Modeling Spatial and Temporal Variation in Motion Data

Manfred Lau Ziv Bar-Joseph James Kuffner
Carnegie Mellon University

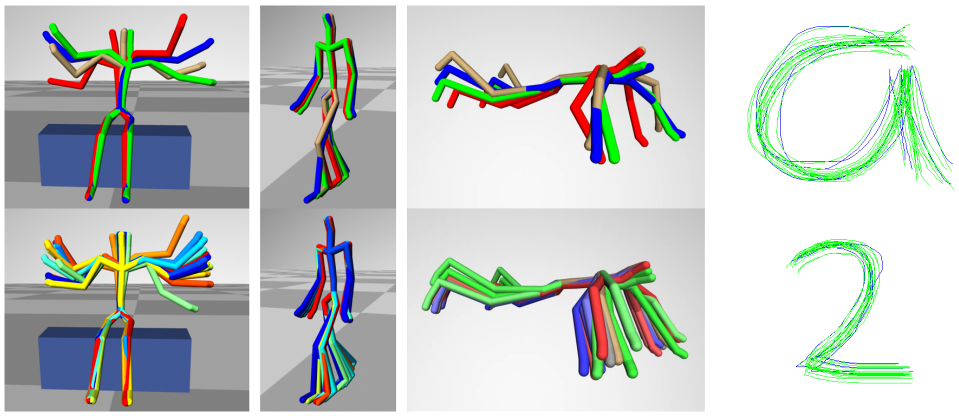

Figure 1: *First 3 columns: Results for cheering, walk cycle, and swimming motion. In each column, the top image shows the 4 inputs (overlapped, each with different color) and the bottom image shows the 15 outputs (overlapped, each with different color). These are frames from the animations. Please see the animations in the video. Last column: Results for 2D handwritten characters "a" and "2". Each image shows both the 4 inputs (blue) and 15 outputs (green).*

Abstract

We present a novel method to model and synthesize variation in motion data. Given a few examples of a particular type of motion as input, we learn a generative model that is able to synthesize a family of spatial and temporal variants that are statistically similar to the input examples. The new variants retain the features of the original examples, but are *not exact copies* of them. We learn a Dynamic Bayesian Network model from the input examples that enables us to capture properties of conditional independence in the data, and model it using a multivariate probability distribution. We present results for a variety of human motion, and 2D handwritten characters. We perform a user study to show that our new variants are less repetitive than typical game and crowd simulation approaches of re-playing a small number of existing motion clips. Our technique can synthesize new variants efficiently and has a small memory requirement.

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

Keywords: Human Animation, Motion Capture, Variation, Machine Learning

1 Introduction

Variation in human motion exists because people do not perform actions in precisely the same manner every time. Even if a person intends to perform the same action more than once, each motion will still be slightly different. However, current animation systems lack the ability to realistically produce these subtle variations. For example, typical crowd animation systems [McDonnell et al. 2006] utilize a few walking motion clips for every walk cycle and every character of the simulation. This can lead to synthesized motion that look unrealistic due to the *exact repetition* of the original walk cycles. Hence a variation model that can generate even slight differences of the original walk cycles has the potential to greatly improve the naturalness of the output motion. Crowds in games and films [Thalmann et al. 2005] also do not produce human-like variations. Animators for films and games often use cycle animation, where a fixed number of motion cycles are used to create the motions for multiple characters. Inevitably, there will be cycles that are *exactly repeated* both spatially (in multiple characters) and temporally (at different times for the same character). As soon as an example repetition is identified, the whole animation can immediately deemed to be unnatural. The methods in this paper can be applied to such animation scenarios to make them more compelling.

Previous methods [Perlin 1995; Bodenheimer et al. 1999] consider variation to be an additive noise component. However, these methods are not robust for automatically generating animations because there is no guarantee that the added noise will match well with the existing motion. In addition, recent biomechanical research [Harris and Wolpert 1998] has argued that variation is *not just noise or error*, but is *a functional component of motion*. From this point of view, adding random noise to existing motion is not a principled approach.

We take a data-driven approach to the problem of modeling and synthesizing variation. Given a small number of examples of a particular type of motion (ie. cheering, walk cycle, swimming breast stroke) as input, we learn a model from the input data, and use this model to synthesize spatial and temporal variants of that motion. We demonstrate that the Dynamic Bayesian Network (DBN)

[Friedman et al. 1998; Ghahramani 1998] model solves this problem well as it provides a formal and robust approach to model the distribution of the data. A DBN represents a multivariate probability distribution of the degrees-of-freedom of motion, and it is from this distribution that we sample to synthesize our new variants. An important feature of our approach is that it can work with a small number of input examples. This is useful as it is difficult to acquire a large number of examples of a particular motion. Another advantage is that no post-process smoothing operation is needed, which is beneficial as such an operation may smooth out details of motion that our method generates. There are three major steps for learning a model and synthesizing new variants. First, we learn the structure of the DBN with the input examples. We use a greedy algorithm based on a variant of the Bayesian Information Criterion score to learn a good structure. Second, we use the learned structure and the original data to synthesize new variants. Third and optionally, we can use an inverse kinematics method developed in conjunction with our DBN framework to satisfy any foot and hand constraints.

The key result of our method is that we can take a few examples of a particular type of motion as input, and produce an unlimited number of spatial and temporal variants as output. A new variant is spatially different as all new poses are distinct from those of the input examples, and temporally different as the timing of the whole motion is distinct from the input examples. The new variants are *statistically and visually similar* to the inputs, but are *not exact copies*. We demonstrate our approach with a variety of full-body human motion data, and 2D handwritten characters. The memory requirement of our model consists of the space required to store the few input examples and the learned DBN structure. Most of the processing time is in the learning phase; the runtime for synthesizing new variants is efficient and can be done as a continuous stream one frame at a time.

To evaluate our approach, we perform a user study to show that: (i) our new variants are just as natural as motion capture data, and (ii) our new variants are less repetitive than "Cycle Animation". In addition, we demonstrate that "just adding noise" to existing motion can create poses and timings that look obviously awkward. We show this with two methods to add noise to motion: (i) a naive/strawman method, and (ii) the Perlin noise function. Finally, since our input examples have to be similar (so that we can model their variation), it is useful to know what we mean by "similar" and how we get them to begin with. Hence we provide a DBN-based and data-driven method to select "similar" examples that can be used well with our approach.

2 Background

One previous approach for generating variation in motion is to add noise. Perlin [1995] adds noise functions to procedural motion to create more realistic animations of running, standing, and dancing. Bodenheimer and his colleagues [1999] adds noise to cyclic running motion. The noise is added only to the upper body, and is synchronized with the arm swings in the running cycle. Adding noise, however, provides no guarantee that the noise will match well with the existing motion. Adding arbitrary noise can lead to artifacts in the motion, while adding noise with tuned distributions requires a trial and error process of manual parameter tuning. In our approach, the variations that we generate come from the data and not from a separate additive component.

Pullen and Bregler's work [2000; 2002] for generating motion variations is most closely related to our work. They model the correlations between the DOFs in the data with a distribution, and synthesize new motion by sampling from this distribution and smoothing the motion. However, important aspects of the correlations are manually defined. For example, they specify manually that the hip angle affects the knee, and the knee angle affects the ankle. The structure learning in our DBN framework learns these joint relationships automatically from data. In addition, they use their method to animate a 2-dimensional 5-DOF wallaby figure, and a more complex 3D character. We demonstrate results of different kinds of motion for a full-body human figure, and for 2D handwritten characters. Furthermore, our approach requires no smoothing, which is a significant advantage as sampling-and-smoothing methods will smooth out certain details of the motion. In comparison to Pullen and Bregler's work, our approach is more automatic, general, and robust. Li and his colleagues [2002] also generate new motion that is statistically similar to the original data. They use 20 minutes of dancing motion as training data. If a large amount of data is available, it is possible to use random resequencing of motion clips without being able to detect repetition in the motion. One of the strengths of our work is that our approach can handle a small amount of original data. Chenney and Forsyth [2000] also generates a space of plausible solutions and sample from it to generate different motion, although their focus is not on human motion.

There has been work on learning the style of motion from training data [Brand and Hertzmann 2000] and transferring the style between motions [Hsu et al. 2005]. Style and variation differ in the following way: a happy walk and a sad walk are different styles of walking, while two "similar" happy walks are different variations of "happy walks". Interpolation methods [Rose et al. 1998; Wiley and Hahn 1997] have been developed to generate a spectrum of new motions that are interpolated from the original data. Interpolation and variation are also different approaches: we interpolate a five-foot jump and a ten-foot jump to get an eight-foot jump, while we take two "similar" five-foot jumps to generate variations of "five-foot jumps".

Bayesian Networks (BNs) have been used in the animation community for solving different problems, in contrast to the variation problem that our paper solves. BNs were used to model the motion of virtual humans [Yu and Terzopoulos 2007], where the variables in their network correspond to high-level behaviors. Kwon and his colleagues [2008] use DBNs for the problem of animating the interactions between two human-like characters. Ikemoto and her colleagues [2009] use similar types of probabilistic methods for the problem of motion editing.

There are existing motion models that are related to our DBN model; in general, our work focuses on the variation problem. Chai and his colleagues [2007] focus on generating motion with a low-dimensional control input. They use an m-order linear time-invariant system. In our work, a fully connected DBN of Markov order m with appropriate conditional probability settings is a linear time-invariant dynamic system. Indeed, we used such settings in our initial attempts to construct a DBN. As this method did not work well, we eventually decided to use a non-parametric approach. The ST-Isomap model [Jenkins and Matarić 2004] focuses on reducing the dimensionality of the data and finding manifolds of the data. While these manifolds represent spatial and temporal relationships, it is not clear whether or not they can be used to generate *natural* variants. This is an issue that we focused on as we designed our method. Wang and his colleagues [2008] models manifolds of motion from data and sample from them to generate new motion. In our initial attempts, we also tried a similar approach of modeling the data with a linear combination of functions (radial basis functions in our case), but this did not work well. We therefore use and advocate a non-parametric approach, as we found that it is more robust than parametric approaches.

There is much interest in the problem of adding variety to virtual crowds. Maim and his colleagues [2008] take a fixed number of template character meshes, and vary them by changing their color and adding different accessories to them. On the other hand, our work takes a fixed number of template *motions* and synthesize new variant motions from them. McDonnell and her colleagues [2008] perform user experiments to study the perception of clones in virtual crowds. They assume that the motion of crowds of characters must be cloned. In contrast, our approach creates motion with *no exact clones*, even though the new variants are visually similar.

3 Overview

We start with a description of a DBN model (Section 4). Given a small number of input motion clips that represent variations of a type of motion, we learn the structure of a DBN model automatically (Section 5). We use a nonparametric regression approach to compute the probability distributions, and we justify this approach in Section 5.2. This is a notable difference between our application of DBN and the common use of DBNs in the literature. The learned model and data can then be used to generate any number of spatial and temporal variants of that motion (Section 6). We develop an inverse kinematics framework that is compatible with our DBN model to satisfy foot and hand constraints (Section 7). For evaluation (Section 8), we describe: (i) results for a variety of full-body human motion and 2D handwritten characters; (ii) a user study to show that our new variants are natural compared to motion capture data, and that our new variants are less repetitive than "Cycle Animation"; and (iii) two methods (strawman and Perlin) for adding noise to existing motion that can lead to unnatural animation. Section 9 describes the limitations of our approach, and introduces a DBN-based method to characterize how "similar" the inputs have to be in order to work well with our approach.

4 Dynamic Bayesian Network

We first describe the basic formulation and notations for a Bayesian Network (BN) model, and then extend this description to a Dynamic Bayesian Network (DBN) model [Friedman et al. 1998; Ghahramani 1998].

A BN is a directed acyclic graph that represents a joint probability distribution over a set of random variables $\mathbf{X} = \{X_1, ..., X_n\}$. Each node of the graph represents a random variable. The edges represent the dependency relationship between the variables. A node X_i is independent of its non-descendants given its parent nodes $\mathbf{Pa}(X_i)$ in the graph. This conditional independency is significant because we only use the values of parent nodes of X_i to predict the value of each X_i. This graph defines a joint probability distribution over $\mathbf{X}$ as follows:

$$P(X_1, ..., X_n) = \prod_i P(X_i \mid \mathbf{Pa}(X_i)) \tag{1}$$

Most BNs and DBNs that treat X_i as a continuous variable use a linear regression model [Neapolitan 2003]. However, we found that linear or non-linear parametric models did not work well for our problem domain. We speculate this may be because the amount of data is not large enough. Hence we compute $P(X_i \mid \mathbf{Pa}(X_i))$ using a non-parametric regression approach, which we found to work well for our motion data.

A DBN models the process of how a set of random variables change over time. It represents a joint probability distribution over all possible trajectories of the random variables. Figure 2 shows an example. In our case of human motion, X_i is the trajectory of values of the i^{th}-DOF of motion, and $\mathbf{X}[t]$ is the set of values of all the DOFs at time t. $X_i[t]$ is the value of the i^{th}-DOF at time t. The prior

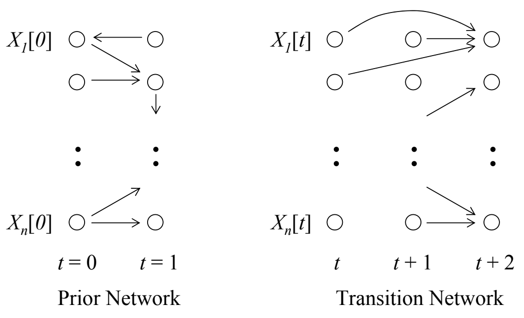

Figure 2: *A DBN for the variables $X_1, ..., X_n$. Each node X_i represents one DOF in the motion data. We use the prior network to model the first 2 frames. The transition network models subsequent frames given the previous 2 frames. We assume a 2nd-order Markov property because it is the simplest model that works well.*

network G_{prior} represents the joint distribution of the nodes in the first two time steps, $\mathbf{X}[0]$ and $\mathbf{X}[1]$. The transition network G_{trans} specifies the transition probability $P(\mathbf{X}[t+2] \mid \mathbf{X}[t], \mathbf{X}[t+1])$ for all t. Note that the transition network predicts the values at time $t+2$ given those at t and $t+1$. Hence there are no incoming edges into the nodes at time t and $t+1$. We assume that the trajectories satisfy the second order Markov property: the values at time t and $t+1$ can be used to predict those at $t+2$. We found that assuming a first order Markov property does not work well for our motion data, and we justify our second order assumption in Section 8. We also assume that the transition probabilities are stationary: the probabilities in G_{trans} are independent of t. The DBN defines a joint probability distribution over $\mathbf{X}[0], ..., \mathbf{X}[T]$:

$$P(\mathbf{X}[0], ..., \mathbf{X}[T]) = \\ P_{G_{prior}}(\mathbf{X}[0], \mathbf{X}[1]) \cdot \prod_{t=0}^{T-2} P_{G_{trans}}(\mathbf{X}[t+2] \mid \mathbf{X}[t], \mathbf{X}[t+1]) \tag{2}$$

Similarly, we apply a non-parametric approach to predict $\mathbf{X}[t+2]$ given $\mathbf{X}[t]$ and $\mathbf{X}[t+1]$. Hence we do not have parameters and we only learn the dependency structure from the data. The data itself implicitly defines the function in a non-parametric approach. Note that our non-parametric regression method for the transition network slightly differs from that of the prior network. This improves the robustness of our approach: no post-process smoothing operation is needed.

5 Structure Learning

We take as input a small number of motion clips (usually four) of a particular type of motion. The motion need not be cyclic. These motion clips must be "similar" to each other, as we are trying to model the variation between them. Hence their lengths can be slightly different.

Let n_{seq} be the number of input motion sequences, where the l^{th} motion sequence has length n_l. For each sequence, the data in the first two frames ($\mathbf{X}[0]$ and $\mathbf{X}[1]$) are used to train the prior network. If n_{seq} is large enough, we can use the first two frames from each sequence. Otherwise, we can also take more pairs of frames near the beginning of each sequence. For example, we can take the first ten pairs of frames ($\mathbf{X}[0]$ and $\mathbf{X}[1]$, $\mathbf{X}[1]$ and $\mathbf{X}[2]$, ..., $\mathbf{X}[9]$ and $\mathbf{X}[10]$) as the training data for the prior network. Let n_{prior} be the total number of such instances or pairs of frames. For the transition network, we use the previous two frames to predict each frame.

Hence there are a total of $n_{trans} = \sum_l (n_l - 2)$ instances of training data for the transition network. The structure for the prior and transition networks are learned separately given this data.

Given the input data, we wish to learn the best structure (or set of edges in the DBN) that matches the data. The set of nodes are already defined as in Figure 2. We would therefore like to find the best G that matches the data D: $P(G|D) \propto P(D|G) \cdot P(G)$. This formulation leads to a scoring function that allows us to compute a score for any graph. We then use a greedy search approach to find a graph with a high score. The DBN literature provides many approaches to compute this score. One possibility is the Bayesian Information Criterion (BIC) score: there is one term in this score corresponding to $P(D|G)$ and one penalty term corresponding to $P(G)$. We use a similar score except we do not have a penalty term. Instead we perform cross validation across the data by splitting the data into training and test sets, a common strategy in existing DBN approaches [Ernst et al. 2007]. Doing cross validation allows us to measure how well a given graph matches the data without overtraining the graph on the data and without using a penalty term. Section 5.1 describes the greedy search for a graph, and the scoring functions for the prior and transition networks in more detail. To compute our score, we have to compute the conditional probability distribution for each node: $P(X_i \mid \mathbf{Pa}(X_i))$. We use a non-parametric regression approach to compute this probability. Section 5.2 provides justification and more details about this approach.

5.1 Structure Search

We learn the structure by defining a scoring function for any graph, and then searching for a graph with a high score. This is done separately for the prior and transition networks of the DBN. The use of search is shared with existing DBN techniques. However, the scoring function is different because of the non-parametric regression. It is intractable to find the graph with the highest score due to the large number of nodes in the graph. We therefore use a greedy search approach.

Prior Network. The prior network is a BN. To learn the structure, we start with any initial set of edges. We then apply an edge update that gives the best improvement towards the overall score. There are three possible edge updates: (i) an edge addition adds a directed edge between two nodes that were not originally connected, (ii) an edge deletion deletes an existing edge, and (iii) an edge reversal reverses the direction of an existing edge. Note that these are all subject to the BN constraint, and so we cannot apply an edge update that creates cycles in the graph. We continue to apply the best edge update until there is no improvement in the overall score. As this greedy method depends on the initial set of edges, we can repeat the algorithm multiple times by initializing with a different set of edges every time. We then take the set of edges with the highest score among the multiple runs.

We derive the scoring function by using a maximum likelihood approach: our goal is to find the graph that maximizes $P(D|G)$. Recall, however, that we do not use a $P(G)$ term as we use cross validation and split the data into training and test sets. The score for the prior network G_{prior} is

$$\begin{aligned} & log\, P(D|G_{prior}) \\ = \; & log \prod_{j=1}^{n_{prior}} P(X^{(j)}|G_{prior}) \\ = \; & \sum_{j=1}^{n_{prior}} log\, P(X^{(j)}|G_{prior}) \\ = \; & \sum_{j=1}^{n_{prior}} \sum_{i=1}^{2n} log\, P(X_i^{(j)}|\mathbf{Pa}(X_i)^{(j)}) \end{aligned} \tag{3}$$

where $X^{(j)}$ represents the j^{th} instance of the prior network training data, and $X_i^{(j)}$ is the value of node X_i of the j^{th} instance of data. We sum over each instance of data for doing leave-one-out cross validation: each j^{th} instance is one example of testing data and the corresponding training data (used in the non-parametric regression) does not include that instance. The training data for the j^{th} instance is the set of all n_{prior} instances of the prior network training data except the j^{th} instance. Note that we do not model the time component in the prior network even though they represent the first and second frames of the motion. Hence there are $2n$ total nodes. The last equality is due to the conditional independence of the nodes given their parent nodes. Since the total score can be separated into sums of terms for each node X_i, we keep track of each node's contribution to the total score. Each edge update in the greedy search can affect only one or two nodes, so we do not need to recompute the total score every time we update an edge.

Transition Network. We use a similar algorithm to learn the structure of the transition network. The difference here is that we do not allow any incoming edges to the nodes at time t and $t + 1$. The nodes at time t and $t + 1$ are assumed to be observed and are used to predict those at time $t + 2$. We initialize the graph with the edges from $X_i[t]$ to $X_i[t + 2]$ $(\forall i)$, and the edges from $X_i[t + 1]$ to $X_i[t + 2]$ $(\forall i)$. From our experience with the data, the search almost always selects these edges and therefore we always keep these edges throughout the search to make the process more efficient. The scoring function is similar to the one for the prior network. The score for the transition network G_{trans} is also derived from the $P(D|G)$ term:

$$\sum_{l=1}^{n_{seq}} \sum_{j=2}^{n_l - 1} \sum_{i=1}^{n} log\, P(X_i[j]^{(l)}|\widehat{\mathbf{Pa}}(X_i[j])^{(l)}) \tag{4}$$

where $X_i[j]^{(l)}$ is the value at node $X_i[j]$ of the l^{th} motion sequence of the transition network training data. This score is different from the BN score in that we start with the first two frames in each sequence, and compute the subsequent frames in the sequence by *propagating* the computed frames. So the second frame and the newly synthesized third frame are used to compute the fourth frame, the newly synthesized third and fourth frames are used to computed the fifth frame, and so on. The $\widehat{\mathbf{Pa}}$ notation represents this propagation of frames. The justification for this propagation instead of treating each instance separately is that the learned structure would otherwise not give a good result: the predicted trajectories deviated from the actual ones when we attempted to treat each instance separately. Intuitively, since we propagate the values when we synthesize a new motion given the first two frames, we should do this propagation when we learn the structure. We are effectively trying to compute how good a given structure is by trying to re-synthesize each input motion sequence given the first two frames, and comparing the synthesized sequence with the original data. Hence we sum over each motion sequence for doing cross validation: each l^{th} sequence is one example of testing data and the corresponding training data (used in the non-parametric regression) does not include that sequence. Thus the training data for the l^{th} sequence is the set of all n_{trans} instances of the transition network training data except those in the l^{th} sequence. Note that we sum over the n nodes in time $t + 2$ as these are the ones we are trying to compute in the transition network. Note also that the non-parametric regression for computing the probability in the transition network slightly differs from that of the prior network.

5.2 Non-Parametric Regression for Computing Conditional Distribution

The scoring functions for the prior and transition networks require the computation of the conditional probability $P(X_i|\mathbf{Pa}(X_i))$. We briefly describe the parametric approaches that we attempted to use. As these approaches did not work well, we instead rely on a non-parametric regression method.

We attempted to model the relationship between X_i and its parent nodes as a linear relationship, but we found that it is not appropriate for our motion data. We then attempted to model this relationship by nonlinear regression. We tried to find the parameters of a nonlinear function that takes the parents of X_i as input and X_i as output, where the nonlinear function is a sum of multivariate radial basis functions. While this worked well for the prior network of the DBN, it performed poorly for the transition network. We speculate this may be because there is not enough data to accurately estimate the parameters of a nonlinear function. Instead, we employ a non-parametric locally-weighted regression technique, which we found worked well for our data.

Prior Network. We assume that $P(X_i|\mathbf{Pa}(X_i))$ follows a Gaussian distribution, and use kernel regression to find the mean and standard deviation of this distribution. Recall that we are given the graph and training data. The graph allows us to find the parent nodes of X_i. The training data allows us to find instances of $(\mathbf{px}_k, x_k)$ corresponding to $(\mathbf{Pa}(X_i), X_i)$. Note that we also have the actual value of $\mathbf{Pa}(X_i)$, which we call $\mathbf{pa}(X_i)$. Since a large number of the instances $\mathbf{px}_k$ are far away from $\mathbf{pa}(X_i)$, we pick the k-nearest instances. The notation with the subscript k represents these nearest instances. We measure the distance with a Euclidean-distance metric: $D(\mathbf{px}_k, \mathbf{pa}(X_i))$. We then compute a weight for each instance:

$$w_k = exp\{-D(\mathbf{px}_k, \mathbf{pa}(X_i))^2/K_W^2\} \tag{5}$$

where K_W is the kernel width. Next, we compute a weighted mean and variance based on these weights:

$$\begin{aligned} \mu(X_i) &= \frac{\sum_k w_k x_k}{\sum_k w_k} \\ var(X_i) &= \frac{n_k}{n_k - 1} \cdot \frac{\sum_k w_k (x_k - \mu(X_i))^2}{\sum_k w_k} \end{aligned} \tag{6}$$

where n_k is the number of non-zero weights w_k, and the standard deviation $\sigma(X_i)$ is the square root of the above variance. For the prior network, we have cases where X_i has no parents. To compute $P(X_i)$, we find instances of x_k corresponding to X_i. The mean and standard deviation of X_i is then the mean and standard deviation of the instances x_k.

Transition Network. We compute one distribution for each node i at time $t+2$ ($X_i[t+2]$). The regression for the transition network is essentially the same as above with two important modifications. The first modification is that we also have a weighted velocity term when computing the distance function $D(\mathbf{px}_k, \mathbf{pa}(X_i[t+2]))$. This velocity term is $(X_i[t+1] - X_i[t])$ (recall that $X_i[t+1]$ and $X_i[t]$ are always parent nodes of $X_i[t + 2]$). Including this term allows us to find k nearest instances that better match $\mathbf{pa}(X_i[t + 2])$. The second modification is related to $X_i[t + 2]$, whose values we have to generate in order to compute probabilities and scores. Instead of dealing with "absolute" $X_i[t + 2]$ values, we deal with "delta" $X_i[t + 2]$ values. In Equation 6, instead of x_k representing the k instances of $x_i[t + 2]$ (where lower x means actual value), x_k now represents the k instances of $(x_i[t + 2] - x_i[t + 1])$. And instead of X_i representing $X_i[t + 2]$, X_i now represents $(X_i[t + 2] - X_i[t + 1])$. To generate an actual value of $x_i[t + 2]$, we take $\mu(X_i[t+2]-X_i[t+1])$ and add this to the existing $x_i[t+1]$ value. Intuitively, since the "absolute" values have a much wider range, sampling from this range will require post-process smoothing. The "delta" values have a small range, and sampling from it is more robust and no post-process smoothing is needed.

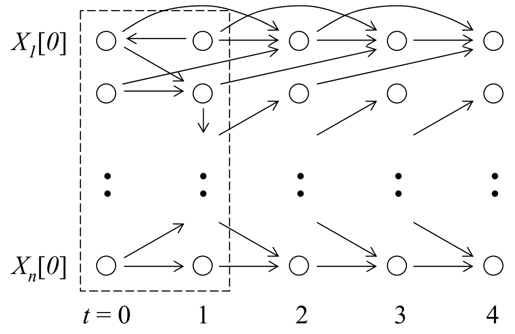

Figure 3: *We "unroll" the DBN from Figure 2 to synthesize new variants. We show here the unrolled network for 5 time frames. The first two frames come from the prior network of the DBN and cannot contain cycles. Since the DBN represents a joint probability distribution over the possible trajectories of each DOF, we sample from this distribution to generate new variants.*

6 Synthesis of New Variants

We can use the learned structure and the input data to synthesize an unlimited number of new spatial and temporal variants. Since the DBN represents a joint probability distribution, we sample from this distribution to synthesize new variants. We represent the μ's and σ's that are computed for each node as a set $(\vec{\mu}, \vec{\sigma})$. If we pick $\vec{\sigma} = \vec{0}$, this gives the mean motion of the inputs. The set $(\vec{\mu}, \vec{\sigma})$ represents variations of motions away from this mean motion. Note that the μ's are not fixed, since the μ's and σ's from previous time frames can affect the μ's in later time frames.

Prior Network. We synthesize the first 2 frames of a new motion with the prior network. We first find the partial ordering of the $2n$ nodes in the prior network. Such an ordering always exists since this network is acylic. We generate values for each of these nodes according to this ordering. The nodes at the beginning will be the ones without parents. We sample a value from each of the Gaussian distribution of these nodes. The rest of the nodes will depend on values already generated. We use the procedure in Section 5.2 to find the mean and standard deviation for each node, except that we use the learned structure and all the n_{prior} instances every time. We then sample a value from the distribution of each node.

Transition Network. Given the first 2 frames, we synthesize subsequent frames by "unrolling" the DBN (Figure 3). We perform one locally-weighted regression for each node at each time frame. We use the learned structure and all the n_{trans} instances every time. We use the procedure in Section 5.2 to compute actual values of

$X_i[t+2]$. The main difference is that after computing $\mu(X_i[t+2]-X_i[t+1])$ and $var(X_i[t+2]-X_i[t+1])$, we sample from this distribution and add the value to the existing $x_i[t+1]$ value to get the $x_i[t+2]$ value. *No post-process smoothing operation* is needed. If the input motions are cyclic, we can synthesize a continuous and unlimited stream of new poses.

7 Constraints

The synthesized poses from the previous section might need to be cleaned up for handling foot and hand constraints. This fixes footskate problems and also deals with cases where the foot/hand has to be at a specific position. We develop an inverse kinematics framework that fits with our DBN approach. Intuitively we need to satisfy three constraints: (i) the foot/hand needs to be at specific positions at certain times, (ii) the solution should be close to the mean values (at each node and time) predicted by the DBN, and (iii) the solution should maintain smoothness with respect to the previous frames. The first constraint is a hard inverse kinematics constraint while the last two are soft constraints. This naturally leads to an optimization solution:

$$\begin{aligned} \min_{\mathbf{q}_t} \quad & \{w_1\|\mathbf{q}_t-\overline{\mathbf{q}}_t\|^2 + w_2\|\mathbf{q}_t - 2\mathbf{q}_{t-1} + \mathbf{q}_{t-2}\|^2\} \\ & \text{s.t. } \|\mathrm{f}(\mathbf{q}_t) - \mathrm{pos}\|^2 = 0 \end{aligned} \tag{7}$$

where $\mathbf{q}_t$ is the set of DOFs for one foot or hand at time t. There are 6 joint angles for each foot, and 7 for each hand. $\overline{\mathbf{q}}_t$ is the set of mean values (of the corresponding nodes and time) predicted by the DBN, $\mathbf{q}_{t-1}$ and $\mathbf{q}_{t-2}$ are the DOFs from the previous two frames, f() is the forward kinematics function that gives the end-effector 3D position corresponding to $\mathbf{q}_t$, and "pos" is the 3D position that we want the foot/hand to be at. We run an optimization for each foot/hand and time frame separately, in the usual forward time order. If there is a large amount of motion, these 3D positions and frames can be found with automated methods [Kovar et al. 2002]. However, we find that it is not difficult to identify these manually for our motions. We initialize the optimization with the solution we sample from the DBN. Since the solution we get from Section 6 is already close to what we want, the optimization only makes minor adjustments and is therefore efficient. The optimization uses a sequential quadratic programming method. We set w_1 to 1 and w_2 to 5.

8 Evaluation

A main result of our work is that we can synthesize spatial and temporal variants of the input examples. *Spatial variation* means that no new pose is exactly the same as any of the input poses or previously synthesized poses. Spatial differences can usually be better seen in images of poses. *Temporal variation* means that a new variant motion has a different timing than any of the input motions or previously synthesized variant motions. It is important to recognize that a new variant does not have a one-to-one correspondence from any of the input motions. This means that the new variant is not simply a copy of one of the input motions plus some slight differences as is the case in previous work, but the timing of the whole motion itself is different. Temporal differences are better visualized in animations.

In general, we expect our approach to work on time-series data with DOFs that are correlated. This means that some DOFs are correlated with others, but it is not necessary that all DOFs are related to each other. The DBN model, by design, works on these types of data. Experimentally, we show that our approach works for two different sets of data: human full-body motion and 2D handwritten

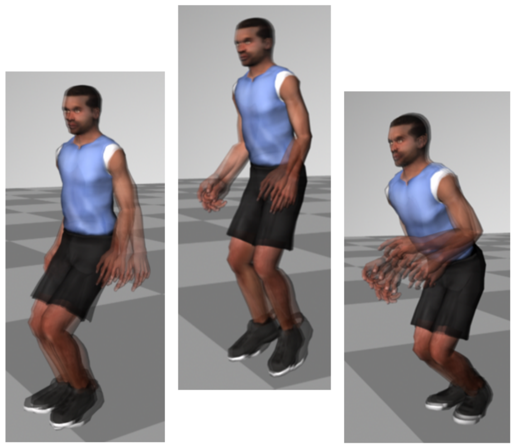
Figure 4: *Given the learned structure and just one jumping motion as inputs, we synthesize four new variant motions. We overlap poses from these four new motions at similar time phases of the jump (lowest point of the character before jump, highest point of jump, and lowest point after jump). We can see the variations in the poses at these time phases. The poses for the head vary the least because the head poses also vary the least in the input data.*

characters. The success of our approach on these two rather different data sets show that our approach is more generally applicable to similar kinds of temporal sequences.

We assume that our data satisfies a second-order Markov property in our DBN model. We made this assumption after first attempting to model the data with a first-order Markov model. While we can learn a structure and generate the first frame of motion from the prior network, the subsequent frames that are generated by the transition network produce highly unnatural motions. After a few frames, the new poses will diverge away from the poses in the input motions. Intuitively, the algorithm is unable to find nearest instances that are truly "near" the existing previous frame, and hence it cannot generate the corresponding next frame accurately. However, we found that using two previous frames works well in finding the k nearest instances, and this is the reason that a 2nd-order model works well. We did not try 3rd or higher order models, since we already have a simpler (2nd-order) model that works well. We believe that higher order models will produce similar results while having a longer run-time. By assuming a 2nd-order model, we have only 2 frames of data; but for our human motion with 62 DOFs, there are actually 124 pieces of information. We found that this information is enough for the algorithm to find the nearest "patches" (or nearest instances) of input data, in order to perform the non-parametric regression to generate a subsequent frame.

Results for Full-body Human Animation. We show results for five types of human motion data: cheering, walk cycle, swimming breast stroke, football throws, and jumping. We use, respectively, 433, 322, 384, 666, and 309 frames of data (at 60 frames per second) as input. These are the total number of frames for each motion type. We have four input motion clips in each case.

We find that four input motions is the smallest number that learns a DBN structure that gives good results. A larger number of inputs also works well, but we show the robustness of our method by showing that it works with only a few inputs. The values of k that we use to find the k nearest instances are between 15 and 60. In the learned DBN structure, each node has between 2 and 15 parent nodes (except for the nodes in the prior network that have no parents). After learning a structure, we can synthesize variants of the

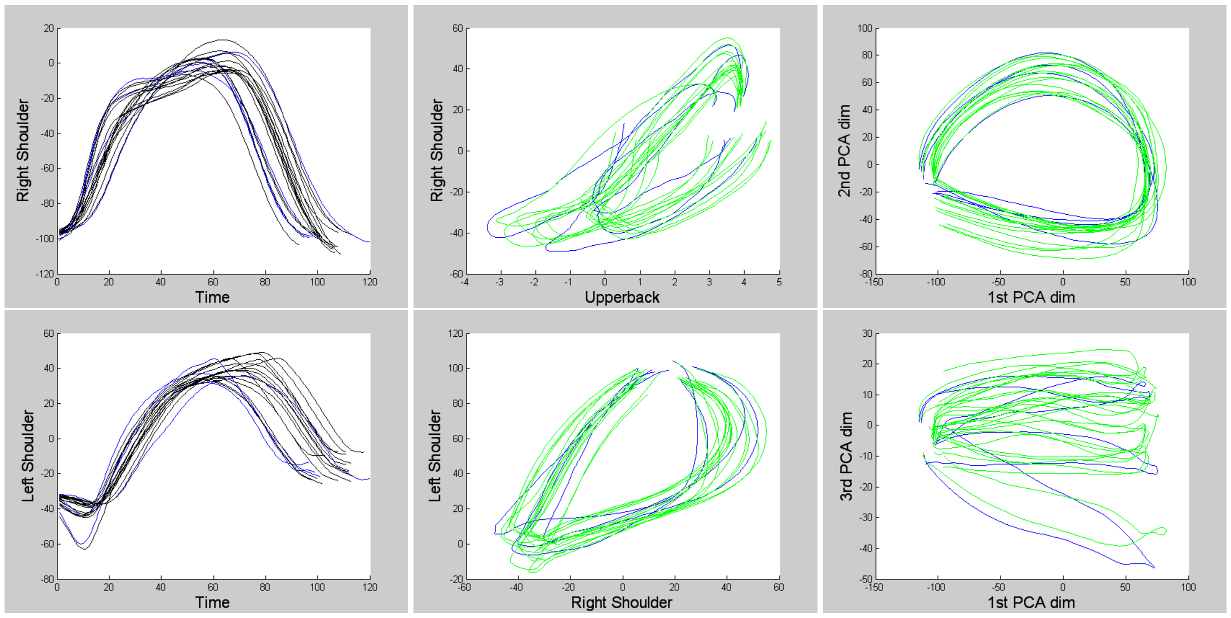

Figure 6: *Plots of four inputs (in blue) and fifteen output variants (in black or green) for cheering motion. Each curve represents one motion clip. Note that these motions are not cyclic. Left Column: Two selected plots of DOF vs. time. Middle Column: Two selected plots of DOF vs. DOF. Right Column: Two selected plots of PCA-dimension vs. PCA-dimension.*

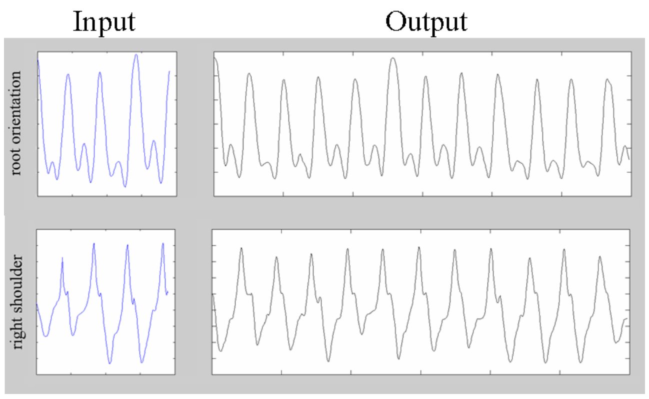

Figure 5: *We show plots of selected DOFs vs. time of the swimming breast stroke motion. We see that each new cycle in the output retains the general shape of the input data, but none of them is an exact copy of the inputs.*

four inputs. The results for cheering, walk cycle, and swimming breast stroke motions (Figure 1) show variants generated with the four inputs in each case. Given the learned structure and just one input motion clip, we can also use the same approach to synthesize variants of that single input. The results for football throws (please see video) and jumping motions (Figure 4) show variants generated with just one input in each case. It is possible to generate interesting variations from a single input. With just one input, the variation at each newly generated frame depends on the data from the k nearest neighbors of the one input motion. This can result in combining the DOFs from different frames to each frame of the newly synthesized motion. In addition, if the motion is cyclic, we can synthesize a continuous stream of new cycles. We show examples of these for walk cycles (please see video) and swimming motion (Figure 5).

Figure 6 show graphs of the input and output cheering motions. While these graphs are for cheering motion, they are typical of similar graphs of other motion types. Note that the new output variants follow the general trajectories of the inputs, but are not exactly the same. In the middle column of the figure, we can see some of the joint correlations. For example, knowing the value of the right shoulder can help us predict the value of the left shoulder. These joint relationships are learned automatically. Indeed, the discovered edges or dependencies in the DBN represent these kinds of joint relationships that exist in the data. For the right column of the figure, we performed PCA of the input and output data, and plotted the results from the first few PCA dimensions. The PCA reduces the 62-DOF data to 11 dimensions, keeping more than 99% of the energy.

Results for 2D Handwritten Characters. The original input data for the 2D characters were manually drawn by the authors directly on the 2D screen. We recorded the x and y positions of the mouse on the screen, and each character was drawn with one continuous stroke. We have one set of data for the character “a” and another set for the digit “2”. The average number of (x, y) points we have for each “a” is 103, and for each “2” is 66. We have four input characters in each case. We learn a DBN structure and synthesize new characters/digits using the same procedure, except that we now have 2 DOFs representing the x and y positions. The last column of Figure 1 shows both the four inputs and fifteen new outputs in each case. We can see the spatial variation in these images. The timing of each new stroke is also different, and this can be seen in the video.

To evaluate our method further, we have another example of 2D strokes where we can compare against the ground truth. We define a simple set of equations to generate the ground truth 2D circular strokes (Figure 7(left)). Note that there is some randomization in the equations to vary the ground truth strokes. We then select four strokes from the set of fifteen ground truth ones and use them as input. We learn a DBN and synthesize new output strokes with the same procedure. Figure 7(left) shows the four inputs and fifteen outputs. As expected, the fifteen outputs generated from our DBN

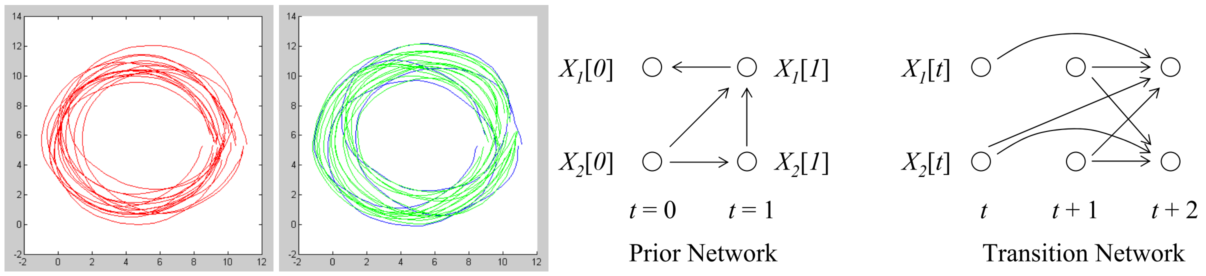

Figure 7: *Left: 15 ground truth strokes (red), 4 input strokes (blue) and 15 output strokes (green). Note that the blue and green ones are overlaid. Right: G_{prior} and G_{trans} from the learned DBN model. $\mu(X_2[0])$ is 5.91 and $\sigma(X_2[0])$ is 0.77.*

approach are visually similar to the ground truth strokes. Figure 7(right) shows the DBN that was learned from the four inputs.

Memory and Performance Time. We require memory to store the learned DBN structure and the four input motions. The DBN structure consists of a set of sparse directed edges in G_{prior} and G_{trans}, and the means and standard deviations of the nodes in G_{prior} that have no parents. The memory for the DBN structure is small, and hence the total memory is essentially the four input motions. It takes between half an hour and two hours to learn the DBN structure for each type of human motion we described above. This learning process can be done offline. The runtime process of synthesizing new human motion can be done efficiently: in our examples, it takes about 0.1 second to generate 1 second of motion.

User Study. We performed two experiments in the user study. For Experiment A, we compare "Our Variants" with "Motion Capture" data. "Our Variants" are motion clips generated by our approach. The purpose is to decide which is more natural. We ran this experiment for cheering motion and walk cycles separately. Each user watches a random mixture of 15 of these motion clips. After watching each motion, we ask the user to provide a score from 1 to 9 (inclusive) of how natural or human-like that motion is. A higher score corresponds to more naturalness. We tested 15 users, and we have a total of 225 scores. We performed ANOVA on these scores (Figure 8). For cheering motion, p is 0.9298 and this suggests that the means from the two samples (of "Our Variants" and "Motion Capture") are not significantly different. For walk cycles, p is 0.5779 and this again suggests that the means from the two samples are not significantly different. Therefore, for both cheering motion and walk cycles, motion synthesized by our approach is just as natural as motion capture data.

For Experiment B, we compare "Our Variants" with "Cycle Animation". "Our Variants" are long sequences where each sequence consists of at least 15 concatenated motion clips generated by our approach. These motion clips are all slightly different. A "Cycle Animation" is a long sequence consisting of at least 15 concatenated motion clips: each of these is randomly selected from the 4 input motions. The purpose is to decide which is more repetitive. A long sequence is repetitive if many of the motion clips are exactly repeated. We ran this experiment for cheering motion and walk cycles separately. Each user watches a random mixture of 15 of these long sequences. After watching each sequence, we ask the user to provide a score from 1 to 9 (inclusive) of how repetitive that sequence is. A higher score corresponds to more repetition. We tested 15 users, and we have a total of 225 scores. We performed ANOVA on these scores (Figure 8). For cheering motion, p is 1.0243e-8 and this suggests that the means from the two samples (of "Our Variants" and "Cycle Animation") are significantly different. For walk cycles, p is 4.4868e-8 and this again suggests that the means from the two samples are significantly different. Therefore, for both cheering motion and walk cycles, "Our Variants" are less repetitive than "Cycle Animation". In Experiment B, note that each long sequence has at least 15 motion clips. It takes some time to recognize whether or not there are clips that are exactly repeated. Hence Experiment B does not apply to relatively short animations, since "motion clones" are difficult to detect in short animations (as shown in [McDonnell et al. 2008]).

Experiments with Adding Noise. A simple possible approach to generate variation is to add noise to existing motion. We experimented with two such methods on the walk cycle data. The first is a naive or strawman method. We time-warp the four input walk cycles, compute simple statistics of the time-warped data for each DOF separately, and use this information to add smoothed noise to one of the four input cycles. We check that the noise-added motion is changed by a similar amount compared to the variants that our DBN approach generates. We do so by taking pairs from our fifteen variants and the four inputs (each pair has one variant and one input), computing the normalized sum of squared differences of joints between each pair, and modeling these sums as a normal distribution. We also compute the normalized sum of squared differences of joints between the noise-added motion and its corresponding input, and check that this sum is within one standard deviation of the mean of the normal distribution above. Our video shows examples where we add noise only to the left shoulder and elbow. The animation shows that the left shoulder/arm motion is unnatural, and does not fit with the rest of the walking motion. In contrast, our DBN approach will learn that the left shoulder is correlated with other joints, and handle these issues autonomously. In another example, we add noise to all joints. While the overall walk motion still exists, it is visibly evident that the poses and timing of the motion are awkward. Furthermore, adding noise requires a smoothing process that can affect details of the original motion.

The second method is to add band-limited noise to one of the four input cycles with the Perlin noise function [1995]. We also perform the same noise-addition check as in the first method. Our video shows examples where we add noise to several joints. We find that a trial-and-error process of manual parameter tuning is needed. Most importantly, a human understanding of the motion (ie. if the left arm swings higher, the right arm is more likely to swing higher) is required to add noise in a principled way. Otherwise, the motion can become spatially or temporally awkward.

9 Limitations

The main limitation of our approach is that the input motion examples have to be "similar but slightly different". They have to be "similar" because we are learning a model for that particular type of motion. They have to be "slightly different" because the small differences among the inputs are where we get the variation from. In the results section, we have shown examples that work with our

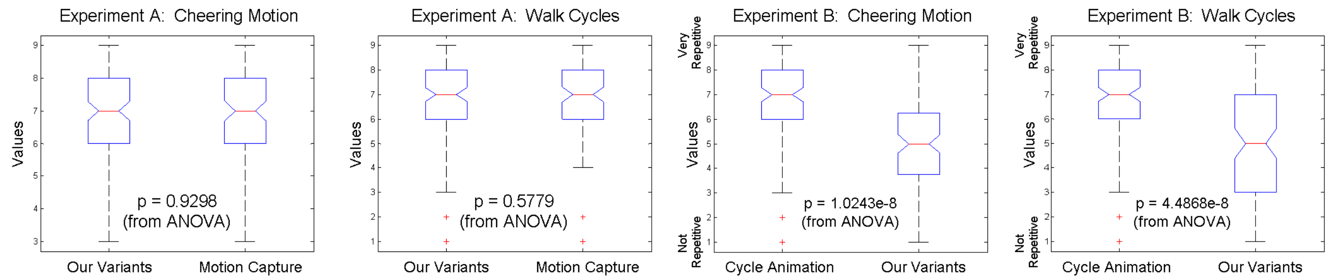

Figure 8: *ANOVA results from the user study.*

approach and are "similar". Here, we describe examples of inputs that do not work with our approach and are not "similar". For our walk cycle data, we have four input motions where the character walks two steps forward. If there is a walk cycle where the character swings the arms much higher (please see video for the animation), it will not fit with the original four motions and will not work as another input motion. We are still able to use the five motions together to learn a DBN structure, but the synthesis step may not produce a reasonable output motion. However, if we have four or more of such higher-arm-swinging walk cycles, they can be used together in our approach effectively. Another example is a walk cycle where the character turns slightly to one side while walking forward. This will also not fit with the original four inputs. We also show examples of the handwritten digit "2" (please see video) that do not work with our original four inputs.

It is difficult to precisely define what is meant by "similar" motions. Instead of making such a definition, we introduce a method to characterize the types of inputs that work well with our approach. We start with a given set of training data that we know works well. This data can either be selected manually (ie. we tested them on our approach) or can come from the results of this method. As an example, we started with eight walk cycles that we have already selected. We split these eight into groups of six and two. We learn a DBN with the group of six and compute likelihoods with the learned DBN for each of the other two. We use the likelihoods that are described in Section 5. We repeat this process for different combinations of six and two. The idea is to get a number of likelihoods that we can use to characterize the training data. With this set of likelihoods, we can set a threshold for deciding the likelihoods that we should accept in a new set of testing data. We set the threshold to be the tenth percentile of all the likelihoods. We can now take a new testing set of motion clips. We used a new test set of eight walk cycles in our example. We again separate this set into different groups of six and two, so that we can compute a likelihood for each motion clip. We then eliminate the motion clip with the lowest likelihood if it is lower than the threshold. We now have seven motion clips and we repeat the process to compute likelihoods for each of the seven clips. We stop this process until the lowest likelihood is above the threshold. In our example, this process stops with five walk cycles.

10 Discussion

We have presented a method for modeling and synthesizing variation in motion data. We use a Dynamic Bayesian Network to model the input data. This allows us to build a multivariate probability distribution of the data, which we sample from to generate new motion. Given input data of a type of motion, our model can be used to generate new spatial and temporal variants of that motion. We show that our approach works with five types of full-body human motion, and two types of 2D handwritten characters. We perform a user study to evaluate our approach. For applications such as crowd animation, our method has the advantage of being able to take small, pre-defined example cycles of motion, and generate many variations of these cycles. We believe that adding noise to existing motion requires a manual trial-and-error process; the significance of this paper is to provide a formal way to autonomously model and synthesize variation with a small amount of data.

It is informative to highlight the differences between our method and interpolation methods [Rose et al. 1998; Wiley and Hahn 1997]. Interpolation methods generate new motions that are "in between" the original examples. Our method models a probability distribution of the original examples. Our method produces motions that have different poses and timings than the input motions. This is difficult to generate with interpolation methods. Finally, interpolation methods require at least two example motions. Given a learned structure, our model can synthesize new variants from just one example motion.

There exist alternative approaches for adding noise to motion data to create variation. We experimented with an approach that represents motions with their time-warped representations and then performs PCA on them. We found that adding noise to the PCA-reduced motion does not necessarily produce natural motion. While the overall motion may be partially synchronized (ie. the left and right arm swings in a walk cycle) due to the PCA reduction, we still require much manual tuning to successfully add noise. Another approach is to take a convex combination of a small number of motions and add noise to it. However, this is similar to the idea of interpolation described above, which we view to be different from our approach. In addition, we believe that adding noise requires a manual trial-and-error process, whereas our approach can autonomously generate a large number of outputs.

The learning and synthesis processes can, in theory, go into regions where there is no data to support the regression. In practice, we have not found this to be an issue. This might be due to the non-parametric regression which is quite robust. On the other hand, this was an issue when we initially attempted to use parametric regression methods. The learning and synthesis processes also need to find the approximate phase of the overall motion in order to perform the regression. It is possible for some of the k nearest neighbors to not be within the correct phase of the overall motion. However, this effect is minimized due to the other nearest neighbors and the non-parametric method as a whole.

One interesting area for future work is to provide a method for the user to control the variation that is generated. One possible challenge is to develop an intuitive way to control the "amount" of variation. It is difficult to define what is "more" variation as this depends on the input data. If the motion is jumping and we have input data that has large variations in the swinging of the arms, then the synthesized motions will also have large variations in the arm swing. If the input data has more variation in the head movement, the synthesized motions will have more variation in the head. Hence one way to "control" the output motion is simply by taking different input data to begin with. Another challenge is to enable the user to generate "more" variation in a motion while automatically constraining the output to lie within the "natural" range of movement.

Another possibility for future work is to further explore the difference between "Cycle Animation" and "Our Variants" (from Experiment B of the user study). The cycle animations are usually relatively easy to identify. However, our variants are more difficult to identify, and can often be mistaken as cycle animations. Even though our variants can be very different numerically, they may only be slightly different visually. Thus, there is still room for improvement.

Another direction of future work is to use the idea of variation to compress motion data. If we can say that a set of motion clips are variations of each other, it may be possible to discard some of these motions. This is beause we can potentially re-synthesize a discarded motion from the remaining motions, since the discarded one is a variation of the remaining ones.

Acknowledgements

We thank the CMU graphics lab for providing advice on this paper and resources for video production. We also thank the anonymous reviewers for their comments.

References

BODENHEIMER, B., SHLEYFMAN, A. V., AND HODGINS, J. K. 1999. The effects of noise on the perception of animated human running. In *Computer Animation and Simulation 1999*, N. Magnenat-Thalmann and D. Thalmann, Eds., 53–63.

BRAND, M., AND HERTZMANN, A. 2000. Style machines. In *SIGGRAPH 2000*, 183–192.

CHAI, J., AND HODGINS, J. K. 2007. Constraint-based motion optimization using a statistical dynamic model. In *ACM Transactions on Graphics 2007*, vol. 26 (3), 8.

CHENNEY, S., AND FORSYTH, D. A. 2000. Sampling plausible solutions to multi-body constraint problems. In *SIGGRAPH 2000*, 219–228.

ERNST, J., VAINAS, O., HARBISON, C. T., SIMON, I., AND BAR-JOSEPH, Z. 2007. Reconstructing dynamic regulatory maps. *Molecular Systems Biology*, 3:74.

FRIEDMAN, N., MURPHY, K., AND RUSSELL, S. 1998. Learning the structure of dynamic probabilistic networks. *Uncertainty in Artifical Intelligence*, 139–147.

GHAHRAMANI, Z. 1998. Learning dynamic bayesian networks. *C.L. Giles and M. Gori (eds.), Adaptive Processing of Sequences and Data Structures. Lecture Notes in Artificial Intelligence*, 168–197.

HARRIS, C. M., AND WOLPERT, D. M. 1998. Signal-dependent noise determines motor planning. *Nature 394*, 780–784.

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

IKEMOTO, L., ARIKAN, O., AND FORSYTH, D. 2009. Generalizing motion edits with gaussian processes. In *ACM Transactions on Graphics 2009*, vol. 28 (1), 1.

JENKINS, O. C., AND MATARIĆ, M. J. 2004. A spatio-temporal extension to isomap nonlinear dimension reduction. In *ICML 2004*, 441–448.

KOVAR, L., GLEICHER, M., AND SCHREINER, J. 2002. Footskate cleanup for motion capture editing. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA) 2002*, 97–104.

KWON, T., CHO, Y.-S., PARK, S. I., AND SHIN, S. Y. 2008. Two-character motion analysis and synthesis. *IEEE Transactions on Visualization and Computer Graphics 14*, 3, 707–720.

LI, Y., WANG, T., AND SHUM, H.-Y. 2002. Motion texture: a two-level statistical model for character motion synthesis. In *ACM Transactions on Graphics 2002*, vol. 21 (3), 465–472.

MAIM, J., YERSIN, B., AND THALMANN, D. 2008. Unique instances for crowds. In *Computer Graphics and Applications*.

MCDONNELL, R., DOBBYN, S., AND O'SULLIVAN, C. 2006. Crowd creation pipeline for games. In *International Conference on Computer Games*, 181–190.

MCDONNELL, R., LARKIN, M., DOBBYN, S., COLLINS, S., AND O'SULLIVAN, C. 2008. Clone attack! perception of crowd variety. In *ACM Transactions on Graphics 2008*, vol. 27 (3), 26.

NEAPOLITAN, R. E. 2003. *Learning Bayesian Networks*. Prentice Hall.

PERLIN, K. 1995. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics 1*, 1, 5–15.

PULLEN, K., AND BREGLER, C. 2000. Animating by multi-level sampling. In *Proceedings of Computer Animation*, IEEE Computer Society, 36–42.

PULLEN, K., AND BREGLER, C. 2002. Motion capture assisted animation: Texturing and synthesis. In *ACM Transactions on Graphics 2002*, vol. 21 (3), 501–508.

ROSE, C., COHEN, M. F., AND BODENHEIMER, B. 1998. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications 18*, 5, 32–41.

THALMANN, D., KERMEL, L., OPDYKE, W., AND REGELOUS, S. 2005. Crowd and group animation. *ACM SIGGRAPH Course Notes*.

WANG, J. M., FLEET, D. J., AND HERTZMANN, A. 2008. Gaussian process dynamical models for human motion. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30 (2), 283–298.

WILEY, D. J., AND HAHN, J. K. 1997. Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Applications 17*, 6, 39–45.

YU, Q., AND TERZOPOULOS, D. 2007. A decision network framework for the behavioral animation of virtual humans. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA) 2007*, 119–128.