Learning Behavior Styles with Inverse Reinforcement Learning

Seong Jae Lee Zoran Popović

University of Washington

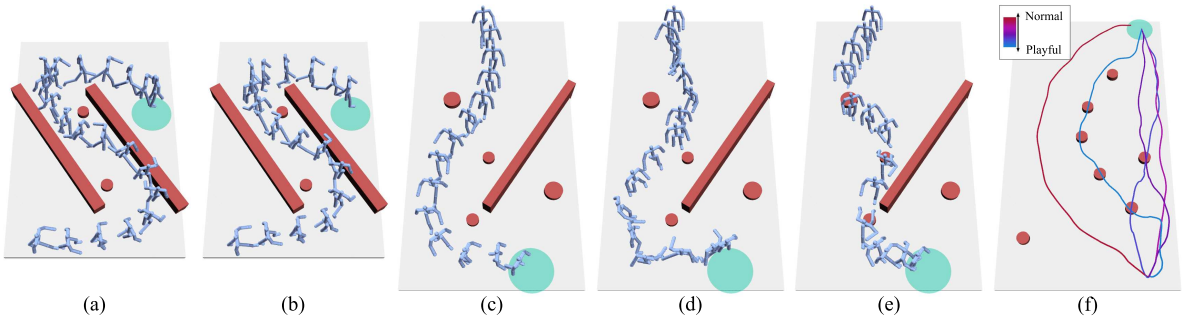

Figure 1: *(a) An expert's example for a normal obstacle avoiding style. (b) The learned normal style in the same map. (c) The learned normal style in an untrained environment. (d) The learned watchful style. (e) The learned playful style. (f) A variety of controllers created by blending between a normal style(red) and a playful style(light blue).*

Abstract

We present a method for inferring the behavior styles of character controllers from a small set of examples. We show that a rich set of behavior variations can be captured by determining the appropriate reward function in the reinforcement learning framework, and show that the discovered reward function can be applied to different environments and scenarios. We also introduce a new algorithm to recover the unknown reward function that improves over the original apprenticeship learning algorithm. We show that the reward function representing a behavior style can be applied to a variety of different tasks, while still preserving the key features of the style present in the given examples. We describe an adaptive process where an author can, with just a few additional examples, refine the behavior so that it has better generalization properties.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: Optimal Control, Data Driven Animation, Human Animation, Inverse Reinforcement Learning, Apprenticeship Learning

1 Introduction

Reinforcement learning-based character controllers have recently enjoyed great attention in computer graphics specifically for their ability to create optimized control policies over a given motion graph structure. The controllers can in real time determine the optimal transition through the motion graph that achieves the task goals. The task goals are represented as a *reward function* defined over the entire state and action space. Reinforcement learning controllers produce the motion with the highest accumulated rewards.

In general, task goals do not fully determine the desired behavior, or its underlying reward function. Often task goals can be achieved in a number of different ways even though only one of them would appear natural or specific to the intention of the designer. For example, the character may navigate to the specific position defined as a task goal, but do so by using many backward steps, by deploying many momentum-jarring motions, or by frequently switching from walking to running. None of these options would likely appear natural. To further define the appropriate motion, the designer needs to craft the behavior style by adding additional terms within the reward function to specify the desired style of achieving the task. Finding the appropriate reward function becomes harder as the target behavior requires subtleties of human motion such as walking cheerfully or being extremely cautious. Furthermore, as larger motion graphs include a greater variety of motion, the reward function needs to be more detailed to specify the behavior style.

Currently, the process of adjusting the reward function to achieve a desired behavior style is performed manually, in a fairly tedious trial-and-error process that often produces imperfect results. A key reason for this is that a small change in the importance of some reward function feature can lead to drastically different behavior. On the other hand, this reward function sensitivity provides enormous expressive power to represent a wide range of behavior variations.

A significantly more natural approach to determining the appropriate reward function is to simply provide examples of the specific behavior style in a few scenarios. The problem of finding the reward function from examples is known as inverse reinforcement learning (IRL). Since providing examples of the intended behavior style is natural to even the most novice designers, the behavior styles can be specified by simply acting out a few examples.

In this paper we present an automated framework that constructs a motion controller from a few example motions achieving the desired task. Instead of specifying a reward function, users provide a number of motion examples and a set of reward function components, or *features*, that seem to be relevant to the task. Our al-

gorithm returns an optimal controller that not only maximizes the match to the given examples, but is also capable of transferring the extracted behavior style to different settings. We present a new version of the apprenticeship learning (AL) algorithm of Abbeel and Ng [2004], adapting it to the specific setting of character controllers. Our method guarantees a deterministic optimal solution for our model, in contrast to the sub-optimal mixed policy provided by the original apprenticeship learning algorithm.

In our examples, we demonstrate the expressive power of representing the behavior style with a reward function. We show that we can learn reward functions capturing a variety of behavior styles, transfer behavior styles to different tasks and environments, and blend different styles. We also describe an iterative process of refining a controller with few additional examples which gracefully handles the inherent problem of overfitting when the initial sample set is too small.

After describing related work, Section 3 describes the details of our motion controller, and Section 4 describes the apprenticeship learning algorithm and our convex adaptation. In section 5, we describe our implementation and results. We conclude with this framework's advantages and limitations, and possible future research directions.

2 Related Work

Motion graphs created from human motion data are commonly used as a basis to reproduce motions (e.g. [Kovar et al. 2002; Shin and Oh 2006; Heck and Gleicher 2007; Beaudoin et al. 2008]). By traversing the motion graph, an arbitrary length of continuous natural motions can be created. Dynamic programming can be used to find the optimal traversal through the graph for a specific goal, while reinforcement learning can produce optimal traversal for the space of all goal parameterizations [Lee and Lee 2006; Treuille et al. 2007]. McCann and Pollard [2007] integrate a model of user behavior into reinforcement learning, enabling highly responsive real-time character control. Lo and Zwicker [2008] employ a tree-based regression method for reinforcement learning, while Lee et al. [2009] compute controllers for doing complex tasks by using a compact motion graph and a compact representation of value functions.

The inverse reinforcement learning recovers an unknown reward function with respect to the given behavior of a control system, or an *expert*, is optimal. Ng and Russell [2000] present an IRL algorithm learning a reward function that minimizes the value difference between example trajectories and simulated ones. Abbeel et al. [2004] present a refined algorithm that compares the trajectories with a more accurate metric and use the algorithm in the context of apprenticeship learning [Abbeel et al. 2008; Coates et al. 2009]. Syed et al. [2008] formulate apprenticeship learning into a linear programming problem in a stochastic environment, expediting the computation time. Ziebart et al. [2008] take a probabilistic approach based on the principle of maximum entropy, enabling the prediction of route preferences from GPS data of taxi-cab driving. In this paper, we adapt the apprenticeship learning [Abbeel and Ng 2004] because it fits our discrete motion model and our deterministic controller model.

The local aspects of movement style have been learned from examples. Brand and Hertzmann [2000] present a probabilistic method synthesizing stylistic human motion by learning a highly varied set of motion capture data using a cross-entropy optimization. Grochow et al. [2004] learns the reduced representation of pose style by modeling the pose space as a Gaussian process. The style difference has been transferred to unknown motion sequences by modeling a linear time-invariant system [Hsu et al. 2005]. Liu et al. [2005] present a nonlinear inverse optimization method that learns the biomechanical properties that give rise to emergent motion styles such as sadness, happiness, or limping. In contrast to work that focuses on stylistic motion variation, we focus on a style representation for a higher level control such as the way people choose to move through the environment, the way they choose to avoid obstacles, etc.

There has been much research towards explicitly modeling the behavior style of characters. Funge et al. [1999] model the behaviors of a character by selecting the appropriate action according to its perceived environment. Sung et al. [2004] present a control structure whose possible behaviors are based on situations, with a probabilistic action selection mechanism. Lee et al. [2007] learn group behaviors from a video capturing the trajectories of corresponding styles. Lau and Kuffner [2006] show an example of representing behavior styles by giving different relative rewards for specific motions. Automatically learning the behavior styles, or the variation of the character's intention, from examples in a reinforcement learning framework has not been addressed for character animation problems.

3 Motion Controller Model

Our IRL system sits on top of another RL-based optimization framework, which sits on top of a character animation system. The motion model uses a motion graph composed of short clips segmented into a step phase. Continuous motion is generated by concatenating a pair of motions which are adjacent in the motion graph. Constraint frames are defined on each clip in a way similar to [Treuille et al. 2007], which are used to align concatenating motions and to remove foot-skating. We used a partially connected motion graph, whose connectivity is determined by the similarity of motions. Each motion clip is parameterized on root's displacement and orientation with a constraint of fixing the ending pose as in [Lee et al. 2009].

Our motion controller is represented as a Markov decision process (MDP) (S, A, T, γ, R), where S is a set of states describing current condition and environment such as a current motion clip, the character's position and its orientation; A is a set of actions, which is a set of motion clips that can be concatenated to the current motion clip; $T : S \times A \mapsto S$ is a deterministic transition function; $\gamma \in (0, 1)$ is a discount factor; and $R : S \times A \mapsto \mathbb{R}$ is a reward function describing the task - for instance, for a controller whose objective is to reach a goal region, a positive reward only when the character is within the goal can be a valid reward function.

The controller decides which action to take from the given state based on a policy $\pi : S \mapsto A$. In our case, a policy gives a motion clip and motion parameters for the next step. This allows us to define a value function $V : S \mapsto \mathbb{R}$ that measures this long-term reward for each state under a policy π:

$$V_\pi(s) = \sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) \quad (1)$$

$$= \max_{a \in A} \Big(R(s,a) + \gamma V_\pi(T(s,a)) \Big), \quad (2)$$

while $s_0 = s$ and $s_{t+1} = T(s_t, \pi(s_t))\ \forall t$.

The optimal policy π^* given a MDP maximizes the value of any state:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\ V_\pi(s) \quad \forall s \in S. \quad (3)$$

Therefore, once we learn the optimal value function, we can com-

pute the optimal policy immediately:

$$\pi^*(s) = \underset{a \in A}{\operatorname{argmax}} \Big(R(s,a) + \gamma V_{\pi^*}(T(s,a)) \Big). \quad (4)$$

There are multiple ways to compute the optimal value function such as value iteration and policy iteration. In our case, we used the value iteration algorithm [Bellman 1957].

To enhance speed and the accuracy of value functions, we discretized state dimensions using parameterized motion clips. For example, when the position of the character is one of the discretized state dimensions, we apply appropriate parameters to the original motion clip so that the ending position of the motion always lies on the center of a grid. Such a technique was also used in Reitsma and Pollard [2004].

4 Inverse Reinforcement Learning

4.1 Problem Definition

The IRL problem is generally defined as follows: given measurements of an agent's behavior and its environment, determine the reward function that the agent is optimizing Russell [1998]. Among many interpretations of this problem, we use the problem definition of Abbeel and Ng [2004].

A reward function is defined as a linear combination of K features $\phi_i : S \times A \mapsto \mathbb{R}$:

$$R(s,a) := w \cdot \Phi(s,a) = \sum_{k=1}^{K} w_k \phi_k(s,a). \quad (5)$$

We call w feature weight. As a metric for comparing policies, we also define a discounted sum of feature values, or a *feature expectation*, given a state s and a policy π:

$$\mu(\pi, s) := \sum_{t=0}^{\infty} \gamma^t \Phi(s_t, \pi(s_t)), \quad (6)$$

where $s_0 = s$ and $s_{t+1} = T(s_t, \pi(s_t))\ \forall t$.

Similarly, given M expert's trajectories $\tau_m = (s_0^m, a_0^m, \ldots)\ \forall m \in \{1, \ldots, M\}$, we define an empirical estimate for the expert's feature expectation μ_E as the following:

$$\mu_E := \frac{1}{M} \sum_{m=1}^{M} \mu(\tau_m) := \frac{1}{M} \sum_{m=1}^{M} \sum_{t=0}^{\infty} \gamma^t \Phi(s_t^m, a_t^m). \quad (7)$$

We also define $\mu(w)$ as $\frac{1}{M} \sum_{m=1}^{M} \mu(\pi_w^*, s_m)$ given M expert trajectories[1], while π_w^* is the optimal policy given $R = w \cdot \Phi$. To avoid multiple optimal policies with different feature expectations, we set priorities between features for a preference order as a tie-breaking rule.

The goal of this problem is to find a reward function whose feature expectation is closest to the expert's, while the reward function is represented as a linear combination of given features:

$$\text{Find} \underset{R = w \cdot \Phi : ||w|| = 1}{\operatorname{argmin}} ||\mu(w) - \mu_E||. \quad (8)$$

Here, we set w to be a unit vector because reward functions with different scales produce the same optimal policy.

Let us examine this problem in detail. We can write the value of a state as a dot product of a feature weight and a feature expectation:

$$\begin{aligned} V_\pi(s) &= \sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) \quad (\because \text{Eq. 1}) \\ &= \sum_{t=0}^{\infty} \gamma^t w \cdot \Phi(s_t, \pi(s_t)) \quad (\because \text{Eq. 5}) \\ &= w \cdot \mu(\pi, s) \quad (\because \text{Eq. 6}). \end{aligned} \quad (9)$$

Now with Eq. 3, we induce:

$$w \cdot \mu(w) \geq w \cdot \mu \quad \forall w \in \mathbb{R}^K \text{ s.t. } \forall \mu \in U, \quad (10)$$

while $U := \{\mu(w)\ \forall w\}$ is a set of all optimal policies from our reward function domain. From this property, we induce that U is on the surface of a convex hull for U, $Co(U)$. If there is w such that $\mu(w)$ is inside the convex, i.e., $\exists \mu(w) = \sum_i \lambda_i \mu_i$ while all $\mu_i \in U$ are vertices of $Co(U)$, $\lambda_i \in [0,1]$, and $\sum_i \lambda_i = 1$, then $w \cdot \mu(w) = w \cdot \sum_i \lambda_i \mu_i \leq \max_{\mu_i} w \cdot \mu_i$. To satisfy this and Eq. 10, $w \cdot \mu(w) = w \cdot \mu_i$ for all i such that $\lambda_i > 0$. This means $\mu(w)$ is on the facet containing other μ_is whose corresponding λ_is are positive.

Furthermore, we induce U is the set of vertices of $Co(U)$. If there is an optimal policy whose feature expectation is on the surface, not on the vertices, its feature weight should be the surface's normal (due to Eq. 10). This means the optimal policy shares its value with other (also optimal) policies whose feature expectations are the neighboring vertices. However, our tie-breaking rule for multiple optimal policies prefers one of these vertices over the point in the middle.

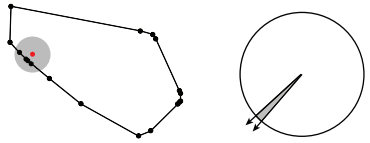

Figure 2: *Left: Visualization of feature expectations. U are black points, and μ_E is shown in red. Right: Gray region indicates the relative range of feature weights ($\{w = (\cos(\theta), \sin(\theta)) | \theta \in [222.5^\circ, 229.2^\circ]\}$) producing the feature expectation within a certain distance from μ_E (shown as gray circle in left) compared to the search space, represented as a black circle.*

Solving an IRL problem manually is often tedious work. Figure 2 visualizes how hard it can be. In this example, only less than two percent of feature weights domain produce satisfying policies. Moreover, the range of acceptable feature weights in the search space would drastically decrease with more features.

4.2 Apprenticeship Learning Algorithm

The apprenticeship learning (AL) algorithm [Abbeel and Ng 2004] solves this IRL problem by iteratively suggesting a feature weight candidate w^i and computing its feature expectation μ^i. The suggested feature weight is a unit vector maximizing the distance between μ_E and the convex polytope of known feature expectations

[1] Abbeel and Ng [2004] define a feature expectation as $\mu(w) := E[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) | w]$, while s_0 is drawn from the starting state distribution D. However, we draw s_0 from the observed trajectories. These definitions are interchangeable and do not affect the algorithm. However, we believe comparing policies on the observed trajectory space would be a better comparison metric than comparing policies on the expected feature expectation space, when the providing a large amount of examples is unavailable, as in our domain.

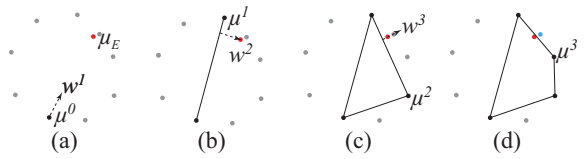

Figure 3: *Visualization of the AL algorithm. Gray, black, and red points represent $U \setminus U^i$, U^i, and μ_E, respectively. (d) When $\mu_E \in C^i$, i.e., when μ_E can be represented as a convex combination of elements of U^i, the algorithm ends. In this case, it misses the optimal solution $\arg\min_{u \in U} ||\mu_E - \mu||$, which is painted blue.*

$C^i := Co(U^i)$, while $U^i := \{\mu^0, \ldots, \mu^i\}$, towards its direction. As a result, the AL algorithm gradually expands C^i towards μ_E until no more improvement is observed. Figure 3 visualizes this process.

The algorithm returns a mixed policy, or an ensemble of policies with a certain probability whose feature expectation, on average, mimics the expert. However, using this mixed policy as it is does not work for our case: for example, randomly choosing between standing still and running does not give a walking behavior. In practice, one of the policies within the mix that is most similar to the expert is selected an used. Still, any individual selected policy does not have to be similar to the expert. For example, in Figure 3(d), the algorithm exits with the closest policy to the expert undiscovered. It happens because the AL algorithm minimizes the distance between the convex polytope C^i and μ_E, while we want to minimize $\min_{\mu \in U^i} ||\mu_E - \mu||$.

4.3 Convex Apprenticeship Learning Algorithm

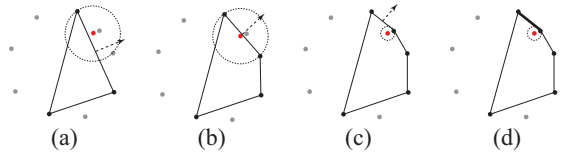

Figure 4: *Visualization of the Convex AL algorithm. Each iteration selects the closest non-saturated facet of C^i to μ_E and uses its normal as the next feature weight. (d) The algorithm exits when all facets within the distance $\min_{\mu \in U^i} ||\mu_E - \mu||$ are saturated (drawn as a bold line).*

To obtain a better policy, we present a novel algorithm based on our geometric interpretation. The Convex AL algorithm maintains the framework of iteratively suggesting a feature weight from observed policies, while changing the weight selection criteria from the original AL algorithm. The Convex AL algorithm builds a convex hull from known feature expectations and chooses the normal of a facet as a new feature weight. The basic idea is that if we keep expanding C^i towards the normals of its facets until no more expansion is found, we will recover $C^* := Co(U)$. To avoid unnecessary computation, we provide facet selection criteria and tighter ending conditions instead of computing the whole C^*. Figure 4 visualizes this process.

In Algorithm 1, $n(f)$ and $o(f)$ denote the normal and offset of a facet f, respectively so that $\forall x \in f\ x \cdot n(f) = o(f)$. We call a facet that cannot expand the convex towards its normal "saturated", and define $N(C)$ as a set of non-saturated facets of a convex polytope C. The Convex AL algorithm is guaranteed to halt and return the optimal solution in our setting. The proof of the optimality can be found in the Appendix.

Algorithm 1 Convex Apprenticeship Learning Algorithm

1: Run the AL algorithm until we can build a K dimensional convex hull $C^i \leftarrow Co(U^i)$.
2: **repeat**
3: $f \leftarrow \begin{cases} \operatorname{argmax}_{f \in N(C^i)} n(f) \cdot \mu_E - o(f) & \text{if } \mu_E \notin C^i, \\ \operatorname{argmin}_{f \in N(C^i)} \min_{x \in f} ||\mu_E - x|| & \text{otherise.} \end{cases}$
4: **if** $\mu(n(f)) \notin U^i$ **then**
5: $w^i \leftarrow n(f), \mu^i \leftarrow \mu(n(f))$.
6: $C^{i+1} \leftarrow C^i.insert(\mu^i)$.
7: $t^i \leftarrow \min_{\mu \in U^i} ||\mu_E - \mu||$.
8: $i \leftarrow i + 1$.
9: **end if**
10: **if** $n(f) \cdot \mu(n(f)) = o(f)$ **then**
11: Mark f as saturated.
12: **end if**
13: **until** $\begin{cases} N(C^i) = \emptyset & \text{or,} \\ \max_{f \in N(C^i)} n(f) \cdot \mu_E - o(f) < -t^i & \text{if } \mu_E \notin C^i, \\ \min_{f \in N(C^i)} \min_{x \in f} ||\mu_E - x|| < t^i & \text{otherwise.} \end{cases}$
14: **return** $\arg\min_{w \in \{w^0, \ldots, w^{i-1}\}} ||\mu_E - \mu(w)||$.

4.4 Applications to Controller Construction

Feature selection. Our framework adds features incrementally. When the current set of features is not descriptive enough, a learned policy does not replicate the input example. By observing the difference between the input and the learned policy, we determine additional parameter that should be considered (e.g. torso orientation), and add a new feature as a simple function of the parameter such as terms raised to different power, cosines or sigmoids. The designer does not have to worry about the overall feature simplicity as long as they have sufficient expressive power. In our cases, the features were determined within a few trials.

Iterative refinement of behavior. Much like many machine learning algorithms, our algorithm can suffer from overfitting. When too few examples are provided, the learned policy will invariably produce unexpected outcomes for untrained cases. It is frequently not clear how many examples are sufficient to fully generalize the behavior style to all environments and all possible goals. We address this problem by including the designer in the process so that she/he can provide corrected examples for each observed undesirable case. With each new example the reward function is further generalized. If the new example shares the same Markov model to the previous ones, the only overhead to re-learn the behavior style is evaluating additional feature expectations, which is negligible. Moreover, we can start from the previous densely constructed convex hull, which makes the re-learning significantly faster than running it from scratch. Even in the case of new examples coming from different Markov models, the running time of re-learning is linear in the number of examples. In our experience, satisfactory controllers are reached with a surprisingly few number of examples with this process, partly because each new example is provided at a failure point of the old style, thus having minimal redundancy with previous examples.

Behavior style transfer to different environments. Once a behavior style is learned from the examples (which themselves can be from different environments or different Markov models) it can be transferred to unknown environments, by changing the environ-

ment and the controller task goal while reusing the learned behavior style.

Blending behavior styles. Once several behavior styles are learned, one can create a variety of styles that blends between them. This allows us to use few examples to first determine extreme styles and then design a specific style by finding the appropriate mixture of learned styles. Obtaining such a design space of behavior styles would be very difficult with rule-based behavior descriptions.

5 Results

We applied our IRL framework to three different problems. For each experiment, we provided and learned three different styles. The Markov models were designed carefully so that running the value iteration algorithm can be done within a short amount of time with the aid of massive caching of feature values and transitions. All experiments were held on a PC with a Xeon 2.33GHz CPU and 8GB memory. The given examples and the learned controllers can be seen in the accompanying video.

We used Qhull [Barber et al. 1995] to construct convex hulls and IBM ILOG CPLEX to solve quadratic problems. Because the algorithm requires too many iterations to end, instead we stopped the algorithm when no improvement is observed for a long period of time. In all our cases, we could obtain a satisfactory controller with less than two hundred iterations and within a half day.

5.1 Simple Navigation

We tested our algorithm on navigation controllers that follow the specified direction in three styles - walking normally, walking backwards and running. A state is defined as a current motion clip and the desired direction relative to the orientation of the character. The angle dimension is discretized into sixty slots. We built stylized controllers on a motion database containing five hundred motion clips with a variety of stylized motions.

For each style, we sampled five trajectories by simulating a manually created controller, each of them five steps long. Each controller had a tiny motion graph with a dozen of manually selected motions and a reward function that returns a positive value when the character proceeds towards the desired direction. This naive reward function works because the motion database only contained relevant stylized motions. For example, a normal walking controller only had walking forwards motions in its motion graph so it does not have to care about its speed or torso orientation. Applying this naive reward function on the large database would not provide the desired behaviors.

For this experiment, we provided eight features:

- $|d|\cos(\theta_d - \theta_v)$ and $|d|\cos(2(\theta_d - \theta_v))$, where θ_d and θ_v are the desired direction and the moving direction, respectively. d is the displacement of current clip.
- $\cos(\theta_r)$, $\cos(2\theta_r)$, $\cos(\theta_h)$ and $\cos(2\theta_h)$, where θ_r and θ_h are the root orientation and the head orientation relative to the desired direction, respectively.
- $|v|, |v|^2, |v|^3$ and $|v|^4$, where v is the velocity.

5.2 Map Navigation

We built controllers that reach their goals while avoiding obstacles on different environments with varying map sizes and number and position of obstacles. We defined three behavior styles: a normal walking style that keeps a distance from obstacles, a playful style that prefers to jump over obstacles, and a watchful style that keeps an eye on the closest obstacle while avoiding it. A state is defined as a current motion clip, the position and orientation of the character. We discretized the angle dimension into 16 slots and the position dimensions into rectangular grids of 0.2 meter. The total number of states is about 0.5 million for each map. We tested our controllers on nine different maps.

We could make each controller work in all test maps with three examples. One of the examples and some of the learned controllers are shown in Figure 1. We also created a variety of controllers by interpolating the feature weights of two controllers (Figure 1(f)).

For this experiment, we provided eight features:

- The time consumed on moving.
- Four boolean indicators specifying if the character is in the goal region, jumping over an obstacle, walking by sideways or jumping.
- Two measures of the distance between the character and the obstacles: $\sum_o sigmoid(d_o - x, \sigma)$ for varying x and constant σ, where d_o is the distance to an obstacle o in meters. This gives bumps around obstacles with different radii.
- A measure describing if the character is facing the closest obstacle: $\cos(\theta_r - \theta_o)$, while θ_r is the orientation of the root and θ_o is direction to the closest obstacle whose distance is less than 1.5 meters from the character.

5.3 Bombardment Scenario

We built a video game-like environment where a character is supposed to avoid exploding bombs inside a narrow corridor. A bomb is dropped right above the character every three steps, and explodes after six steps. A state is defined as a current motion clip, the position and orientation of the character, and the position and the number of steps left before an explosion for each bomb. We synchronized the timings of bombs to the length of the motion clip being played to simplify the time dimension. We discretized the angle dimension into 8 slots and the position dimensions into rectangular grids of 0.2 meter. The total number of states is about a million.

The example of behaviors are gathered from a human player controlling the character with a game pad. The player attempted to express three different behavior styles: a relaxed style that gracefully navigates the map avoiding abrupt motions such as stopping and running, a scared style that always runs away, trying to be at the safest place, and a procrastinating style that moves as little as possible, and jumps away from the explosion at the last moment. For each style, we recorded about a minute long sequence of motions, and sliced it into multiple pieces so that the later parts of the example are treated as equally important as the earlier parts.

The learned controllers were satisfactory in the same environment to the examples, and also in an environment with a different bomb falling sequence. We also transferred the learned controllers on a two dimensional map, whose floor is falling apart reducing the possible region one can step on as time passes. Learning a style on simpler environments first and then applying it to more complex tasks allows us to avoid difficulties with collecting expressive user playing data on difficult tasks.

For this experiment, we provided seven features:

- Four boolean indicators describing if the character is damaged by an explosion, standing still, walking or jumping.
- Two measures of the distance between the character and each bomb: $\exp(-d_b)$, where d_b is the distance for each bomb b in

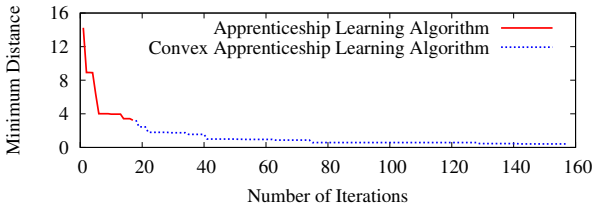

Figure 5: *Comparison between the AL algorithm and the Convex AL algorithm. Minimum distance is defined as* $\min_{\mu \in U^i} ||\mu_E - \mu||$. *AL algorithm ends with a suboptimal solution, while Convex AL algorithm runs until finding the optimal solution.*

meters.

- A measure of the turning amount of current motion: $\cos(\Delta\theta_r)$, where θ_r is the root orientation.

Figure 5 compares the performance of the AL algorithm and the performance of the Convex AL algorithm in the case of learning the procrastinating style. In this experiment, the policy from the AL algorithm is about eight times farther from the closest policy to the expert's trajectories than the one from the Convex AL.

6 Conclusion

We presented a framework that infers behavior styles from a small set of examples using IRL. The resulting styles can be applied to new unseen environments and new task goals. Style reward functions can also be blended together to produce a rich design space for character behaviors. We also present an iterative behavior refinement process that allows the designer to provide minimal number of samples for a desired behavior.

Towards this goal we introduced a novel algorithm that finds more precise controllers compared to the existing algorithm. Our algorithm guarantees the optimal solution when the Markov model is deterministic and discrete, which is the property of the character control problems. In contrast, the original apprenticeship learning computes a suboptimal mixed policy which is too inaccurate for deterministic character control problems.

Finally, we showed a number of applications. We learned stylized controllers from a large motion database and created general controllers working on various environments from a few examples. We also learned behavior styles from traces of video game player's data.

We see numerous opportunities for further improvement of our framework. First, much like all other IRL methods, ours greatly depends on the quality of the feature set. If a key feature of the reward function is missing, no IRL method will learn the meaningful behavior style. In the future, we will investigate automatic methods to determine the most salient features for each context. Second, the Convex AL runs considerably slower with a large number of features. Our algorithm expands a K-dimensional convex hull on each iteration. The maximum number of facets for a convex hull of v points is $O(\frac{v^{\lfloor K/2 \rfloor}}{\lfloor K/2 \rfloor !})$ [Barber et al. 1995], meaning that for large K, convex expansion will be prohibitively slow. We have found that most practical styles can be represented with a small number of features, and in fact in all our experiments convex expansion time is negligible compared to the time spent on reinforcement learning. Still, for possible applications with large numbers of features, the original AL should be used instead since it is less sensitive to the feature set size.

Looking forward, we believe our framework will be useful for human motion research in many directions. We would like to apply our method to learning the reward functions in the dynamic controller settings. When using reinforcement learning to control underactuated dynamic characters the accuracy of the reward function is particularly important. Furthermore, in these settings it is very difficult to find the reward function that works not just for one example but for a number of control settings. We are also interested in exploring alternative IRL methods that would allow for realtime controller design and adaptation. For example, the controller can adapt the reward function on the fly to perform optimally on changing environments. Finally, we believe our algorithm will be useful for adaptive game AI and adjusting the game level difficulty. The bombardment scenario showed we can learn the player's behavior from a short sequence of play examples. By extension, we would like to develop this direction further to learn the aptitude and the preference of a game player and uses this information to adapt the opponent's behavior. That way, the game can itself adapt to the player's strategies, presenting new challenges to the player with every new game playing session.

7 Appendix

Claim 1. *A facet* f *is marked as saturated if and only if* f *is a facet of* $C^* := Co(U)$. *In other words,* $n(f) \cdot \mu(n(f)) = o(f)$ *if and only if* $n(f) \cdot \mu \leq o(f)\ \forall \mu \in U$.

Proof. $\Rightarrow$ Suppose $n(f) \cdot \mu(n(f)) = o(f)$. From Eq 10, $n(f) \cdot \mu \leq n(f) \cdot \mu(n(f)) = o(f)\ \forall \mu$.

$\Leftarrow$ Suppose $n(f) \cdot \mu \leq o(f) \forall \mu \in U$. Because a facet f's vertices are from U, $\exists \mu' \in U \cap f$. Since $\mu' \in f$, $n(f) \cdot \mu' = o(f)$. From Eq 10, $o(f) = n(f) \cdot \mu' \leq n(f) \cdot \mu(n(f))$. Therefore, with our assumption, $n(f) \cdot \mu(n(f)) = o(f)$. □

Claim 2. *The Convex AL algorithm ends within a finite number of iterations.*

Proof. For each iteration, a facet's corresponding feature expectation should be either over or on the facet plane (due to Eq. 10). If it is over the facet, then it is out of the convex C^i, which means that we discovered a new policy. For the other case, the facet is one of the facets of C^* (from Claim 1) and is marked as saturated and will never be selected again. Therefore, for each iteration, the algorithm either reveals an undiscovered policy or marks a non-saturated facet of C^* as saturated (or does both).

However, in our discrete setting $|S|$ and $|A|$ are finite, the number of possible deterministic policies is also finite. Therefore, $|U|$ and the number of facets in C^* should be also finite. Therefore, the algorithm has to terminate within a finite number of iterations. □

Claim 3. *If* $\mu_E \notin C^i$ *and* $\max_{f \in N(C^i)} n(f) \cdot \mu_E - o(f) < -\min_{\mu \in U^i} ||\mu_E - \mu||$, *then* $\mu^* := \arg\min_{\mu \in U} ||\mu - \mu_E|| \in U^i$.

Proof. Suppose $\mu^* \notin U^i$ while $\max_{f \in N(C^i)} n(f) \cdot \mu_E - o(f) < -\min_{\mu \in U^i} ||\mu_E - \mu||$ and $\mu_E \notin C^i$. Because $\mu^* \notin C^i$, there is a facet f such that $n(f) \cdot \mu^* > o(f)$. From Claim 1, $f \in N(C^i)$. However, $n(f) \cdot \mu_E - o(f) > \min_{\mu \in U^i} ||\mu_E - \mu||$:

$$\begin{aligned} -\min_{\mu \in U^i} ||\mu_E - \mu|| &\leq -||\mu_E - \mu^*|| \quad (\because \text{def. of } \mu^*) \\ &\leq n(f) \cdot (\mu_E - \mu^*) \quad (\because ||n(f)|| = 1) \\ &< n(f) \cdot \mu_E - o(f). \end{aligned} \tag{11}$$

This contradicts to our assumption. □

Claim 4. *If* $\mu_E \in C^i$ *and* $\min_{f \in N(C^i)} \min_{x \in f} ||\mu_E - x|| > \min_{\mu \in U^i} ||\mu_E - \mu||$, *then* $\mu^* \in U^i$.

Proof. Suppose $\mu^* \notin U^i$ while $\min_{f \in N(C^i)} \min_{x \in f} ||\mu_E - x|| > \min_{\mu \in U^i} ||\mu_E - \mu||$ and $\mu_E \in C^i$. Because $\mu_E \in C^i$ and $\mu^* \notin C^i$, there is a facet f between these two points, i.e., $\exists x \in f$ such that $x = t\mu_E + (1-t)\mu^*, t \in (0,1)$. In this case, $\min_{x' \in f} ||\mu_E - x'|| \leq ||\mu_E - x|| < ||\mu_E - \mu^*|| < \min_{\mu \in U^i} ||\mu_E - \mu||$. However,

$$
\begin{aligned}
& n(f) \cdot x = n(f) \cdot (t\mu_E + (1-t)\mu^*) \\
\Leftrightarrow & o(f) = n(f) \cdot (t\mu_E + (1-t)\mu^*) \ (\because x \in f, n(f) \cdot x = o(f)) \\
\Rightarrow & (1-t)(n(f) \cdot \mu^* - o(f)) = -t(n(f) \cdot \mu_E - o(f)) \\
\Rightarrow & n(f) \cdot \mu^* - o(f) > 0 \ (\because n(f) \cdot \mu_E < o(f), t \in (0,1)). \quad (12)
\end{aligned}
$$

By Claim 1, $f \in N(C^i)$, contradicting to our assumption. □

Acknowledgements

We thank Yongjoon Lee for providing motion data. This work was supported by the UW Animation Research Labs, UW Center for Game Science, Samsung Fellowship, Microsoft, Intel, Adobe and Pixar.

References

ABBEEL, P., AND NG, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning*, ACM Press.

ABBEEL, P., DOLGOV, D., NG, A., AND THRUN, S. 2008. Apprenticeship learning for motion planning, with application to parking lot navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE.

BARBER, C. B., DOBKIN, D. P., AND HUHDANPAA, H. 1995. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software 22*, 469–483.

BEAUDOIN, P., COROS, S., VAN DE PANNE, M., AND POULIN, P. 2008. Motion-motif graphs. In *Proceedings of the 2008 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 117–126.

BELLMAN, R. E. 1957. *Dynamic Programming*. Princeton University Press.

BRAND, M., AND HERTZMANN, A. 2000. Style machines. In *Proceedings of SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 183–192.

COATES, A., ABBEEL, P., AND NG, A. Y. 2009. Apprenticeship learning for helicopter control. *Communications of the ACM 52*, 7, 97–105.

FUNGE, J., TU, X., AND TERZOPOULOS, D. 1999. Cognitive modeling: knowledge, reasoning and planning for intelligent characters. In *Proceedings of SIGGRAPH 99*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 29–38.

GROCHOW, K., MARTIN, S. L., HERTZMANN, A., AND POPOVIĆ, Z. 2004. Style-based inverse kinematics. *ACM Transactions on Graphics 23*, 3, 522–531.

HECK, R., AND GLEICHER, M. 2007. Parametric motion graphs. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, ACM, 129–136.

HSU, E., PULLI, K., AND POPOVIĆ, J. 2005. Style translation for human motion. *ACM Transactions on Graphics 24*, 3, 1082–1089.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. In *Proceedings of SIGGRAPH 2002*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 473–482.

LAU, M., AND KUFFNER, J. J. 2006. Precomputed search trees: Planning for interactive goal-driven animation. In *Proceedings of the 2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, Eurographics Association, 299–308.

LEE, J., AND LEE, K. 2006. Precomputing avatar behavior from human motion data. *Graphics Models 68*, 2, 158–174.

LEE, K. H., CHOI, M. G., HONG, Q., AND LEE, J. 2007. Group behavior from video: a data-driven approach to crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, Eurographics Association, 109–118.

LEE, Y., LEE, S., AND POPOVIĆ, Z. 2009. Compact character controllers. *ACM Transaction on Graphics 28*, 5, 169:1–169:8.

LIU, K., HERTZMANN, A., AND POPOVIĆ, Z. 2005. Learning physics-based motion style with nonlinear inverse optimization. *ACM Transactions on Graphics 24*, 3 (Aug.), 1071–1081.

LO, W., AND ZWICKER, M. 2008. Real-time planning for parameterized human motion. In *Proceedings of the 2008 Eurographics / ACM SIGGRAPH Symposium on Computer Animation*, Eurographics Association, 29–38.

MCCANN, J., AND POLLARD, N. 2007. Responsive characters from motion fragments. *ACM Transactions on Graphics 26*, 3 (July), 6:1–6:7.

NG, A. Y., AND RUSSELL, S. 2000. Algorithms for inverse reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning*, Morgan Kaufmann, 663–670.

REITSMA, P. S. A., AND POLLARD, N. S. 2004. Evaluating motion graphs for character navigation. In *Proceedings of the 2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, Eurographics Association, 89–98.

RUSSELL, S. 1998. Learning agents for uncertain environments (extended abstract). In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, ACM Press, 101–103.

SHIN, H. J., AND OH, H. S. 2006. Fat graphs: constructing an interactive character with continuous controls. In *Proceedings of the 2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, Eurographics Association, 291–298.

SUNG, M., GLEICHER, M., AND CHENNEY, S. 2004. Scalable behaviors for crowd simulation. *Computer Graphics Forum 23*, 3, 519–528.

SYED, U., BOWLING, M., AND SCHAPIRE, R. E. 2008. Apprenticeship learning using linear programming. In *Proceedings of the 25th international conference on Machine learning*, ACM, 1032–1039.

TREUILLE, A., LEE, Y., AND POPOVIĆ, Z. 2007. Near-optimal character animation with continuous control. *ACM Transactions on Graphics 26*, 3 (July), 7:1–7:7.

ZIEBART, B. D., MAAS, A., BAGNELL, J. A., AND DEY, A. K. 2008. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd national conference on Artificial intelligence*, AAAI Press, 1433–1438.