Compact Character Controllers

Yongjoon Lee Seong Jae Lee Zoran Popović

University of Washington

Abstract

We present methods for creating compact and efficient data-driven character controllers. Our first method identifies the essential motion data examples tailored for a given task. It enables complex yet efficient high-dimensional controllers, as well as automatically generated connecting controllers that merge a set of independent controllers into a much larger aggregate one without modifying existing ones. Our second method iteratively refines basis functions to enable highly complex value functions. We show that our methods dramatically reduce the computation and storage requirement of controllers and enable very complex behaviors.

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

Keywords: Optimal Control, Data Driven Animation, Human Animation

1 Introduction

Over the past decade motion graphs composed from a large set of captured motion data have been commonly used to construct interactive controllers of realistic human motion. More recently, reinforcement learning approaches have shown that for a given motion graph a wide variety of optimized controllers can be automatically constructed [Lee and Lee 2004; McCann and Pollard 2007; Treuille et al. 2007; Lo and Zwicker 2008].

While controllers based on motion graphs have been successfully demonstrated for specific subsets of human motions, the problem of constructing controllers that cover the entire space of all human behaviors and motion tasks remains an open problem. Extrapolating current techniques to build such a controller would require a prohibitively large number of motion clips and value functions of dimensionality beyond what current methods can handle.

In this paper we consider two fundamental hurdles towards the goal of comprehensive controllers. The first hurdle is the selection of the right compact subset of motion data that covers a large number of different controllers. It is important to provide the drastically different sets of motion examples that each task requires, while minimizing non-essential redundant motion data so that the entire system can accommodate a wider variety of behaviors. In addition, for an aggregate controller system that represents many different behaviors by combining the separately constructed controllers, automatically finding the natural transition examples among the controllers is an important advance. It enables the individual controllers to be designed without worrying about their connection to other existing controllers. For example, we can seperately create a standing controller and running controller, and then automatically identify necessary speed-up motions to make the transition between them more realistic. This is a practical way to rapidly expand a library of achievable tasks with minimal design cost. Compact yet maximally expressive sets of clips allow complex motion controllers to fit on game platforms with a relatively limited storage (e.g. mobile devices).

The second hurdle towards automatic synthesis of comprehensive controllers is the appropriate selection of compact basis functions used for the value function representations. When a complex task requires a lot of parameters to be modeled, its value function becomes sufficiently high-dimensional so that naive distribution of basis functions over such space becomes impractical. An automatic basis selection and refinement method is required before larger problems can be solved.

We present methods for constructing complex individual and connecting controllers over an automatically selected compact set of motion clips. Our methods systematically analyze the controller's preferences and performance bottlenecks to produce larger aggregate controllers as well as complex high dimensional controllers using less resources. We demonstrate the effectiveness of our framework on a number of controller examples, and provide compactness and optimality comparisons.

2 Related Work

Pre-planning and learning methods have been used to create interactive controllers, including value iteration [Lee and Lee 2004], explicitly calculating (and caching) reward over a short window of time [Ikemoto et al. 2005; Lau and Kuffner 2006], learning user command statistics for responsive characters [McCann and Pollard 2007], constructing linear approximate value functions using continuous basis functions [Treuille et al. 2007], and a tree-based regression method on parameterized data [Lo and Zwicker 2008].

Since data-driven animation by nature requires a large amount of example motion data, researchers have tried to keep the data requirement manageable. Many works since Lamouret and van de Panne [1996] pruned the database by identifying similar or redundant motion data in the collection [Kovar and Gleicher 2004; Beaudoin et al. 2007; Beaudoin et al. 2008; Zhao et al. 2009]. Recent works incorporate specific purposes or task objectives into consideration in addition to redundancy reduction. Cooper et al. [2007] used an active learning technique to adaptively improve the coverage of motion synthesis. Reitsma and Pollard [2007] adjusted given motion graphs to achieve a good trade-off between the graph size and the ability to navigate through specific environments. Our method automatically identifies highly compact sets of example data specifically tailored for the given user-defined task objectives or constraints. We improve the long term achievement of the task objectives, instead of simply creating a sparse coverage of various motion repertoire.

Many methods for automatically finding the right basis functions to approximate the value functions have been proposed. The proto-value functions use harmonic analysis on state transitions to capture ridges in the state connectivity and decision boundaries [Mahadevan and Maggioni 2006]. Keller et al. [2006] used neighborhood component analysis to aggregate states of similar Bellman error, and used the clusters as the basis functions. Variable resolution methods effectively adapt the local structure of value functions [Moore 1991; Munos and Moore 2002]. Munos and Moore [2002] present an octree-based hierarchical refinement process based on state influence and error variance statistics.

3 Animation with Parametric Data

This section describes the basic components of our animation framework: the parametric motion data representation and the reinforcement learning formulation. Both components are largely adopted from Treuille et al. [2007] and Lo and Zwicker [2008] with some modifications.

3.1 Parametric Motion Model

Our motion model is based on Treuille et al [2007] that uses the step-phase-based clip segmentation and foot contact annotation. This is the only partially manual processing necessary. Continuous animation is synthesized by the same process that aligns the pivot foot and blends the clips. Lo and Zwicker [2008] extended the model by weighted interpolation on the motion clips, which enabled a wider variety of animation and more precise controls with a significantly reduced amount of data. We further extend the model by introducing another parameterization method by *transformation*. While the interpolated clips produce novel motions by interpolating motion data, our transformation methods directly alter the joint configurations to create new motion. We employ computationally inexpensive methods that can be used in realtime synthesis, such as directly modifying the root's translation, orientation, or clip length. By continuously increasing the modification amount through time, we can alter the clips to turn different angles, climb steps of various heights, or change step lengths and timing.

Transformation has advantages over interpolation. First, it does not require creating clusters for similar motion clips which can be tedious manually, and error-prone through automated methods. Second, transformation is better for precise controls over multiple parameters. A clip can be transformed to satisfy many simultaneous desired changes such as step length, height, direction, and timing. Constructing interpolated clips that represent such parameters requires exponential number of clip examples. Moreover, finding the blending weights that satisfy every simultaneous constraint is even challenging and often not possible. Another advantage is the predictability of transformation. The result of transformation can be known with minimal prediction operations without actual transformation or interpolation operations. This speeds up the decision processes described in Section 3.2.

Unfortunately, the transformation may violate physical properties and create unrealistic animation. Methods that produce physically correct motions through a full-body simulation and extensive optimizations [Liu et al. 2005] are infeasible for interactive applications. Instead, we sacrifice physical correctness for efficiency by using simple transformations. However, such transformations are far more likely to introduce unpleasant distortions as the amount of warping increases and moves away from the original motion. The key idea is that reinforcement learning can be applied to intelligently adjust the degree of transformation in order to achieve both runtime efficiency and motion quality.

The motion model synthesizes continuous animation by concatenating clips in succession as in Treuille et al. [2007]. The necessary *constraint frames* information can be recomputed in any instances of parametrized clips by interpolating the constraint frame poses or applying transformation on the pose. We can handle foot-skating artifacts using a stock IK solver, although parameterized locomotion clips produce little foot-skating artifacts after blending.

A parameterized clip is specified by the clip data C and the clip parameters $\theta_{\mathbf{P}}$. For an interpolated clip, C is a cluster of clips, and $\theta_{\mathbf{P}}$ are the blending weights. For a transformed clip, C is a single motion clip, and $\theta_{\mathbf{P}}$ encode the transformation parameters.

3.2 Reinforcement Learning Formulation

We use a modified version of the reinforcement learning (RL) formulation in Treuille et al [2007] and Lo and Zwicker [2008]. The learning algorithms construct intelligent mechanisms that synthesize realtime animation for interactive user controls. Specifically, the mechanism makes decisions on which sequence of clips to concatenate, to produce a natural and effective long term behavior. In this section we describe the components of the RL formulation.

A *state* encapsulates all the information necessary to make the decisions. With non-parametric clips, states are defined as a pair

$$s = (C, \theta_{\mathbf{T}}) \tag{1}$$

of the clip C, and the current task parameters $\theta_{\mathbf{T}}$ that are unique to each task definition. The $\theta_{\mathbf{T}}$ are defined to be at the center of the clip (Figure 1(a)). With parameterized clips, this definition is insufficient, because the clip parameters $\theta_{\mathbf{P}}$ alter both the produced motion and $\theta_{\mathbf{T}}$ (Figure 1(b)). In other words, each variation $C' = W(C, \theta_{\mathbf{P}})$ by a transformation W acts as a distinct non-parametric clip. This means we need the clip parameters $\theta_{\mathbf{P}}$ in the state definition,

$$s = (C, \theta_{\mathbf{P}}, \theta_{\mathbf{T}}). \tag{2}$$

Unfortunately, reinforcement learning tasks become exponentially harder as the number of state parameters increases [Bellman 1957]. Lo and Zwicker [2008] omitted $\theta_{\mathbf{P}}$ in the state definition by crafting interpolated clusters with very similar motion clips. However, the errors are still present, and many clusters are required because each represented a relatively small variation of motion.

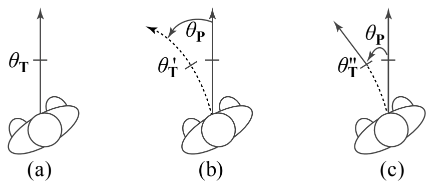

Figure 1: Parameterization by transformation. (a) *The original clip C.* (b) *The clip C transformed by parameters* $\theta_{\mathbf{P}}$. (c) *The clip C transformed by parameters* $\theta_{\mathbf{P}}$ *with transformation acceleration. In all figures, the solid line represents the original motion and the dotted line represents the modified motion.*

Parameterization by transformation allows an optional solution we call *transformation acceleration*, which accelerates the transformation to complete within the first segment (Figure 1(c)) instead of spanning over the entire clip (Figure 1(b)). The key observation is that the next clip sees only the second segment of the current transformed clip. If the second segment of the clip remains unmodified, the next decision can be made as if the entire clip was unmodified. Since $\theta_{\mathbf{P}}$ in (2) captures the modification of the clip data C, it can be ignored safely. Notice the task parameters θ''_T are still affected by $\theta_{\mathbf{P}}$, but the original state definition already includes them.

Note that the acceleration is *optional*. In cases where transformations need to span both segments such as waving hands, the full state representation in (1) can be used. Also, states represented in (1) and (2) can coexist, allowing us to use the correct representation when necessary, while using reduced state whenever possible.

An *action* represents a decision on the next motion, such as turning, changing speeds, or climbing stairs. With non-parametric clips, the action is simply the choice of the next clip. In the parametric case, an action is a pair $a = (C, \boldsymbol{\theta}_{\mathbf{P}})$ because both the clip and its parameters determine the resulting motion. The *transition function* f determines the next state s' from a state s and an action a: $s' = f(s, a)$.

A policy Π is an automatic mechanism that determines the action for any state, as in $\Pi(s) = (C, \boldsymbol{\theta}_{\mathbf{P}})$. Since this is what a *controller* is, we use the terms "controller" and "policy" interchangeably.

The goal of the RL framework is to construct controllers that achieve pre-defined tasks, as described by the *reward function*, $R(s, a)$. The learning algorithm finds the optimal policy Π^* that maximizes a discounted long term reward on every state $s = s_0$:

$$\Pi^* = \mathrm{argmax}_{\Pi} \sum_t \alpha^t R(s_t, \Pi(s_t)) \tag{3}$$

for $s_{t+1} = f(s_t, \Pi(s_t))$, the discount factor $\alpha \in [0, 1)$, and

$$\Pi(s) = \mathrm{argmax}_a (R(s, a) + \alpha V^{\Pi}(f(s, a))) \tag{4}$$

where the *value function* V^{Π} of a policy Π is defined as

$$\begin{aligned} V^{\Pi}(s) &= \sum_t \alpha^t R(s_t, \Pi(s_t)) & (5) \\ &= R(s, \Pi(s)) + \alpha V^{\Pi}(f(s, \Pi(s))). & (6) \end{aligned}$$

For continuous state parameters in $\boldsymbol{\theta}_{\mathbf{T}}$, the value function is approximated with a linear combination of *basis functions* $\Phi = \{\phi_i\}$. Letting $V^{\Pi}(s) \approx \Phi(s)\mathbf{w}$, we can rewrite (6) as,

$$\Phi(s)\mathbf{w} = R(s, \Pi(s)) + \alpha \Phi(s')\mathbf{w}. \tag{7}$$

We use the least squares policy iteration (LSPI) [Lagoudakis and Parr 2003] to solve for $\mathbf{w}$:

$$\min_{\mathbf{w}} |[\Phi(s) - \alpha\Phi(s')]\mathbf{w} - R(s, \Pi(s))|, \forall s. \tag{8}$$

We measure the *performance* Q of a controller by how well it achieves the long term reward:

$$Q(\Pi) = \sum_{s_0 \in D} \sum_t \alpha^t R(s_t, \Pi(s_t)) \tag{9}$$

where the initial state distribution D can be chosen by the user. The distribution D can span every state, or be restricted to interested regions. For example, when constructing a controller to go through revolving doors, we can specify D to be the states before the doors.

4 Motion Selection

Parametric motion clips can synthesize a wide variety of novel motions with significantly less data. The reduced data requirement translates to tangible savings in storage because a clip typically needs more storage than other components do such as value functions. The growing demand for a richer set of character behavior controllers with better runtime performance, especially on mobile gaming platforms, further motivates storage savings.

Unfortunately, it is unclear how to find a compact set of data, or clips in our setup, that produces a well-performing controller for an arbitrary task. Experienced designers usually rely on their intuition to identify relevant motion data for the character's given task. However, it is becoming less practical to use human intuition on the growing amount of motion data and even larger number of parametric transitions. This is a significant bottleneck to the content creation pipeline when each new behavior or task definition requires the entire selection process to be redone.

Systematically selecting the right set of clips is a challenging problem. In a typical motion database, there are numerous versions of similar motions, yet we have found that visually similar clips can have drastically different effects on the controller. Omission of key clips noticeably degrades the perceived intelligence and realism, so we have to judiciously pick the right clip even among the similar clips. Naively searching over all possible combinations of clips is impractical even with a modest-sized motion clip database.

In this section, we present a method to automatically identify compact sets of clips that produce high performance controllers. In order to cope with the exponential search space, we employ an iterative search process. At each iteration, we score every candidate clip according to how much it benefits the controller, and pick the one with the most desirable effects.

4.1 Motion Selection Criteria

We need a clip selection criteria to measure the benefit of using a particular clip for a controller. Since an optimal controller by definition more frequently utilizes clips that are beneficial to achieving the task, the controller's usage preference gives a good insight of which clips are considered more useful by the controller.

The concept of *influence* captures such usage preferences [Munos and Moore 2002]. The influence I of a state s' under a policy Π is,

$$I(s') = 1_{s' \in D} + \sum_{s \in B(s')} \alpha I(s) \tag{10}$$

where $B(s') = \{s | f(s, \Pi(s)) = s'\}$ and D is a user-specified initial state distribution. Informally, the influence of a state s' measures how many other states s eventually transition to s' under policy Π. A policy change at the state s' recursively influences the policy at every preceding state s, hence the term. The discount factor α ensures the immediate states have more impact on the influence than the distant states in a transition chain. We set D to match the D in the performance metric in (9) so that we get the influence on the user-interested states in D. Intuitively, a clip becomes influential when the controller decides to use the clip more than others: An influential clip is a useful clip.

On the other hand, not all useful clips are influential. For example, in a directional controller, straight clips are more influential than turning clips because the controller only uses the turning clips in a couple of steps to converge to the desired direction of movement, after which only the straight clips are used. However, the lack of responsive turning ability will significantly decrease the quality of motion and perceived naturalness. That means we need to consider the actual performance contribution of each clip to the controller.

The *marginal value contribution* of a clip C to a state s is defined,

$$\Delta V_C(s) = V_{C+}(s) - V_{C-}(s) \tag{11}$$

where $V_{C+}(s)$ is the value of state s when C is included in the controller, and $V_{C-}(s)$ is the value when C is not included.

The marginal value contribution alone as a selection criteria is also insufficient. If the clip brings drastic improvements to states that are almost never visited by the controller, such contribution will do little to enhance overall controller performance. Therefore, the most beneficial clips have high value contribution on the controller's influential states. This leads to a combined scoring metric,

$$M(C) = \sum_s I(s) \cdot \Delta V_C(s). \tag{12}$$

Notice the term $I(s) \cdot \Delta V_C(s)$ approximates the actual change of performance in the controller, because the improvement of value predicted by $\Delta V_C(s)$ propagates exactly the amount of $I(s)$ in sum to the states that lead to s. See Appendix A for how (12) and (9) are related under some assumptions.

4.2 Motion Selection Process

We formulate the motion selection process as an iterative clip *addition* process, where we start with a single clip then successively include more clips until we reach the desired clip size or controller performance. At each iteration, we need to evaluate a candidate clip

Algorithm 1 Motion Selection Process

```
Input: The reward function $R$, the initial clip $C$.
1: $\mathscr{C} \leftarrow \{C\}$
2: repeat
3:     Construct $\Pi$ from $R$ and the current $\mathscr{C}$.
4:     Update influence $I$ for $\Pi$.
5:     $C^* \leftarrow \arg\max_{C \notin \mathscr{C}} M(C)$.
6:     $\mathscr{C} \leftarrow \mathscr{C} \cup \{C^*\}$.
7: until desired $|\mathscr{C}|$ and $Q(\Pi)$ trade-off is achieved.
```

C for its additional benefit to the controller. Since $C \notin \mathscr{C}$, we have $V_{C-}(s) = V(s)$. V_{C+} is the better of the current value and the value induced by taking the optimal action a_{C+} that uses C,

$$V_{C+}(s) = \max(V(s), R(s, a_{C+}) + \alpha V_p(f(s, a_{C+})) \tag{13}$$

where V_p is a *predicted value* for the new state containing C, approximated by taking the better of the values induced by taking the optimal action into $\mathscr{C}$ and the optimal action containing C again. This approximation correctly predicts the actual change of performance with correlation coefficients ranging from 0.77 to 0.91. The initial clip can be chosen to be one that produces the best controller with a single clip, but in our experience the choice makes little impact on the final convergence.

A major advantage of this approach is the computational feasibility. The evaluation of the score metric is more efficient than the full value function construction especially for larger clip sets. For a candidate set of size C and a target number of clips N, our method requires the fast score evaluation CN times and the slow value function construction N times. This means our method scales well with respect to both C and N. On the other hand, a brute force search requires $\binom{C}{N}$ number of the expensive value function construction.

As an iterative process, our formulation lacks any optimality guarantee. A smaller set of clips could achieve better performance, or the selection process could potentially fail completely when a task requires long elaborate sequences of motion clips. The influence-based scoring metrics implicitly assumes a static current policy, even though the optimal policy could be substantially different after an addition. Nevertheless, our evaluations show a remarkable convergence to the global optimum after just a few iterations.

Alternatively, we can start with all candidate clips included, then iteratively remove the *least* scoring clips. We can set for a clip C,

$$V_{C-}(s) = \begin{cases} R(s, a_{C-}) + \alpha V(f(s, a_{C-})), & \text{if } \Pi(s) \text{ contains } C \\ V(s), & \text{otherwise} \end{cases} \tag{14}$$

for the optimal action a_{C-} *not* containing C. However, this method is less viable because it requires a huge initial value function with all the clips. Also the selection process needs much more iterations because typically a fraction of candidate clips performs almost as well as the entire set.

4.3 Applications

The motion clip selection extends beyond single controller cases. It is possible to formulate similar selection methods for a group of controllers to improve collective performance.

Controllers with Separable Parameters. For some tasks, the task parameters $\theta_\mathbf{T}$ contain *separable* parameters $\theta_\mathbf{T}^\mathbf{s}$. Treuille et al. [2007] shows that *partial policies* Π_i using specific settings of $\theta_\mathbf{T}^\mathbf{s}$ can be separately constructed and then combined to form the full policy that covers the entire parameter space. The separated partial policies are much easier to construct, so we can build higher dimensional controllers using the partial policies as building blocks.

We are interested in the motion selection process that benefits the full policy Π. Since the reward functions are identical in every Π_i, the scores M^{Π_i} are directly comparable. Also we maintain the identical set of clips in all Π_i through the selection process. Therefore we can define the aggregate scoring metric to be the sum of the scores of individual partial policies.

$$M_{agg}(C) = \sum_{\Pi_i} M^{\Pi_i}(C) = \sum_{\Pi_i} \sum_s I(s) \cdot \Delta V_C^{\Pi_i}(s) \tag{15}$$

Transitions for Switching Controllers. Switching is also possible between any controllers with different set of clips, because the optimal action depends only on the next controller's reward and value functions (see Equation (4)) and the motion model admits valid transition between any clips [Treuille et al. 2007]. This means we can add new controllers to the framework with no modification to existing controllers. A rich library of modular behaviors can be built by simply adding independently created controllers.

However, switching between controllers does not always produce visually natural transitions. For example, a walking controller and a running controller with no clips for speed adjustments would produce abrupt unnatural speed changes while switching. The compact controllers with small specialized sets of clips from the selection process only exacerbate the issue.

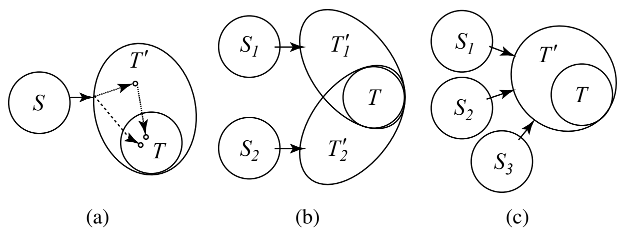

Figure 2: Transition controllers. (a) *A transition controller T' mediates switching from S to T by finding better paths that lead to T.* (b) *Specialized transition controllers can be built for each switching scenario.* (c) *A single transition controller can be optimized for multiple transition scenarios simultaneously.*

We can apply the clip selection process to find natural transitional clips for switching. In order to keep the existing source and target controllers unmodified, we introduce a *transitional controller* that incorporates the newly selected clips into the switching process, as described in Figure 2(a). When switching from the source controller S to the target controller T, the transitional controller T'

provides an alternate transition route using the transitional clips not included in T. We construct T' with the reward function of T, so the alternate route is an optimal path for the target task. Also by fixing the value function of T during construction of T' the policy T can remain unmodified.

We define the scoring metric to measure the benefit of a given clip to the entire switching process using the transitional controller.

$$M_{tran}(C) = \sum_{s \in D^S} I^S(s) \cdot (V_{C+}^{T'}(s) - V_{C-}^{T}(s)) \tag{16}$$

for the *source* controller's state distribution D^S, the *source* controller's influence I^S , the predicted value $V_{C+}^{T'}$ of the *transition* controller, and the value function $V_{C-}^{T} = V^T$ of the *target* controller.

A transition controller is built for each controller pair, therefore can be highly specialized and modular (See Figure 2(b)). Alternatively, the selection metric can consider clips that benefits *all* switching transitions by summing every transition's score (See Figure 2(c)). This can further reduce the overall number of clips at the cost of specialization of each transitional controller.

The automation of transitional controller synthesis enables a designer to concentrate on crafting novel individual controllers without concerns for connections with existing controllers.

5 Basis Refinement

A controller's ability to make optimal decisions relies directly on the correctness of the value functions, which are in turn approximated by a set of basis functions. Therefore the basis functions must have enough representational power to approximate the value functions especially for complex tasks that have complicated value functions. Each value function has a different set of basis functions that can approximate it well. Naively using all possible basis functions is clearly infeasible.

A common approach is to adapt a set of basis functions until they provide enough representational power. Munos and Moore [2002] identified and iteratively improved regions where basis functions need more power. This refinement process effectively produced solutions for high dimensional problems. In this section we present the refinement process and how we incorporate it in our setup.

We need basis functions that allow high degree of localized modifications for the refinement process. To that end, we employ *piecewise constant basis functions* Φ that are a collection of functions $\phi_{B_i} = 1_{s \in B_i}$ for a boxed region $l \leq B_i < u$ for various l, u. The supports B_i are mutually exclusive and exhaustive in the parameter space. Each boxed region, or a cell, can be split to locally increase the resolution of the piecewise constant basis functions and provide additional representational power. Figure 6 shows a splitting example. From now on, we simply denote $\phi_i = \phi_{B_i}$.

The approximation by basis functions inevitably produces inaccuracies called the *Bellman error*,

$$e(s) = [R(s, \Pi(s)) + \alpha V(s')] - V(s) \tag{17}$$

which is the disagreement between the approximated value at the current state and the one-step look ahead value. An important observation is that the Bellman error should be zero everywhere for a correct value function (see Equation (6)). In fact, nonexistence of the Bellman error is a sufficient condition for obtaining the *optimal* value function [Bellman 1957]. To identify the regions with policy degradation due to Bellman error, Munos and Moore [2002] introduce the concept of *variance* σ^2,

$$\sigma^2(s) = \alpha^2 \sigma^2(s') + e^2(s) \tag{18}$$

for $s' = f(s, \Pi(s))$. In essence, the variance measures an aggregate approximation error including the state's own Bellman error as well as the discounted approximation error propagated from the future states. Since the errors lead the current state to suboptimal actions, states with high variance are good candidates for the refinement effort. The influence is useful for measuring the scope that the state's error potentially propagates to. The combined scoring metric

$$M(\phi_i) = \sum_s \phi_i(s) I(s) \sigma(s) \tag{19}$$

therefore identifies the cells that cause large overall propagated errors in the entire value function.

6 Results

We demonstrate the effectiveness of our methods creating compact controllers on several locomotion tasks. We captured the motion data by freely performing given locomotion tasks without specific instructions other than to try various turns and speeds at will. The motion data is captured at 120Hz using a Vicon system. Each clip is about 70 to 120 frames long, and takes about 60KB of storage.

For motion selection experiments, we arbitrarily picked the set of candidate clips using rough tags such as 'walk straight', 'sharp turn', or 'ascend stairs'. We limited the size of the candidate set to 100 to make the comparison with human manual selection process feasible in a reasonable amount of time. We note that the motion selection scales well (linearly) with the number of clips, so we can easily use our entire database of more than 3000 clips.

6.1 Motion Selection for Single Compact Controller

Parameterization by transformation provides variations to example motions so we can create a walking controller with a single clip. However, the resulting animation has lower quality due to large amount of transformation. On the other hand, a walking controller using 88 clips produced natural and responsive animation.

We used the motion selection process to find a set of only 5 clips that produce visually indistinguishable animation with the 88-clip controller. The value function was stored in less than 1KB.

6.2 Motion Selection for Separable Controllers

We applied the motion selection on a stairs navigation controller. We captured the motion data for this example on multiple set of stairs with varying heights. The motion capture subject freely walked around for some time. Parameterization by transformation enables navigation on stairs with different tread heights and widths.

The task parameters are defined as $\theta_{\mathbf{T}} = (\theta_c, \theta_d, d_s, w_s, h_s)$ where θ_c is the orientation of the character, θ_d is the desired direction of movement, d_s is the distance from the next tread, w_s is the width of a tread, and h_s is the relative height of the next tread (Figure 3). Notice that θ_d, w_s and h_s are separable parameters. The clip transformation has three parameters $\theta_{\mathbf{P}} = (\tau, \mu, h)$ where $\tau \in (-0.2\pi, 0.2\pi)$ is the amount of directional change and $\mu \in (0.8, 1.2)$ is the ratio of adjusted step length with respect to original motion clip. The step height adjustment h is determined by the next step location. The reward function is defined as,

$$R = \Psi - \omega_d |\rho - \theta_d| - \omega_F F \tag{20}$$

where Ψ is the naturalness of the transition, θ_d is the desired direction, ρ is the actual movement direction, F is the foot collision penalty, and ω_d and ω_F are weighting coefficients.

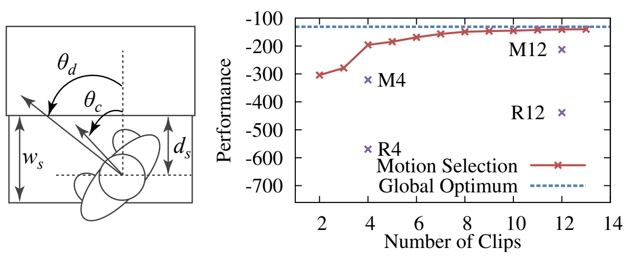

Figure 3: ***Left:*** *Stairs Task Parameters.* ***Right:*** *Performance improvement by motion selection.*

We applied the motion selection algorithm on the 8 partial controllers with separable parameter θ_d spanning $[-\pi, \pi)$, with w_s and h_s fixed. The performance improvement after each iteration is plotted in Figure 3. The improvement occurs early in the first few iterations of motion selection and quickly approaches the *global optimum* performance produced with all 100 candidate clips. We set D for the performance measure to be the entire state space.

For comparison, we asked an animation researcher to select a set of clips from the candidates. In Figure 3, M4 and M12 represent the best performance achieved by the researcher in 30 minutes using 4 clips and 12 clips respectively. We also ran a naive random search over combinations of clips for the same amount of time with the selection method. R4 and R12 represent the best performance found by random trials using 4 clips and 12 clips respectively.

The result shows our selection method outperforms both human and random selection by producing a better controller with only 4 motions than the manual controller with three times as many motion clips (R12). Random selection with limited time significantly underperforms both methods. We believe the manual motion selection on separable controllers is difficult because one needs to consider possible benefits to every partial controller simultaneously.

We believe our method outperforms the manual selection because the manual process involves inspecting an overwhelming number of possible parametric transitions between clips. In addition, it is difficult for humans to predict the overall contribution of a clip from isolated inspections. Due to these difficulties, the users tend to lean towards simply picking natural transitions in a few isolated cases.

6.3 Motion Selection for Transition Controllers

We applied our motion selection to generate transition controllers between controllers that walk forward, walk backwards, jog and jump over ditches. Each controller is generated through the motion selection algorithm. The walking and running controllers have the reward function

$$R = \Psi - \omega_d|\rho - \theta_d| - \omega_\tau|\tau - \tau_d| - \omega_v|v - v_d| \tag{21}$$

where τ, τ_d are actual and desired torso orientations, v, v_d are actual and desired movement speed, and ω_τ, ω_v are coefficients. The jumping-over-ditch controller uses the reward function

$$R = \Psi - \omega_d|\rho - \theta_d| - \omega_J J \tag{22}$$

for the desired direction θ_d fixed perpendicular to the ditch and the successful jump reward J. The jumping controller initially contains four jumping clips only, so the transitional controllers are crucial.

We construct transitional controllers for all possible 12 pairs of controllers. We used D to be the entire state space except for the jumping controller where D is restricted to before the ditch. Figures 4(a)-(d) show their performance improvements relative to the global optimum that uses all the given candidate motions. In general, our algorithm converges to the global optimum very quickly. Figure 4(e) shows the performance improvement for walking backwards to jumping controller transitions with our algorithm, compared with the manually selected clips by a motion expert who spent approximately 8 hours to beat our method. Our method significantly outperforms the manual selection: with only 2 clips, it performs better than the best 11 manually picked clips.

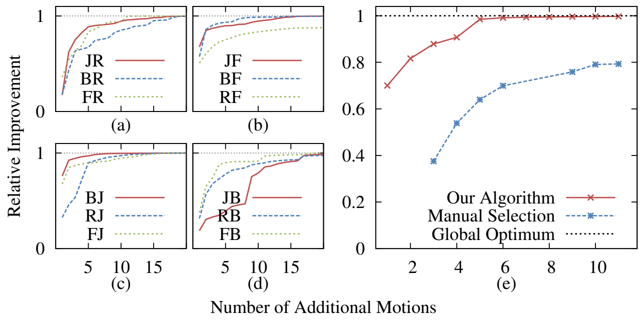

Figure 4: (a)-(d) *Performance improvements for each pair of controllers, relative to the global optimum. Walking forward, walking backwards, jogging, and jumping-over-ditches controllers are abbreviated to F, B, R, and J, respectively.* (e) *Comparison with manual selection on the BJ transition controller.*

6.4 Basis Refinement

We applied the basis refinement method to create a controller that can navigate through a checkerboard with varying tile sizes. The goal is to follow the desired direction, stepping only on the white tiles. The task parameters are defined as $\boldsymbol{\theta_T} = (\theta_c, \theta_d, l_s, p_x, p_z)$ for the orientation of the character θ_c, the desired direction θ_d, the length of the square l_s, and the relative position of the character to the checkerboard (p_x, p_z). Here θ_d and l_s are separable parameters. We use the same clip transformation as the stairs controllers, with the step height change h fixed at 0. The reward function is defined

$$R = \Psi - \omega_d|\rho - \theta_d| - \omega_T T \tag{23}$$

with the black tile step penalty T and coefficients ω_d, ω_T.

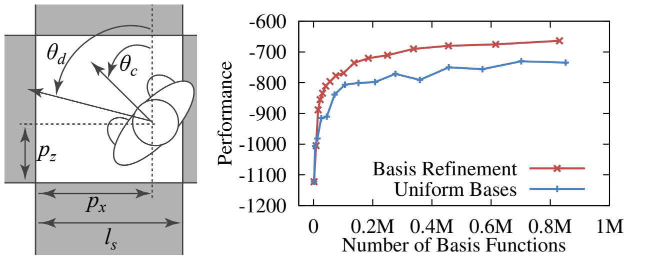

Figure 5: ***Left:*** *Checkerboard task Parameters.* ***Right:*** *Performance improvement by basis refinement.*

Figure 5 shows the performance of our iterative basis refinement compared to the one using uniform piecewise constant basis functions. Refined bases clearly produce better policy than uniform bases: the performance with one million uniform bases is equivalent to the one with our 0.13 million refined bases. Figure 6 shows the successive refinement results.

Our octree keeps the parent, children and its value in each cell. The revolving door example used up to 1 million cells, or 6MB.

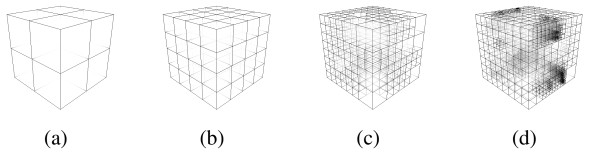

Figure 6: Basis refinement iterations. *The axes represent the orientation θ_d and the relative position p_x, p_z of the character.* (a) *Before refinement.* (b) *Iteration 1.* (c) *Iteration 2.* (d) *Iteration 6.*

6.5 Combination

The motion selection and the basis refinement methods can be combined to create a highly complex controller that can navigate through a set of revolving doors spinning at constant velocity. The task parameters $\theta_{\mathbf{T}} = (\theta_r, d_r, \theta_c, t_r, w_r, s_d, n_d)$ include the direction θ_r and the distance d_r from the door , the relative character orientation θ_c, the timing t_r, the width w_r and the speed s_d of the door, and the number of doors n_d. Here w_r, s_d, n_d are separable, but $\theta_r, d_r, \theta_c, t_r$ form a single high dimensional control problem. We use the same clip transformation as the checkerboard controller. The reward function is defined as

$$R = \Psi + \omega_d |\rho - \theta_d| + \omega_C C \tag{24}$$

where $C(s,a)$ is the collision penalty for any body part against doors or walls, and ω_d, ω_C are coefficients.

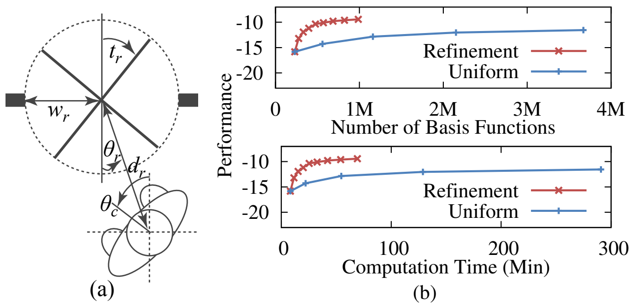

Figure 7: (a) *Revolving doors task parameters.* (b) *Performance improvement comparison by basis refinement. Our method outperforms uniform basis functions with significantly fewer basis functions and computation time. In both graphs, we used the 7 motion clips that our motion selection method produced.*

We started with a single-clip controller using coarse uniform piecewise constant bases, and applied basis refinement algorithm and motion selection algorithm iteratively. We split 20% of the bases at each refinement step. On an Intel Xeon 2.33GHz machine with 8GB RAM, creating a controller with seven clips from 63 candidates took about 11 hours in our unoptimized C# implementation in the release mode. Motion selection, basis refinement, and value function construction took 79%, 8%, and 13% of the precomputation time, respectively.

Figure 7(b) compares the performance of our refined bases and one of uniform piecewise constant bases given an identical set of motions. Refined bases require less storage and computation time to achieve the same performance as that of the uniform piecewise constant bases: the performance of 3.8M uniform bases is almost same as the one of 0.4M refined bases, while the computation time of the former takes about 15 times more than the latter does.

We expedited the computation by caching. Because the basis refinement keeps a huge portion of the bases from the last step, it can reuse previously computed transitions and rewards. In Figure 7(b), it takes about 70 minutes to compute a value function with a million bases with eight refinement processes, while it takes about 50 minutes to compute a value function with 1.2 million uniform bases. Considering that the former constructed nine value functions, it is a significant increase in speed.

7 Conclusion and Future Work

This paper presents methods for constructing compact controllers with significantly reduced data requirement and improved performance. The motion selection algorithm can select a compact set of clips that produces high performance controllers. Our method consistently outperforms expert manual selections and approaches a global optimum in just a few iterations. We extend the method to automatically create high quality transition controllers. This enables creating a rich library of behaviors with completely modular controllers as building blocks. The basis refinement method selectively enhances the power of the value function near critical decision boundaries, while sparing resources in less critical regions. The refinement can adapt very coarse initial basis functions to create effective controllers for highly complex tasks. These methods enable a five-dimensional (one discrete, four continuous) revolving doors controller which would be infeasible with known alternatives.

Our selection and refinement methods apply naturally to our parametric motion model, but also to any motion representation where a Markov decision process (MDP) can be defined. For example, on the original motion graph, an MDP can be defined by states at each branching point of the graph. The motion selection would be choosing which edge to admit. Interpolated or parametrized motion graph structures are all similarly applicable. Application on the modular dynamic step controllers [Muico et al. 2009] should be an interesting step towards compactly representing a dynamic human motion mechanism.

A major limitation of our motion selection is the lack of theoretical guarantee of optimality. Each selection iteration greedily picks the single best contributing clip, instead of considering collaborative effect of several new clips. Thus it can fail to identify a long specific sequence of clips typically required for more deliberate tasks. Still, for locomotion tasks in our experiments we obtained a consistent convergence to the global optimum.

Another limitation is that our selection process cannot synthesize novel clips to use. Instead, the algorithm does its best with the existing clips. If no improvement is possible with existing clips, the user has to provide more relevant data. It will be very interesting to start from only a description of the task and progressively build the most effective motion repertoire.

The basis refinement depends on an octree-based representation that requires exponential storage space. This is currently the fundamental limiting factor on the complexity of achievable tasks. A storage-efficient spatial partitioning structure, such as linkless octrees [Choi et al. 2009] can be beneficial in the near term. In the long term, more effective methods to model high dimensional decision processes will enable more delicate and complex behaviors.

We believe our work enables interesting applications. Automatic selection of clips and bases brings the entire process of controller authoring closer to a level where novices can author complex realistic controllers. Simply by choosing a few task objectives, one can generate a specialized compact task controller and transition controllers to other existing task controllers. Our hope is that game players and virtual world participants will be able to author not just their appearance, but also their behaviors, and enable avatars to learn new skills by extending the existing behaviors with new controllers that can deal with new environments.

With the ability to create large interconnectable collection of controllers, we can envision planning techniques with the controllers as the building blocks. This higher-level *meta-controller* finds an optimal sequence of controllers that achieves its high level objectives. For example, when the character is thirsty, a standing up controller, a door opening controller, a walking down the stairs controller, and a drink from a water fountain controller can be sequentially activated. A meta-controller can potentially plan very efficiently by delegating the responsibilities for motion quality and local task achievement to specific task controllers. This should enable the character to navigate complex scenes that are even changing dynamically, with the same motion quality provided by the controllers. The motion selection can be extended as a controller selection method, where we pick essential controllers for a meta-task.

Acknowledgments.

We thank Erik Anderson and Robert Forsberg for help with videos, and the anonymous reviewers for their helpful comments. This work was supported by the UW Animation Research Labs, NSF grant HCC-0811902, Intel, Samsung, and Microsoft Research.

A Controller Performance Prediction

This section shows how (12) is related to actual performance changes in (9). Assume the new clip changes the policy only at a single state s, and the effects of cyclic transitions can be ignored. Note the influence can be rewritten as,

$$I(s) = 1 + \sum_{k=1} \alpha^k \left| B_D^k(s) \right| \tag{25}$$

where $B_D^k(s)$ is the intersection of D and the set of states that transition to s in k steps. The overall propagated performance change $q_C(s)$ by the new clip C on the state s is,

$$\begin{aligned} q_C(s) &= \Delta V_C(s) + \sum_{s' \in B(s)} \alpha q_C(s') && (26) \\ &= \Delta V_C(s) + \sum_{k=1} \left(\sum_{s' \in B_D^k(s)} \alpha^k \Delta V_C(s) \right) && (27) \\ &= \Delta V_C(s) \cdot (1 + \sum_{k=1} \alpha^k \left| B_D^k(s) \right|) && (28) \\ &= \Delta V_C(s) \cdot I(s) && (29) \end{aligned}$$

Now the overall performance change can be approximated by summing individual performance changes at every state,

$$\Delta Q(\Pi) \approx \sum_s q_C(s) = \sum_s \Delta V_C(s) \cdot I(s) \triangleq M(C). \tag{30}$$

References

BEAUDOIN, P., VAN DE PANNE, M., AND POULIN, P. 2007. Automatic construction of compact motion graphs. Tech. Rep. 1296, Universite de Montreal, May. DIRO.

BEAUDOIN, P., VAN DE PANNE, M., POULIN, P., AND COROS, S. 2008. Motion-motif graphs. In *Symposium on Computer Animation 2008*, ACM.

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

CHOI, M. G., JU, E., CHANG, J., KIM, Y. J., AND LEE, J. 2009. Linkless octree using multi-level perfect hashing. *Pacific Graphics 2009*.

COOPER, S., HERTZMANN, A., AND POPOVIĆ, Z. 2007. Active learning for real-time motion controllers. *ACM Transactions on Graphics 26*, 3 (July), 5.

IKEMOTO, L., ARIKAN, O., AND FORSYTH, D. 2005. Learning to move autonomously in a hostile environment. Tech. Rep. UCB/CSD-5-1395, University of California at Berkeley, June.

KELLER, P. W., MANNOR, S., AND PRECUP, D. 2006. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, ACM, New York, NY, USA, 449–456.

KOVAR, L., AND GLEICHER, M. 2004. Automated extraction and parameterization of motions in large data sets. *ACM Transactions on Graphics 23*, 3.

LAGOUDAKIS, M. G., AND PARR, R. 2003. Least-squares policy iteration. *Journal of Machine Learning Research 4*, 1107–1149.

LAMOURET, A., AND VAN DE PANNE, M. 1996. Motion synthesis by example. In *In EGCAS 96: Seventh International Workshop on Computer Animation and Simulation, Eurographics*, 199–212.

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*, 299–308.

LEE, J., AND LEE, K. H. 2004. Precomputing avatar behavior from human motion data. In *Proceedings of the 2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, ACM Press, 79–87.

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

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

MAHADEVAN, S., AND MAGGIONI, M. 2006. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. Tech. Rep. TR-2006-36, University of Massachusetts, Department of Computer Science.

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

MOORE, A. 1991. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces. In *Machine Learning: Proceedings of the Eighth International Conference*, L. Birnbaum and G. Collins, Eds.

MUICO, U., LEE, Y., POPOVIĆ, J., AND POPOVIĆ, Z. 2009. Contact-aware nonlinear control of dynamic characters. *ACM Transactions on Graphics 28*, 3.

MUNOS, R., AND MOORE, A. 2002. Variable resolution discretization in optimal control. *Machine Learning 49*, 2-3, 291–323.

REITSMA, P., AND POLLARD, N. 2007. Evaluating motion graphs for character animation. *ACM Transactions on Graphics 26*, 4 (Oct.), 18.

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

ZHAO, L., NORMOYLE, A., KHANNA, S., AND SAFONOVA, A. 2009. Automatic construction of a minimum size motion graph. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation.*