance or refuel cost a unit of fuel; UAVs gain back fuel by staying in refuel state. UAVs have perfect sensors to monitor for motor and camera failures; failed parts can be repaired by going to maintenance. Parts have a 5% chance of failing at each time step: broken motors require immediate fixing while broken cameras prevent the UAV from monitoring a target. All together, the state is a 12-dimensional vector of remaining fuel, location, motor sensor status and camera sensor status for each of the three UAVs for a total of approximately 150 million state-action pairs. Initial features for each UAV are the fuel indicator, location, and the state of each of the two sensors. The action space is the combination of actions for all UAVs. We set γ to 0.9 and capped the episodes at $1,000$ steps.



(a) Persistent Surveillance

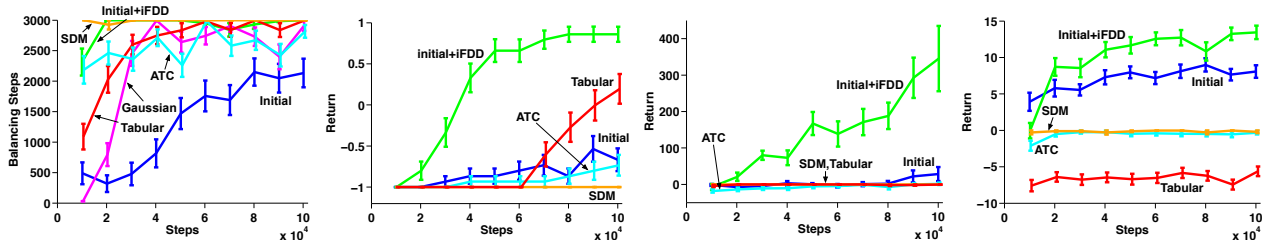


(b) Rescue Mission

Figure 3. Two UAV mission planning scenarios

The team received a $+20$ reward for every time step a target was reported; to report a target a UAV had to see it from the target state and a UAV had to relay the message from the communication state. UAVs were penalized for each unit of fuel used; running out fuel outside the refuel area cost -50 for each UAV and ended the episode. The results in figure 2(c) show that the lack of generalization slowed learning for the tabular case: even after 10^5 steps, it held all agents in the maintenance area for the entire mission. ATC and SDM had similarly inefficient generalization in this high-dimensional space. As before, the initial feature set could not capture the required correlations: it incorrectly generalized the consequences of running out of fuel. In contrast, the iFDD method corrected the improper generalization by incrementally adding feature conjunctions combining fuel and location. The resulting representation was able to switch to the better policy of sending UAVs out after only $20,000$ steps.

Rescue Mission Fig. 3(b) shows a mission-planning task where a medic UAV and a communication UAV must complete a rescue mission. The green circle shows UAVs' base location; numbers above the remaining nodes indicate the number of injured people at that node; and the cloud numbers are the probability of successful rescue. Victims are



(a) Inverted Pendulum (b) BlocksWorld (c) Persistent Surveillance (d) Rescue Mission

Figure 2. Empirical results of Sarsa algorithm using various representational schemes in in four RL domains: Inverted Pendulum, BlocksWorld, Persistent Surveillance, and Rescue Mission.

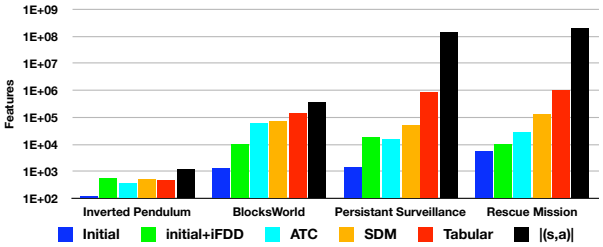


Figure 4. Average final feature counts. ATC and SDM, even using more features, performed poorly on high-dimensional examples. The black bar depicts the total number of state-action pairs.

saved when the medic UAV is at their node and the communication UAV is no farther than one edge away to relay back information. The medic UAV consumes a unit of fuel per movement or hover; the communication UAV may move (costs one fuel cell) or perch (costs nothing). Initial features were the fuel and position of each UAV, the communication UAV mode, and the rescue status at each node. The total state-action pairs exceeded 200 million. γ was set to 1.

The team received $+1$ for each person rescued, -0.1 for each unit of fuel spent, and -23 if not at base after 10 time steps or depleting all fuel. Fig. 2(d) shows the tabular representation was crippled by the scale of the problem. ATC and SDM fared somewhat better by capturing the notion of crashing early on but could not capture the complex reward structure. Learning with only the initial features proceeded quickly for the first $10,000$ steps, showing that the initial features are largely independent for this domain. However, after $20,000$ steps, iFDD's richer representation allowed it to encode a policy that outperformed all other methods.

Fig. 4 shows the average final feature counts for each domain. In Sec. 3.2, we showed iFDD can lead to a representation with more features than the tabular, but in the UAV domains, iFDD discovered approximately two orders of magnitude fewer features than tabular. Even when ATC and SDM had more features than iFDD, they still did not match iFDD's performance (except for SDM on pendulum).

Finally, to verify the effectiveness of error guided representation expansion, we compared iFDD with a random approach that adds feature conjunctions out of the potential set of features uniformly. For a fair comparison, we replaced iFDD's discovery threshold with a fixed discovery rate shared with the random approach. Table 1 shows the mean final performance of both methods (100 runs per domain except Persistent Surveillance with 30 runs); non-overlapping 95% confidence intervals show iFDD was significantly better than random expansion in all domains. Both methods took roughly the same computation time.[5]

Table 1. The final performance with 95% confidence intervals of iFDD and random expansion with equal number of features.

	Expansion Scheme	
Domain	Random	iFDD
Inverted Pendulum	2953 ± 30	$\mathbf{3000 \pm 0}$
BlocksWorld	-0.80 ± 0.06	$\mathbf{-0.24 \pm 0.10}$
Persistent Surveillance	174 ± 44	$\mathbf{280 \pm 49}$
Rescue Mission	$10 \pm .74$	$\mathbf{12 \pm .75}$

5. Discussion and Related Work

Adaptive Function Approximators (AFAs) have been studied for more than a decade (see Buşoniu et al., 2010, Section 3.6.2). We focused on refinement AFAs with linear and sublinear per-time-step complexities, amenable to online settings. We empirically showed that iFDD scales RL methods to high-dimensional problems by creating features in low dimensional subspaces of the full state space. For example, in the UAV domain, when a new feature conjunction of low fuel and being at base is discovered, this feature is defined in the two-dimensional subspace of fuel and location, ignoring all other dimensions. Previous work has done this process manually through tile coding (Sutton, 1996).

Both ATC and SDM create features in the full dimensional space limiting generalization as the dimensionality grows. While batch AFAs (Keller et al., 2006; Mahadevan et al., 2006; Parr et al., 2007) are promising, their computational demand often limit their application in online settings. Fur-

[5]Table 1 and Figure 2 differ because the discovery rate was set low enough to ensure a non-empty pool of potential features at all time steps. See our technical report for more information at `http://acl.mit.edu/iFDD-Tech.pdf`

thermore, batch methods face the inconsistency between the distribution of obtained samples and the distribution of samples under the current policy. Mahadevan and Maggioni (Mahadevan et al., 2006) suggested the use of a fixed policy for the sample gathering phase. This fixed policy can still cause problems for some domains, as the representation expansion method can exert a lot of effort representing complex value functions corresponding to poor initial policies. This observation motivated researchers to manually include samples that are highly likely to be visited during the execution of the optimal policy (*e.g.,* the bicycle domain in Petrik et al., 2010).

6. Conclusion and Future Work

We introduced iFDD as a general approach for expanding a linear function approximation of a value function from an initial set of binary features to a more expressive representation that incorporates feature conjunctions. Our algorithm is simple to implement, fast to execute, and can be combined with any online RL technique that provides an error signal. In the policy evaluation case, we proved that iFDD asymptotically produces an approximation to the value function that is no worse than using a fully-expanded representation (that is, a representation with all possible feature conjunctions). Furthermore, we empirically showed that iFDD not only boosts learning in classical RL domains, such as inverted pendulum and BlocksWorld, but also scales to UAV mission planning problems with hundreds of millions of state-action pairs where other adaptive methods, such as ATC and SDM, do not scale. Finally, while we focused on binary feature in this paper, the core idea of iFDD can be extended to other types of features. For example, given continuous features on $[0, 1]$, a threshold can convert features to a binary setting.

One area for future refinement is the noise sensitivity of iFDD. Highly stochastic domains may encourage the process to quickly add many features or add unnecessary features after the convergence of weights. While these factors do not affect the asymptotic guarantees, unnecessary features can increase computational load and slow down learning by refining the representation too quickly. Utilizing techniques that learn a model online, such as linear Dyna (Sutton et al., 2008), to substitute the sampled TD error with its expectation is a promising future work.

While orthogonal to the problem of how to best use a given set of features, specifying a good initial representation is also an important problem: although ATC and SDM did not scale well to high dimensional domains, they still have the advantage of not requiring an initial feature set. In contrast, the choice of discretization when applying iFDD is user-dependent: a coarse representation may not allow the value function to be approximated well, while a fine representation might limit generalization along a given dimension. We are currently exploring methods to combine adaptive discretization within the iFDD process.

7. Acknowledgement

We thank John Tsitsiklis, David Silver, Leslie Kaelbling, and Amir Massoud Farahmand for their constructive feedback. This work was sponsored by the AFOSR and USAF under grant FA9550-09-1-0522 and NSERC. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government.

References

Albus, James S. A theory of cerebellar function. *Mathematical Biosciences*, 10: 25–61, 1971.

Buro, Michael. From simple features to sophisticated evaluation functions. In *Computers and Games, Proceedings of CG98, LNCS 1558*, pp. 126–145. Springer-Verlag, 1999.

Buşoniu, L., Babuška, R., De Schutter, B., and Ernst, D. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Boca Raton, Florida, 2010.

Geramifard, Alborz, Bowling, Michael, and Sutton, Richard S. Incremental least-square temporal difference learning. In *The Twenty-first National Conference on Artificial Intelligence (AAAI)*, pp. 356–361, 2006.

Keller, Philipp W., Mannor, Shie, and Precup, Doina. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pp. 449–456, New York, NY, USA, 2006. ACM.

Lagoudakis, Michail G. and Parr, Ronald. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

Mahadevan, Sridhar, Maggioni, Mauro, and Guestrin, Carlos. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 8:2007, 2006.

Munos, Rémi and Moore, Andrew. Variable resolution discretization in optimal control. *Machine Learning Journal*, 49(2-3):291–323, 2002.

Parr, Ronald, Painter-Wakefield, Christopher, Li, Lihong, and Littman, Michael. Analyzing feature generation for value-function approximation. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pp. 737–744, New York, NY, USA, 2007. ACM.

Petrik, Marek, Taylor, Gavin, Parr, Ronald, and Zilberstein, Shlomo. Feature selection using regularization in approximate linear programs for markov decision processes. In *ICML '10: Proceedings of the 27th Annual International Conference on Machine Learning*, 2010.

Ratitch, Bohdana and Precup, Doina. Sparse distributed memories for on-line value-based reinforcement learning. In Boulicaut, Jean-François, Esposito, Floriana, Giannotti, Fosca, and Pedreschi, Dino (eds.), *ECML*, volume 3201 of *Lecture Notes in Computer Science*, pp. 347–358. Springer, 2004.

Sturtevant, Nathan R. and White, Adam M. Feature construction for reinforcement learning in hearts. In *In 5th International Conference on Computers and Games (ICCG)*, 2006.

Sutton, Richard S. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

Sutton, Richard S. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pp. 1038–1044. The MIT Press, 1996.

Sutton, Richard S. and Barto, Andrew G. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Sutton, Richard S., Szepesvari, Csaba, Geramifard, Alborz, and Bowling, Michael. Dyna-style planning with linear function approximation and prioritized sweeping. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, pp. 528–536, 2008.

Tsitsiklis, John N. and Van Roy, Benjamin. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

Whiteson, Shimon, Taylor, Matthew E., and Stone, Peter. Adaptive tile coding for value function approximation. Technical Report AI-TR-07-339, University of Texas at Austin, 2007.