Modeling Variation in Motion Data

Manfred Lau **Ziv Bar-Joseph** **James Kuffner**

April 2008
CMU-CS-08-118

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We present a new method to model and synthesize variation in human motion. Given a small amount of input motion data, we learn a generative model that is able to synthesize new output motion variations that are statistically similar to the input data. The new variations retain the features of the original data examples, but are not exact copies. Our model does not require timewarping or synchronization of similar sequences of motions. We learn a Dynamic Bayesian Network model from the input data that enables us to capture properties of conditional independence in the data, and build a multivariate probability distribution of it. We present synthesis results across a range of different types of motions, and demonstrate novelty and aesthetic appeal of the new variations generated with respect to the input motions. Our technique can synthesize new motions efficiently and has a small memory requirement.

Keywords: Human Simulation, Variation, Machine Learning

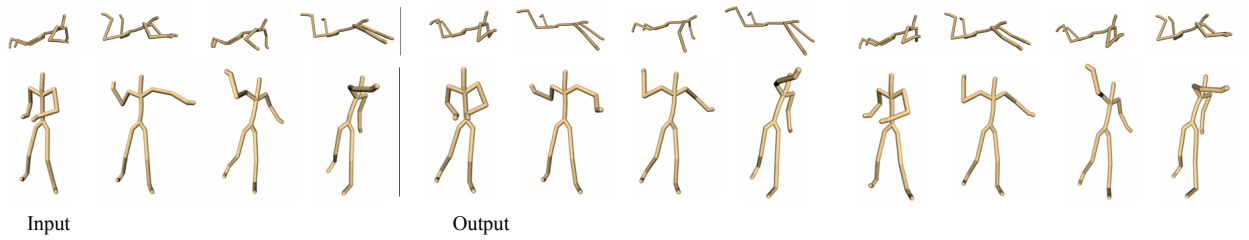

Figure 1: **Top row:** We take a few cycles of a swimming motion as input, and synthesize a long and continuous stream of new cycles. Each new cycle has a different timing and none of the poses is exactly the same as any other pose. **Bottom row:** Given a trained model structure for throwing motions, we can take just one throwing motion and generate new variations of it.

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. Unfortunately, current animation systems lack the ability to realistically produce these subtle variations. Variation is important because it can serve as a distinguishing feature between realistic and unrealistic motion sequences. For example, if an animation contains motion cycles that are simply played back repeatedly, users recognize that the animation is synthetic and unrealistic. This is because actual human motion rarely exhibits precisely the same cycle times with exactly the same poses.

We are motivated by applications such as crowds and character animation in games, where a small number of motion clips that are played repeatedly appear monotonous and unnatural. Typical crowd animation systems [14] utilize a few walking motion clips for every walking cycle and every character of the simulation. This can lead to synthesized motions that look obviously repetitive and unrealistic. Hence a variation model for even one such walk cycle has the potential to greatly improve the naturalness of the output animations. Crowds in films [24] may also depend on a small number cycles of motions to animate multiple characters. Our method can be used in conjunction with existing crowd motion generation techniques [22, 23] to synthesize a larger variety of motions for crowds. Game applications, which tend to have strict memory limitations, also typically contain a small number of motion clips that are played repeatedly. For example, a character in a football video game that often reuses the same motion clips may appear repetitive and unnatural to the user. Our motion variability model can also be applied to databases of motions (*mocap.cs.cmu.edu* and *www.moves.com*). By synthesizing variations for every motion in a database, the overall space of possible motions can be greatly enlarged. We can also apply our variation model to motion graph techniques [1, 9, 11], and generate variations of motions in the existing data in addition to obtaining the benefit of the motion graph structure.

Previous methods have constructed noise functions and added them directly to procedural motions [16] and motion data [2]. However, there is no guarantee that the added noise will match well with the existing motions. Adding unsupervised random noise to existing motions can lead to artifacts in the motion, while adding noise in a supervised way [2] requires human intervention. Adding noise also requires a trial and error process of manual parameter tuning.

We believe that variation should not be just an additive noise component. Recent biomechanical

research has argued that variation is not just noise or error, but is a functional component of motion. For example, one theory for explaining variation in arm trajectories is that the variation exists as a result of minimizing the variance in the final arm position [7]. From this point of view, adding random noise to existing motions is not accurate.

We instead approach the problem by taking a small number of motion clips that represent variations of the same motion, learning a model from this data, and then using this model to synthesize new variations of the motion. This is different from previous methods as the variations we generate come from the data, and is not a separate additive component. The advantages of this approach are that the variation model can be learned automatically from data, and the synthesized output motions will be statistically similar to the input data. Our goal is to generate new motions that are not exactly the same as the original motions while retaining their major salient features. We would like to have a model that can handle a small number of motion clips, as it is difficult to acquire a large number of examples of one motion. In addition, we do not want our method to require timewarping or synchronization of these motion clips. This is common in interpolation methods [20], and requires manual intervention.

We model the data with a Dynamic Bayesian Network (DBN) [5, 6], which allows us to achieve the properties discussed above. A DBN represents a multivariate probability distribution of the variables in the model. The variations that we generate come from this distribution. For our motion data, it captures the conditional independence relationships of the degrees-of-freedom (DOF) in the motion. This relationship allows us to build a distribution of values for each DOF based on the values of a few DOFs in the previous time steps. The DBN model also describes the temporal relationship in the data. This allows us to use only a few examples of each motion rather than a large number of examples. In addition, each synthesized motion does not have a one-to-one correspondence with any of the input motions. This means that the synthesized motion is not just a copy of one of the input motions plus some slight differences, but the timing of the whole motion itself is different. There are three major steps for learning a model and synthesizing new motions. First, we learn the structure of the DBN with a few motion examples. We use a greedy algorithm based on a variant of the Bayesian Information Criterion score to find a good structure. Second, we use the learned structure and the original data to synthesize new motions. Given a learned structure and just one motion cycle, we can even generate variations of this one cycle from the model. Finally, we use an inverse kinematics method developed within our DBN framework to satisfy the foot and hand constraints.

We demonstrate our approach by synthesizing new variations of different kinds of motion: jumping, football throws, swimming, and reaching. One important use of our method is that we can take a few examples of a motion as input, and then synthesize an unlimited number of variations of these motions. If the motion is cyclic, we can generate an unlimited and continuous stream of motions. Given a trained model structure, we can even take just one example of the motion and generate variations of that piece of motion. We show that the new variations generated from the model have different timing and poses distinct from the input motions. Hence no pose will be repeated even in a long continuous stream, much like what is observed in actual human repetitive motions. The memory requirement of the model consists of only the space required to store a few examples of input motion. Most of the processing time is in the learning phase; the runtime for synthesizing new motions is very efficient and can be done as a continuous stream one frame at a time.

1.1 Related Work

There has not been much work on generating variation in human motion. We describe related work in three areas: adding noise to motions, generating statistically similar motions by learning from data, and style and interpolation methods for human motion synthesis.

One previous approach for generating variation in motions is to add noise. Perlin [16] adds noise functions to procedural motions to create more realistic animations of running, standing, and dancing. Bodenheimer and his colleagues [2] adds noise to cyclic running motions. The noise is added only to the upper body, and is synchronized with the arm swings in the running cycle. Adding noise in such a supervised way requires human knowledge and parameter tuning. We show that adding random noise in an unsupervised way can lead to unnatural motions. Our approach is fundamentally different because the variations that we generate automatically come from the data and is not a separate additive component.

Pullen and Bregler's work [17] to generate motions that are slightly different but similar to the data is most closely related to our work. They model the correlations between the DOFs in the data with a distribution, and synthesize new motions by sampling from this distribution and smoothing the motions. However, they have to define certain correlations manually. 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 relationships directly from data. In addition, they used their method to animate a 2-dimensional 5-DOF wallaby figure, and a more complex 3D character in later work [19]. We demonstrate results of different kinds of motions for a full human figure. Our method is also similar to the "Texturing" method by Pullen and her colleagues [18]. They used the idea that the joints of a human figure are correlated to predict the values for some DOFs given the values of other DOFs. Our DBN framework also depends on this observation to predict new DOF values. Li and his colleagues [12] also generated new motions that are statistically similar to the original data. However, they used 20 minutes of dancing motion as training data. If a large amount of data is available, it is possible to just randomly replay or re-organize certain 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.

There has been work on learning the style of motions from training data [3] and transferring the style between motions [8]. Style and variation differ in the following way: a happy walk and a sad walk are different styles of walking, while two happy walks are different variations of a motion. Interpolation methods [20, 25] 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 five-foot jumps to generate variations of that jump.

1.2 Overview

We start with a description of a Dynamic Bayesian Network model (Section 2). Given a small number of motion clips that represent variations of a motion, we learn the structure of a DBN model automatically (Section 3). We use a nonparametric regression approach to compute the probability distributions, and we justify this approach in Section 3.2. This is one important 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 variations of that motion (Section 4). We develop

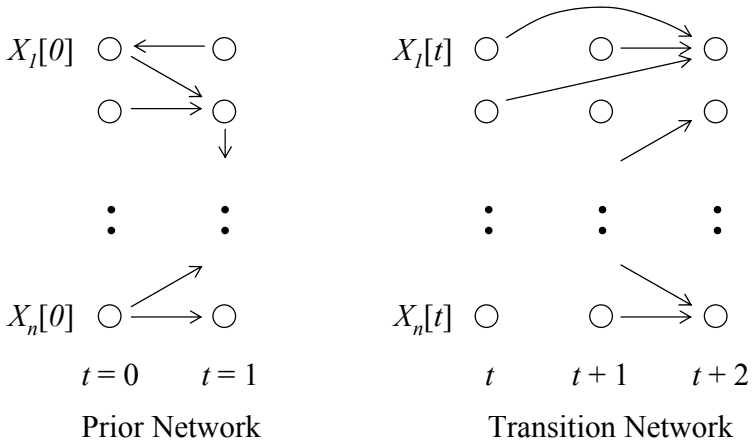

Figure 2: A DBN for the variables $X_1, ..., X_n$. Each node X_i represents one DOF in the motion data. We use the prior network to model the first 2 frames. The transition network then models subsequent frames given the previous 2 frames. We assume a 2nd-order Markov property because it is the simplest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

an inverse kinematics framework that fits with our DBN model to satisfy foot and hand constraints (Section 5).

2 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 [5, 6].

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 [15]. However, we found that neither using a linear model nor a nonlinear model works well for our case. This might 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 the 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. We have 62 DOFs in our motion data, so n is 62. The prior network G_{prior} represents the

joint distribution of the nodes in the first two time points, $\mathbf{X}[0]$ 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 6. 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 represents the "function" defined in a non-parametric approach.

3 Structure Learning

We take as input a small number (usually four in our examples) of motion sequences of a particular motion. These motions must be similar to each other, as we are trying to model the variation between similar motions. For example, if we are modeling swimming motions, we can take similar cycles of breast stroke motions or similar cycles of free style motions, but not a mixture of these two types. The motion need not be cyclic. Although if it is cyclic, we can synthesize an unlimited amount of new cycles as a continuous stream.

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 that matches the data. This means that we would like to find the best graph or set of edges in the DBN that best 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 [4]. 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 3.1 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 3.2 provides justification for this approach, and more details about the method.

3.1 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 search part of our algorithm is the same as existing DBN techniques; the scoring function however 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. Hence we 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: 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 $P(D|G)$. Remember 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 at 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. So 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 will not have to recompute the total score every time we update an edge.

Transition Network. For the transition network, we use a similar algorithm to learn the structure. 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 $t+2$. 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 (Figure 3). 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.

3.2 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 decided to use 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

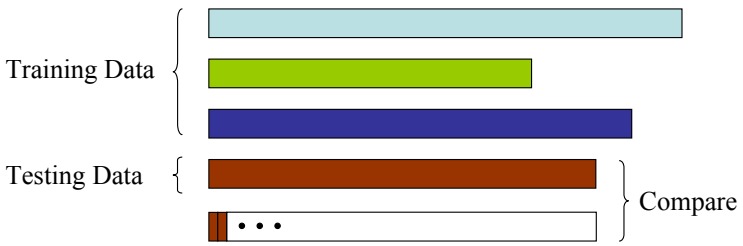

Figure 3: When learning the structure for the transition network, we do a cross validation over each motion sequence. We take each sequence as testing data, and use the others as training data. For the testing sequence, we take the first two frames as input and re-synthesize the whole sequence with the given structure. The newly synthesized sequence is then compared to the original data to evaluate the structure. This is what we compute intuitively in the scoring function for the transition network of the DBN.

multivariate radial basis functions. While this worked well for the prior network of the DBN, it performed poorly for the transition network. This might be because there is not enough data to accurately estimate the parameters of a nonlinear function. Hence we decided to try a non-parametric method. We found that a kernel regression approach worked well for our data.

We assume that $P(X_i|\mathbf{Pa}(X_i))$ is a guassian distribution, and use kernel regression to find the mean and standard deviation of this distribution. We compute this distribution for each node i and time t (for the transition network). 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 allow 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 $\mathbf{Pa}(X_i)$ in the training set, 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. With the mean and standard deviation of X_i, we can now compute $P(X_i|\mathbf{Pa}(X_i))$. 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.

We use these weighted means and variances when we synthesize new motions. However, when we learn the structure, we only use the weighted means. This is because we only need to find the most likely values when we compute scores for learning the structure.

4 Synthesis of New Motions

We use the learned structure and the input data to synthesize new motions that are different variations of the input motions. Since the DBN represents a joint probability distribution, we sample from this distribution to synthesize new motions. We can synthesize an unlimited amount of new motions that will not repeat themselves.

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 input motions. 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 guassian distribution of these nodes. The rest of the nodes will depend on values already generated. We use the procedure given in Section 3.2 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 4). We similarly find the mean and standard deviation for each node at each time frame, and sample from this distribution. We perform one non-parametric regression for each node at each time frame. Again, we use the learned structure and all the n_{trans} instances every time. To prevent the values from being too far away from the mean, we avoid sampling values more than two standard deviations away from the mean. Since sampling the values does not take into account the smoothness of the trajectories, we smooth the trajectories for each DOF independently. Finally, if the input motions are cyclic, we can repeatedly generate an unlimited number of frames to synthesize a continuous stream of poses.

5 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 have 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:

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

We run an optimization for each foot/hand and time frame separately. So $\mathbf{q}_t$ 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. $\overline{\mathbf{q}}_t$

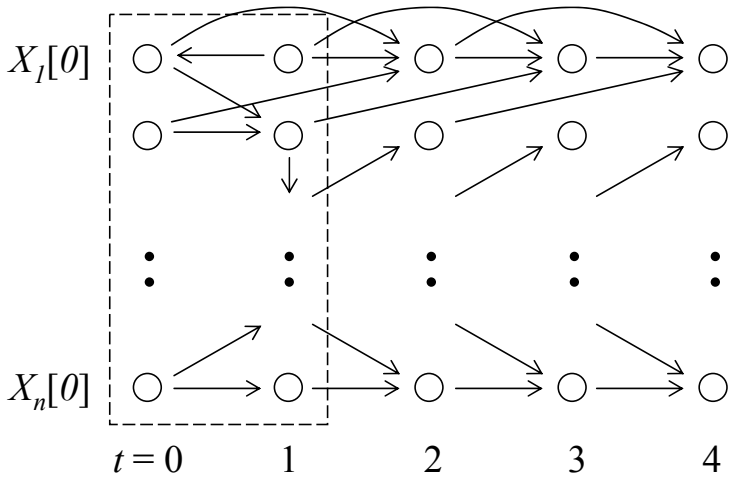

Figure 4: We "unroll" the DBN from Figure 2 to synthesize new motions. We show here the unrolled network for 5 time frames. Note that the first two frames come from the prior network of the DBN and may contain cycles. Since the DBN represents a joint probability distribution over the possible trajectories of each DOF, we sample from this distribution to generate new motions. It is important to recognize that the synthesized motion does not have a one-to-one correspondence from any one of the input motions. This means that the synthesized motion is not just a copy of one of the input motions plus some slight differences, but the timing of the whole motion itself is different. Furthermore, no new pose is exactly the same as any previous pose.

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. If there is a large amount of motion, these 3D positions and frames can be found with automated methods [10]. But we find that it is not difficult to identify these manually for our motions. We initialize the optimization with the solution we sampled from the DBN. Since the solution we got from Section 4 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.

6 Results

We demonstrate the generality of our method by taking motion data of different kinds of motions and synthesizing new variations of the input data. The key result to recognize in our examples is that the new variations retain the features of the input, but are not exact copies of it. The new motions have different timings compared to the inputs, and no pose is repeated even if a long sequence is synthesized.

In our examples, we find that we need about four motions to train the DBN structure. A smaller amount did not produce a structure that can synthesize good motions. A larger amount would work better, but four is the smallest number that learned a reasonable structure. Once we learn a structure, we can synthesize many variations of these four inputs. We can also take just one or two of

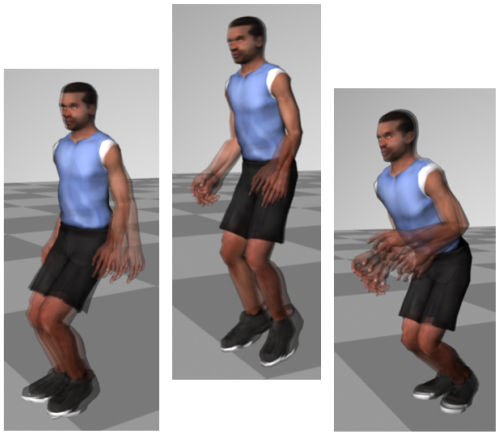

Figure 5: We take one jumping motion as input, and synthesize four new variations of this motion. We overlay poses from these four new motions at similar time phases of the jump (lowest point of the character before jump, highest point of jump, and lowest point after jump). We can see the variations in the poses at these time phases. The poses for the head vary the least because these head poses also vary the least in the input data.

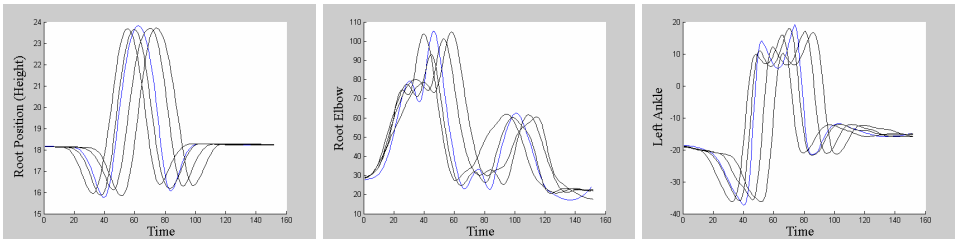

Figure 6: We use one jumping motion as input to generate four new motions. We show here plots of selected DOFs vs. time. In each plot, the blue curve is the input motion, and the four black curves are the new motions. The new curves retain the features of the input motion, but are not exact copies of it.

these inputs, and synthesize many variations of these. Our results can be seen in the accompanying video. We also show in our video that taking an input motion, adding noise randomly to some of the DOFs (except the DOFs for the feet), and smoothing the motion trajectories do not work well. We argue therefore that adding noise randomly to existing motions does not necessarily produce natural motions. It often requires manual parameter tuning along with a trial-and-error process.

Jumping. We use four jumping motions to train the DBN structure. We then take just one of these inputs and generate four new variations of it. The reason for taking one of these inputs is to test if our method can work with just one input motion; we could have taken any number of the original inputs. We can see in the animation that the timing of these jumps are different. Figure 5 shows some of the poses of these four new motions. Note that the poses at similar time phases of the jump are different. Figure 6 shows the trajectories of some DOFs of the motion for the single input motion and four new motions. Note that the new motion curves are similar to the input curve, but none of them are exactly the same.

Swimming - Breast Stroke. The swimming motions (breast stroke and free style) are cap-

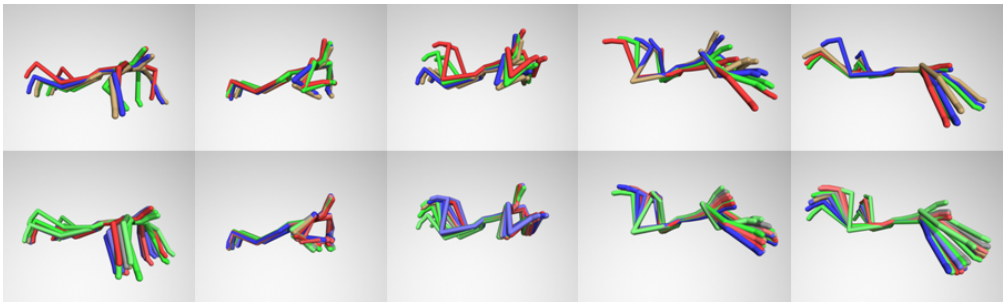

Figure 7: **Top row:** We take four cycles of a swimming breast stroke motion as input. The sequence of images show the motion of each of the four cycles overlayed with each other. Each color represents a separate cycle. **Bottom row:** We synthesize eleven new cycles, overlaid with each other here.

tured with the subject lying on a stationary platform on the ground. We take six seconds of a swimming breast stroke motion as input data, and synthesize twenty seconds of new motion. The new motion is a continuous sequence. No blending is required between each cycle of the breast stroke motion. Figure 1 shows some example poses from the input and output sequences. Figure 7 shows the image sequences for the cycles of the input data and new motion. We can see the variations of the motions in these sequences. Figure 8 shows plots of selected DOFs of the motions. We can use our method to generate a continuous and long sequence of new motion given just four cycles of input data.

Swimming - Free Style. Similar to the breast stroke motion, we take a few cycles of the free style motion to generate new cycles of motion. The important difference for the free style motion is that the arms and legs do not synchronize with each other. The arms exhibit a cyclic motion while the legs exhibit another cyclic motion that does not synchronize with the arms. One limitation of our structure learning process is that it was not able to learn this separation between the arms and legs. If we perform learning in the usual way, the synthesized motion would produce correct cycles of motions for the arms, but the legs would try to stop and synchronize itself with the arms. Our solution here is to specify that the DOFs of the arms cannot affect the DOFs of the legs, and vice versa. This is not manually tedious, and produces motions where the arms and legs move naturally. Figure 9(left) shows a similar pose from four of the newly synthesized cycles. We can see that the arms in these four poses are synchronized but the legs are not. This is expected as the arms and legs each have their own cyclic motions.

Football Throw. We use four football throwing motions to train the DBN structure. Given this learned structure, we take just one input motion and synthesize four variations of it. Figure 1 shows some example poses of the input and output motions. Figure 9(middle) shows selected poses of the four motions overlaid with each other. We can see the variations among the new poses in the figure.

Reaching. We use four reaching motions to train the DBN structure. We then take two of these inputs to generate new variations of these two. The significance here is that we use our IK framework to fix the hand position of the poses near the end of the motion. We show that we can get the hands to a specific position, and still get variations in the motion trajectories of the arm. Figure 9(right) shows the poses at the end of the four new motions.

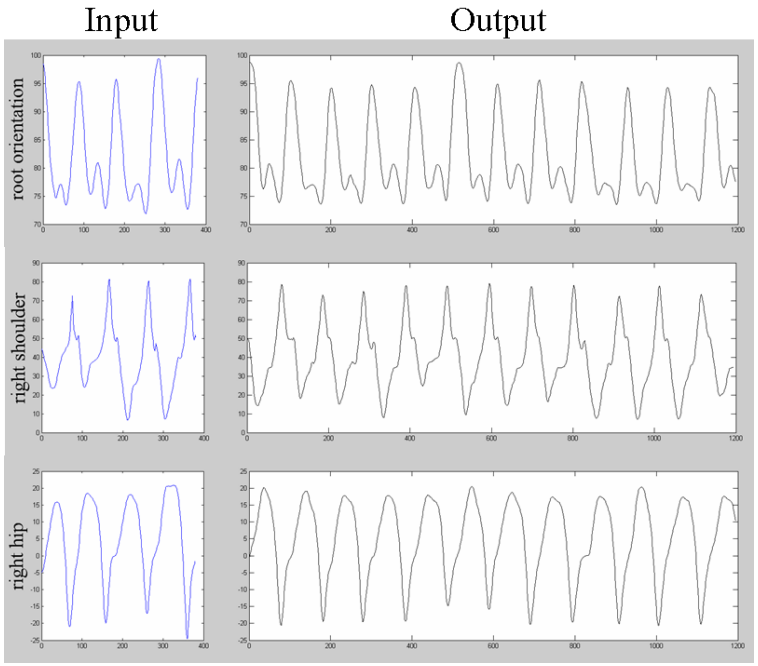

Figure 8: We show plots of selected DOFs of the swimming breast stroke motion. We see that each new cycle of the output retains the general shape of the input data, but none of them is an exact copy of the input.

The amount of data and memory required for our model is small. The synthesis process is also efficient once a model structure has been learned.

Memory. The memory required to store the learned structure is small compared to the memory required to store the input motions. Since our method is non-parametric, we have to keep the original input motions. Our model only requires four input motions of a particular type of motion for learning. It is important that we can learn with a small number of inputs. If we have a large number of these motions already, it might be possible to just randomly select one each time and play them back. Hence the memory storage is essentially the four input motions. Once a model is learned, we can also keep just one of these input motions. This allows us to keep the memory requirement to a minimum, as we can keep just one of the inputs and generate many variations from it. However, it is better to use more than just one input, as the newly synthesized motions will be variations of the inputs. There would be more variations in the new motions if we use more inputs.

Performance Time. It takes between one and two hours to learn the DBN structure for each type of motion we described above. This learning process can be done offline. The process of synthesizing new motions can be done efficiently: in our examples, it takes about 0.1 seconds to generate 1 second of motion. It is interesting to note that 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). This is important in that it allows the runtime synthesis process to be efficient.

We assume that the DBN has a 2nd-order Markov property. The justification is that we tried the algorithm by assuming a 1st-order Markov property, and the learned structure does not produce good motions at all. The 2nd-order is therefore the simplest model that works well. It is possible

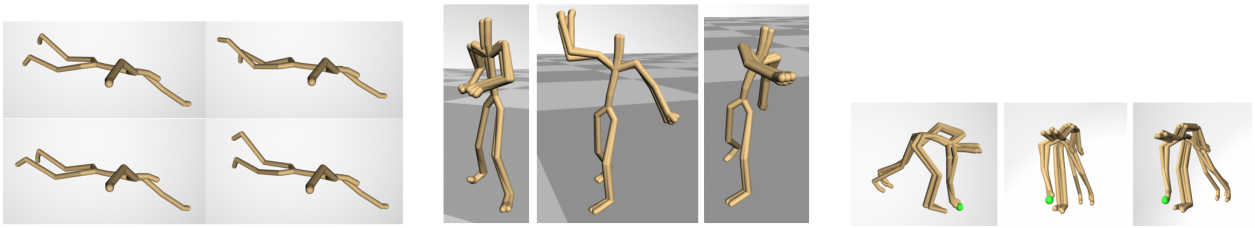

Figure 9: **Left:** Poses from four of the newly generated free style motions. Note that the arms are synchronized but the legs are not, as expected. **Middle:** Selected poses from the four new football throwing motions. The poses are selected at similar time phases of the throw. **Right:** We overlay the poses at the end of the four new motions. These are 3 separate views of the same pose. Note that the hands can reach the same position at the end, while the motion trajectory (seen from the video) can be different.

to use a more complicated model (3rd-order or higher). We did not try more complicated models, although we believe that they would produce similar results while taking a longer runtime.

7 Discussion

We have presented a model for synthesizing variations in human motion. 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 particular motion, our model can be used to generate new variations of the motion. After training for the model structure, we can even take one instance of the motion, and generate variations of it. We demonstrate our method by showing that the new variations have different timings and poses distinct from the input motion data. 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.

One limitation of our method is that it requires at least a few examples of a particular motion in order to generate a model for that motion. These examples have to be similar because we are modeling the variation of similar motions. Our method, however, does not require a large number of examples.

One interesting area for future work is to provide a method for the user to control the variation that is generated. One of the challenges 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" what kinds of output motions we get 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 detecting or constraining the output to lie with the "natural" range of movement.

It is important to highlight the differences between our method and interpolation methods [20,

25]. Interpolation methods generate new motions that are "in between" the original examples. Our method models a probability distribution of the original examples. If we synthesize a large number of new motions, these motions follow the learned distribution. Hence the majority of the motions will be close to the "mean" motion of the original examples. Our method also produces motions that have different timings than the input motions. This is difficult to generate with interpolation methods, which usually require a manual time synchronization process of the input motions to begin with. Finally, interpolation methods require at least two example motions. Given a learned structure, our model can synthesize new variations from just one example motion. Our method is also different from physically-based methods [13, 21] in that there is no guarantee we can generate physically-valid motions. Our method is based on learning the statistical properties of input data; we can generate motions that are statistically similar to the data.

Another area 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.

References

[1] Okan Arikan and D. A. Forsyth. Interactive motion generation from examples. *ACM Transactions on Graphics*, 21(3):483–490, July 2002.

[2] Bobby Bodenheimer, Anna V. Shleyfman, and Jessica K. Hodgins. The effects of noise on the perception of animated human running. In N. Magnenat-Thalmann and D. Thalmann, editors, *Computer Animation and Simulation 1999*, pages 53–63. Springer-Verlag, Wien, September 1999.

[3] Matthew Brand and Aaron Hertzmann. Style machines. In *SIGGRAPH 2000*, pages 183–192, 2000.

[4] Jason Ernst, Oded Vainas, Christopher T Harbison, Itamar Simon, and Ziv Bar-Joseph. Reconstructing dynamic regulatory maps. *Molecular Systems Biology*, page 3:74, 2007.

[5] Nir Friedman, Kevin Murphy, and Stuart Russell. Learning the structure of dynamic probabilistic networks. *Uncertainty in Artifical Intelligence*, pages 139–147, 1998.

[6] Zoubin Ghahramani. Learning dynamic bayesian networks. *Lecture Notes in Computer Science*, 1387, 1998.

[7] C. M. Harris and D. M. Wolpert. Signal-dependent noise determines motor planning. *Nature*, 394:780–784, August 1998.

[8] Eugene Hsu, Kari Pulli, and Jovan Popović. Style translation for human motion. In *SIGGRAPH 2005*, pages 1082–1089, 2005.

[9] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. *ACM Transactions on Graphics*, 21(3):473–482, July 2002.

[10] Lucas Kovar, Michael Gleicher, and John Schreiner. Footskate cleanup for motion capture editing. In *ACM SIGGRAPH/Eurographics symposium on Computer animation (SCA) 2002*, pages 97–104, 2002.

[11] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics*, 21(3):491–500, July 2002.

[12] Yan Li, Tianshu Wang, and Heung-Yeung Shum. Motion texture: a two-level statistical model for character motion synthesis. In *SIGGRAPH 2002*, pages 465–472, 2002.

[13] C. Karen Liu, Aaron Hertzmann, and Zoran Popović. Learning physics-based motion style with nonlinear inverse optimization. *ACM Transactions on Graphics*, 24(3):1071–1081, August 2005.

[14] Rachel McDonnell, Simon Dobbyn, and Carol O'Sullivan. Crowd creation pipeline for games. In *International Conference on Computer Games*, pages 181–190, 2006.

[15] Richard E. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2003.

[16] Ken Perlin. Real time responsive animation with personality. In *Transactions on Visualization and Computer Graphics*, pages 5–15, 1995.

[17] Katherine Pullen and Christoph Bregler. Animating by multi-level sampling. In *Proceedings of the Computer Animation*, pages 36–42. IEEE Computer Society, 2000.

[18] Katherine Pullen and Christoph Bregler. Motion capture assisted animation: Texturing and synthesis. *ACM Transactions on Graphics*, 21(3):501–508, July 2002.

[19] Katherine Pullen and Christoph Bregler. Synthesis of cyclic motions with texture (unpublished), 2002.

[20] Charles Rose, Michael F. Cohen, and Bobby Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, 18(5):32–41, 1998.

[21] Alla Safonova, Jessica K. Hodgins, and Nancy S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics*, 23(3):514–521, August 2004.

[22] Mankyu Sung, Michael Gleicher, and Stephen Chenney. Scalable behaviors for crowd simulation. *Computer Graphics Forum*, 23(3):519–528, September 2004.

[23] Mankyu Sung, Lucas Kovar, and Michael Gleicher. Fast and accurate goal-directed motion synthesis for crowds. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 291–300, July 2005.

[24] Daniel Thalmann, Laurent Kermel, William Opdyke, and Stephen Regelous. Crowd and group animation (siggraph course notes). 2005.

[25] Douglas J. Wiley and James K. Hahn. Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Applications*, 17(6):39–45, 1997.