An Evolutionary Approach for Learning Motion Class Patterns

Meinard Müller[1], Bastian Demuth[2], and Bodo Rosenhahn[1]

[1] Max-Planck-Institut für Informatik
Campus E1-4, 66123 Saarbrücken, Germany
meinard@mpi-inf.mpg.de
[2] Universität Bonn, Institut für Informatik III
Römerstr. 164, 53117 Bonn, Germany

Abstract. This article presents a genetic learning algorithm to derive discrete patterns that can be used for classification and retrieval of 3D motion capture data. Based on boolean motion features, the idea is to learn motion class patterns in an evolutionary process with the objective to discriminate a given set of positive from a given set of negative training motions. Here, the fitness of a pattern is measured with respect to precision and recall in a retrieval scenario, where the pattern is used as a motion query. Our experiments show that motion class patterns can automate query specification without loss of retrieval quality.

1 Introduction

Motion capture or mocap systems [3] allow for tracking and recording of human motions at high spatial and temporal resolutions. The resulting 3D mocap data is used for motion analysis and synthesis in fields such as sports sciences, biomechanics, and computer animation [2, 4, 9]. Furthermore, in computer vision, it has also been used as prior knowledge for human tracking [10–12]. Even though there is a rapidly growing corpus of freely available mocap data, there still is a lack of efficient motion retrieval systems that work in a purely content-based fashion without relying on manually generated annotations. Here, the main difficulty is due to the fact that similar types of motions may exhibit significant spatial as well as temporal variations [2, 4]. To cope with spatial variations, Müller et al. [6] have introduced the concept of *relational features* that capture semantically meaningful boolean relations between specified points of the kinematic chain underlying the mocap data. For example, such features may express whether a hand is raised or not or whether certain parts of the body such as legs are bent or stretched, see Fig. 1a. Furthermore, to cope with temporal variations, a feature-dependent temporal segmentation is used by merging adjacent motion frames that satisfy the same boolean relations, see Fig. 1b. Motion retrieval can then be performed very efficiently by using standard index-based string matching techniques, see [6]. One remaining major problem is that the retrieval performance heavily depends on the query formulation, which involves manual and time-consuming specification of a query-dependent feature selection.

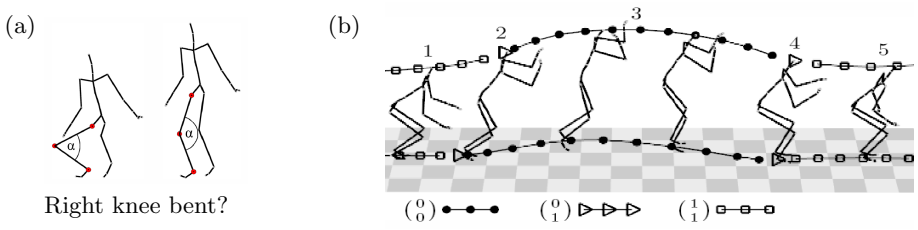

Fig. 1. (a) Relational feature checking whether the right knee is bent or stretched. **(b)** Segmentation of a parallel leg jumping motion $D = D_{\text{jump}}$ with respect to a combination of the features "left knee bent" and "right knee bent". Poses assuming the same feature values are indicated by identically marked trajectory segments. The trajectories of the body points "top of head" and "right ankle" are shown.

The main contribution of this paper is to overcome this problem by applying a genetic algorithm to learn *motion class patterns* from positive and negative training motions. Such a pattern can be thought of as an automated, locally adaptive feature selection for the motion class represented by the positive training motions. In the experiments, we will demonstrate that the retrieval performance of automatically learned patterns is similar to manually selected and heavily tuned patterns. The paper is organized as follows. In Sect. 2, we briefly review the underlying motion retrieval procedure and fix the notation. Then, in Sect. 3, we describe in detail our proposed evolutionary learning algorithm. Finally, in Sect. 4, we summarize our experiments and conclude in Sect. 5 with prospects on future work. Further references will be given in the respective sections.

2 Motion Representation and Retrieval

In this section, we summarize the motion retrieval concept described in [6], while introducing some notation. In the following, a mocap data stream D is regarded as a sequence of data frames, each representing a human pose as illustrated by Fig. 1b. A human pose is encoded in terms of a global position and orientation as well as joint angles of an underlying skeleton model. Mathematically, a relational feature is a boolean function from the set of poses to the set $\{0, 1\}$. In the following, we fix a feature function $F = (F_1, \ldots, F_f)$, which consists of f such relational features. A feature function is applied to a mocap data stream D in a pose-wise fashion. An *F-segment* of D is defined to be a subsequence of D of maximal length consisting of consecutive frames that exhibit the same F-feature values. Picking for each segment one representative feature vector one obtains a so-called *F-feature sequence* of D, which is denoted by $F[D]$. We also represent such a sequence by a matrix $M_F[D] \in \{0, 1\}^{f \times K}$, where K denotes the number of F-segments and the columns of $M_F[D]$ correspond to the feature vectors in $F[D]$. For example, consider a feature function $F = (F_1, F_2)$ that consists of $f = 2$ relational features, the first checking whether the left and the second

checking whether right knee is bent or stretched. Applying F to the jumping motion $D = D_{\text{jump}}$ as shown in Fig. 1b results in the feature sequence

$$F[D] = \left(\binom{1}{1}, \binom{0}{1}, \binom{0}{0}, \binom{0}{1}, \binom{1}{1}\right) \quad \text{and} \quad M_F[D] = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix}, \tag{1}$$

where the five columns correspond to the $K = 5$ motion segments. Such a feature sequence or matrix can then be used as a query for efficient index-based motion retrieval as described in [6].

Note that for the parallel leg jumping motion above, first both knees are bent, then stretched, and finally bent again. This information is encoded by the first, third, and fifth column of $F[D]$. The feature vectors $\binom{0}{1}$ in the second and fourth column arise because the actor does not bend or stretch both legs at the same time. Instead, he has a tendency to keep the right leg bent a bit longer than the left leg. However, in a parallel leg jumping motion, there are many possible transitions from, e.g., $\binom{1}{1}$ (legs bent) to $\binom{0}{0}$ (legs stretched), such as $\binom{1}{1} \to \binom{0}{0}$, $\binom{1}{1} \to \binom{0}{1} \to \binom{0}{0}$, or $\binom{1}{1} \to \binom{1}{0} \to \binom{0}{1} \to \binom{0}{0}$. To account for such irrelevant transitions, Müller et al. [6] introduce some fault tolerance mechanism based on fuzzy sets. The idea is to allow at each position in the query sequence a whole set of possible, alternative feature vectors instead of a single one. In the following, we encode alternative feature values by asterisks in the matrix representation $M_F[D]$. For example, the second column of the matrix $\begin{pmatrix} 1 & * & 0 & * & 1 \\ 1 & * & 0 & 1 & 1 \end{pmatrix}$ encodes the set $V_2 = \left\{\binom{0}{0}, \binom{0}{1}, \binom{1}{0}, \binom{1}{1}\right\}$ of four alternative feature vectors and the fourth column the set $V_4 = \left\{\binom{0}{1}, \binom{1}{1}\right\}$. The asterisks can be used to mask out irrelevant transitions between the alternating vectors $\binom{1}{1}$ and $\binom{0}{0}$ from the original feature sequence.

More generally, we consider matrices $X \in \{0, 1, *\}^{f \times K}$, which are referred to as *motion class patterns*. As indicated by the example above, the k^{th} column $X(k)$ of X encodes a subset $V_k \subseteq \{0, 1\}^f$ of alternative feature vectors, also referred to as *fuzzy set*. Furthermore, X encodes a sequences $\boldsymbol{V}(X) := (V_1, \ldots, V_K)$ of fuzzy sets, also referred to as *fuzzy query*. Such fuzzy queries[3] can be efficiently processed using an index that is based on inverted lists, see [6].

3 Genetic Learning Algorithm

Let $\mathcal{T}^+$ be a set of positive training motions representing a certain class of semantically related motions. For example, $\mathcal{T}^+$ may consist of several parallel leg jumping motions performed by different actors and in various styles. Furthermore, let $\mathcal{T}^-$ be a set of negative training motions that are not considered to be in the motion class of interest. For example, $\mathcal{T}^-$ may contain one leg jumping motions, jumping jacks, or walking motions. The overall goal is to learn a motion class pattern that suitably characterizes the motion class represented by $\mathcal{T}^+$. In this section, we present a genetic algorithm to automatically derive such

[3] Note that one can enforce the admissible condition needed in the matching algorithm by taking suitable complements and unions of neighboring fuzzy sets, see [4, 6].

Input:
- $\mathcal{T}^+$ Set of positive training motions.
- $\mathcal{T}^-$ Set of negative training motions.
- F Feature function consisting of f components $F_1, \ldots, F_f$.
- p Size of the population, where $p > f$.
- μ Rate of mutation.
- r Number of parents used in the recombination step.
- s Number of fittest offsprings used in the next generation, where $s < p$.
- G Number of generations.

Initialization: Compute indices $I^F_{\mathcal{T}^+}$ and $I^F_{\mathcal{T}^-}$ for the training sets, see [6].
Choose an initial population $\Pi^{(0)}$ of size $|\Pi^{(0)}| = p$.

Evolution: Repeat the following procedure for $g = 0, \ldots, G-1$:
(1) Compute the fitness of the individuals in the population $\Pi^{(g)}$.
(2) Select r individuals as parents based on universal stochastic sampling.
(3) Generate $r(r-1)/2$ offsprings by pairwise recombination of the parents.
(4) Modify offspring via mutation with respect to the mutation rate μ.
(5) Compute the fitness of all resulting offspring and pick the s fittest one.
(6) Replace the s individuals in $\Pi^{(g)}$ exhibiting the lowest fitness by the s offsprings picked in (5). The resulting population is denoted by $\Pi^{(g+1)}$.

Output: Fittest individuum in $\Pi^{(G)}$.

Fig. 2. Genetic algorithm for learning motion class patterns.

pattern from $\mathcal{T}^+$ and $\mathcal{T}^-$. Generally, a genetic algorithm is a population-based optimization technique to find approximate solutions to optimization problems, see [8]. In our case, the optimization criterion is based on the retrieval performance in terms of precision and recall when using the motion class pattern as fuzzy query. Fig. 2 gives an overview of our genetic learning algorithm, which is explained in the subsequent subsections. In the following, we fix a feature function $F = (F_1, \ldots, F_f)$ (typically consisting of $10 - 20$ components) and omit F in the notation by writing $M[D]$ instead of $M_F[D]$.

3.1 A Model for Individuals

A population consists of a set of individuals, where each individual is a description of some motion class pattern representing a candidate solution of the optimization problem. In our scenario, an individual Ind is defined to be a tuple $Ind := (D, Feat, Col, Fix)$. Here, $D \in \mathcal{T}^+$ denotes a reference motion and $Feat \subseteq [1:f]$ represents a selection of features comprised in the feature function $F = (F_1, \ldots, F_f)$. From these two parameters, one obtains a motion class pattern $M[D, Feat]$ with entries in $\Sigma = \{0, 1, *\}$ as follows. First, the feature sequence matrix $M[D] \in \{0,1\}^{f \times K}$ is modified by replacing all entries with a row index in $[1:f] \setminus Feat$ by the entry $*$. Then, any coinciding consecutive columns are replaced by a single column yielding the matrix $M[D, Feat]$ having L columns, $L \leq K$. The component Col is a subset $Col \subseteq [1:L]$, which models the transition segments. Furthermore, the component Fix is a map $Fix : Col \mapsto 2^{Feat}$ that assigns to each column $\ell \in Col$ a subset $Fix(\ell) \subseteq Feat$. Intuitively, the function Fix is used to fix and blend out certain entries in $M[D, Feat]$. More precisely, all entries of $M[D, Feat]$ having some column index $\ell \in Col$ and some row index in $Feat \setminus Fix(\ell)$ are replaced by the entry $*$. The resulting matrix is denoted by $M[Ind]$. For example, let $K = 6$, $f = 4$, and $Feat = \{1, 3, 4\}$. Then, for the

pattern $M[D]$ given in (2), one obtains $L = 5$. Furthemore, let $Col = \{2, 4\}$ with $Fix(2) = \emptyset$ and $Fix(4) = \{1, 4\}$. Then one obtains the following patterns $M[D, Feat]$ and $M[Ind]$:

$$M[D] = \begin{pmatrix} 0&0&0&0&0&0 \\ 1&1&1&0&1&0 \\ 1&0&0&0&1&1 \\ 1&1&0&1&1&1 \end{pmatrix}, \quad M[D, Feat] = \begin{pmatrix} 0&0&0&0&0 \\ *&*&*&*&* \\ 1&0&0&0&1 \\ 1&1&0&1&1 \end{pmatrix}, \quad M[Ind] = \begin{pmatrix} 0&*&0&0&0 \\ *&*&*&*&* \\ 1&*&0&*&1 \\ 1&*&0&1&1 \end{pmatrix} \tag{2}$$

3.2 Fitness Function

We now define a fitness function *Fit* that measures the quality of a given individual. First, an individual *Ind* is transformed into a fuzzy query $\boldsymbol{V} := \boldsymbol{V}(M[Ind])$. The resulting fuzzy query is evaluated on the union $\mathcal{T} := \mathcal{T}^+ \cup \mathcal{T}^-$. The fitness function assigns a high fitness to the individual if the associated fuzzy query leads to a high recall as well as a high precision. More precisely, let $\mathcal{H}(Ind) \subset \mathcal{T}$ denote the subset of motion documents retrieved by $\boldsymbol{V}$. The recall value R and the precision value P with respect to $\boldsymbol{V}$ are defined as

$$R := R(Ind) := \frac{|\mathcal{H}(Ind) \cap \mathcal{T}^+|}{|\mathcal{T}^+|} \quad \text{and} \quad P := P(Ind) := \frac{|\mathcal{H}(Ind) \cap \mathcal{T}^+|}{|\mathcal{H}(Ind)|}. \tag{3}$$

Obviously, $0 \leq R \leq 1$ and $0 \leq P \leq 1$. Note that a high recall value R implies that many motions from $\mathcal{T}^+$ are retrieved by $\boldsymbol{V}$, whereas a high precision value $P(Ind)$ implies that most motions from $\mathcal{T}^-$ are rejected by $\boldsymbol{V}$. It is the objective to identify fuzzy queries, which simultaneously maximize R and P.

As described later, our genetic algorithm will be initialized by individuals typically revealing high recall but low precision values. In the course of the evolutionary process more and more specialized individuals will enter the population. Therefore, at the beginning of the process the improvement of the precision values is of major concern, while with increasing generations the recall values gain more and more importance. This motivates the following definition of a fitness function, which depends on the number g of generations already performed:

$$Fit(g, Ind) := P(Ind) \cdot R(Ind)^{\min(4, \sqrt{g})}. \tag{4}$$

for $g \in \mathbb{N}$. Note that $0 \leq Fit(g, Ind) \leq 1$. The formula $\min(4, \sqrt{g})$ in the exponent of $R(Ind)$ puts an increasing emphasis on the recall value for increasing g and has been determined experimentally.

3.3 Initialization

For the start, we generate an initial population $\Pi^{(0)}$ of size $p > f$. First, we define f individuals $Ind_n = (D_n, Feat_n, Col_n)$, $n \in [1 : f]$, based on some randomly chosen but fixed reference motion $D_n := D_0 \in \mathcal{T}^+$. Furthermore, we set $Feat_n := \{n\}$ and $Col := \emptyset$ with trivial Fix. The entries of $M[Ind_n]$ have the value $*$ except for the entries in row n, which alternately assume the values 0 and 1. For the remaining $p - f$ elements of $\Pi^{(0)}$, we generate individuals $Ind = (D, Feat, Col, Fix)$ with a randomly chosen reference motion $D \in \mathcal{T}^+$, a randomly chosen set *Feat* consisting of two elements in $[1 : f]$, and $Col = \emptyset$.

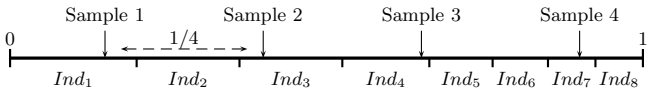

Fig. 3. An example for universal stochastic sampling selecting $r = 4$ individuals from a population of size $p = 8$. The selected individuals are Ind_1, Ind_3, Ind_4, and Ind_7.

3.4 Genetic Operations

We now describe the genetic operations of selection, recombination, and mutation, which are needed to breed a new population from a given population.

Selection. In each step of our genetic algorithm, we select r individuals (the parents), which are used to create new individuals (the offsprings). To increase the possibility of obtaining offsprings of high fitness, one should obviously revert to parents that reveal the highest fitness values in the current population. On the other hand, to avoid an early convergence towards some poor local optimum, it is important to have some genetic diversity within the parents used in the reproduction process. Therefore, we use a selection process known as *stochastic universal sampling*, see [8]. Let $\Pi^{(g)}$ denote the current population. Then all individuals of $\Pi^{(g)}$ are arranged in a list $(Ind_1, \ldots, Ind_p)$ with decreasing fitness. The interval $[0, 1]$ is partitioned into p subintervals, where the kth subinterval, $1 \leq k \leq p$, has a length corresponding to the proportion of the fitness value of Ind_k to the sum of the fitness values over all individuals contained in $\Pi^{(g)}$. To select r individuals from $\Pi^{(g)}$, the interval $[0, 1]$ is sampled in some equidistant fashion, where two neighboring samples have a distance of $1/r$. Furthermore, the first sample is randomly chosen with the interval $[0, 1/r]$, see Fig. 3. Each sample determines an individual according the subinterval it is contained in. That way, each individual of $\Pi^{(g)}$ has a probability that is proportional to its relative fitness value to be selected for reproduction.

Recombination. In the recombination step two parent individuals are used to derive a new offspring individual by combining and mixing the properties (the genes) of the parents. In this process, we need a suitable notion of randomly choosing subsets from a given set. For a finite set A, we first randomly generate a natural number n with $1 \leq n \leq |A|$ by suitably rounding the output of some normally distributed random variable with expectation value μ and standard deviation σ. We then form a subset B of size n by randomly picking n elements of A with respect to a uniform distribution on A. For short, we write $B \subseteq_{\text{rand}}^{(\mu,\sigma)} A$. In our recombination procedure, we use the parameters $\mu = 0.7|A|$ and $\sigma = 0.3|A|$. For these parameters, we will simply write $B \subseteq_{\text{rand}} A$. We now describe how to recombine two parents $Ind_1 := (D_1, Feat_1, Col_1, Fix_1)$ and $Ind_2 := (D_2, Feat_2, Col_2, Fix_2)$ in order to obtain a new offspring $Ind := (D, Feat, Col, Fix)$. The following list defines the recombination operator depending on the degree of correspondence of the two parents.

1. Suppose $D_1 = D_2$, $Feat_1 = Feat_2$, and $Col_1 = Col_2$. Then set $D := D_1$, $Feat := Feat_1$, $Col := Col_1$, and define Fix by randomly choosing

$Fix(\ell) \subseteq_{\text{rand}} Fix_1(\ell) \cup Fix_2(\ell)$ for $\ell \in Feat$. In case this results in $Fix(\ell) = Feat$, the element ℓ is removed from Col and Fix is suitably restricted.

2. Suppose $D_1 = D_2$, $Feat_1 = Feat_2$, and $Col_1 \neq Col_2$. Then set $D := D_1$, $Feat := Feat_1$, and choose $Col \subseteq_{\text{rand}} Col_1 \cup Col_2$. Finally, define Fix by $Fix(\ell) := Fix_1(\ell)$ for $\ell \in Col \cap Col_1$ and $Fix(\ell) := Fix_2(\ell)$ for $\ell \in Col \setminus Col_1$.
3. Suppose $D_1 = D_2$ and $Feat_1 \neq Feat_2$. Then set $D := D_1$ and chose $Feat \subseteq_{\text{rand}} Feat_1 \cup Feat_2$. Note that $Feat_1$, $Feat_2$, and $Feat$ generally induce different segmentations on D, which prevents to directly transfer properties encoded by the parameters Col and Fix from the parents to the offspring. To transfer at least part of the information, we proceed as follows. Suppose D is segmented into K segments with respect to $Feat_1$, then let $(1 = s_1 < s_2 < \ldots < s_K)$ be the indices of the start frames of the segments. Similarly, let $(1 = t_1 < t_2 < \ldots < t_L)$ be the start indices in the segmentation of D induced by $Feat$. Then all $\ell \in [1 : L]$ with $s_k = t_\ell$ for some $k \in Col_1$ are added to Col and $Fix(\ell) := Fix_1(k)$. In other words, the offspring inherits all Fix_1-properties from the first parent that refer to segments having the same starting frame with respect to $Feat_1$ and $Feat$. Similarly, the offspring also inherits all Fix_2-properties from the second parent.
4. Suppose $D_1 \neq D_2$. Then randomly set $D := D_1$ or $D := D_2$, and chose $Feat \subseteq_{\text{rand}} Feat_1 \cup Feat_2$. Finally, define $Col := \emptyset$ with trivial Fix.

Mutation. The concept of mutation introduces another degree of randomness in the reproduction process to ensure that all potential individuals have some probability to appear in the population thus avoiding an early convergence towards some poor local optimum. Each of the offspring individuals $Ind := (D, Feat, Col, Fix)$ are further modified in the following way. First, a number $d \in \mathbb{N}_0$ of modifications is determined by rounding the output of some exponentially distributed random variable with expectation value μ (in our experiments we chose $\mu = 2$). Then, one successively performs d basic mutations, where the type of modification is chosen randomly from the following list of five types:.

1. The reference motion D is replaced by a randomly chosen motion in $\mathcal{T}^+$. Furthermore, Col is replaced by the empty set with trivial Fix.
2. In case $Feat \neq [1 : f]$ (otherwise perform Step 3), the set $Feat$ is extended by an additional element, which is randomly chosen from $[1 : f] \setminus Feat$. Note that this leads to a refined segmentation. Therefore, the parameters Col and Fix are adjusted by suitably relabeling segment indices.
3. In case $|Feat| > 1$ (otherwise perform Step 2), an element is randomly removed from $Feat$. This may lead to a coarsened segmentation. In this case, Col and Fix are adjusted as described in Step 3 of the recombination.
4. Let L be the number of columns of $M[Ind]$. In case $|Col| < L - 1$ (otherwise perform Step 5), the set Col is extended by an additional element ℓ randomly chosen from $[1 : L] \setminus Col$ and Fix is extended by setting $Fix(\ell) := \emptyset$.
5. In case $Col \neq \emptyset$ (otherwise perform Step 4), an element $\ell \in Col$ is chosen randomly and the set $Fix(\ell)$ is extended by an element randomly chosen from $Feat \setminus Fix(\ell)$. In case this results in $Fix(\ell) = Feat$, the element ℓ is removed from Col and Fix is suitably restricted.

3.5 Breeding the Next Generation

To create the population $\Pi^{(g+1)}$ from the population $\Pi^{(g)}$, we proceed as follows. First, r parent individuals are selected from $\Pi^{(g)}$. Any two of these r parent individuals are recombined to yield $\frac{r(r-1)}{2}$ offsprings, which are then mutated individually. Then the fitness values are computed for the resulting offsprings. Finally, the s fittest offsprings are picked to replace the s individuals in $\Pi^{(g)}$ that exhibit the lowest fitness.

4 Experiments

For our experiments, we used the HDM05 motion database that consists of several hours of systematically recorded motion capture data [7]. From this data, we manually cut out suitable motion clips and arranged them into 64 different classes. Each such motion class (MC) contains 10 to 50 different realizations of the same type of motion, covering a broad spectrum of semantically meaningful variations. For example, the motion class "Parallel Leg Jump" contains 36 variations of a jumping motion with both legs. The resulting *motion class database* $\mathcal{D}^{\mathrm{MC}}$ contains $1,457$ motion clips, amounting to 50 minutes of motion data.

We now describe one of our experiments conducted with 15 motion classes, see Table 1. For each of these motion classes, we automatically generated a set $\mathcal{T}^+$ of positive training motions consisting of one third of the available example motions of the respective class (e. g. 12 motions in case of "Parallel Leg Jump"). Similarly, $\mathcal{T}^-$ was generated by randomly choosing one third of the motions of the other 14 classes. Depending on the respective motion class, we either used a feature function F_ℓ or a feature function F_u similar to the ones described in [6]. F_ℓ has 11 components characterizing the lower body, whereas F_u has 12 components characterizing the upper body. In our experiments, the following parameters turned out to be suitable: we chose a population size of $p = 50$ and used $r = 7$ parents for the recombination. All resulting $s = 28$ offsprings were used for replacement. Furthermore, $G = 50$ generations turned out to yield a good convergence.

With these parameters, our evolutionary algorithm required in average 7.2 seconds for each generation (using MATLAB 6.5 run on an Athlon XP 1800^+). Therefore, using $G = 50$, it took roughly six minutes to compute one motion class pattern. For each of the 15 motion classes, we performed the entire algorithm ten times. To determine the quality, the resulting patterns were used as fuzzy queries and evaluated on the entire motion class database $\mathcal{D}^{\mathrm{MC}}$. Table 1 summarizes the retrieval results in terms of precision and recall. The first precision-recall (PR) pair of each of the 15 motion classes indicates the average PR values over the ten patterns. The second and third pairs show the best and worst PR values among the ten patterns. Finally, the fourth pair shows the PR values of a manually optimized query specification by a retrieval expert indicating what seems to be achievable by a manual query process. For example, in case of the class "Elbow-to-knee" in average 25.6 of the 27 correct motions with a precision of

	Elbow-to-knee	Cartwheel	Jumping Jack	Parallel Leg Jump	One Leg Jump
					
recall (average)	25.6/27 = 0.95	19.3/21 = 0.92	49.5/52 = 0.95	25.3/36 = 0.70	40.0/42 = 0.95
precision (average)	0.92	0.83	0.99	0.56	0.94
recall (best)	26/27 = 0.96	21/21 = 1.00	52/52 = 1.00	28/36 = 0.78	41/42 = 0.98
precision (best)	26/26 = 1.00	21/21 = 1.00	52/52 = 1.00	28/44 = 0.64	41/43 = 0.95
recall (worst)	26/27 = 0.96	18/21 = 0.86	47/52 = 0.90	18/36 = 0.50	39/42 = 0.93
precision (worst)	26/33 = 0.79	18/199 = 0.09	47/48 = 0.98	18/28 = 0.64	39/42 = 0.93
recall (manual)	24/27 = 0.89	21/21 = 1.00	51/52 = 0.98	21/36 = 0.58	33/42 = 0.79
precision (manual)	24/26 = 0.92	21/21 = 1.00	51/55 = 0.93	21/34 = 0.62	33/182 = 0.18
	Hit-on-Head	Right Kick	Sit-down	Lie-down	Rotate Arms
					
recall (average)	10.9/13 = 0.84	26.0/30 = 0.87	14.2/20 = 0.71	15.7/20 = 0.78	14.6/16 = 0.91
precision (average)	0.82	0.65	0.85	0.89	0.95
recall (best)	11/13 = 0.85	26/30 = 0.87	15/20 = 0.75	18/20 = 0.90	16/16 = 1.00
precision (best)	11/13 = 0.85	26/28 = 0.93	15/16 = 0.94	18/19 = 0.95	16/16 = 1.00
recall (worst)	10/13 = 0.77	20/30 = 0.67	13/20 = 0.65	11/20 = 0.55	14/16 = 0.88
precision (worst)	10/16 = 0.63	20/49 = 0.41	13/16 = 0.81	11/28 = 0.39	14/16 = 0.88
recall (manual)	12/13 = 0.92	25/30 = 0.83	16/20 = 0.80	19/20 = 0.95	16/16 = 1.00
precision (manual)	12/14 = 0.86	25/41 = 0.61	16/40 = 0.40	19/21 = 0.90	16/16 = 1.00
	Climb Stairs	Walk	Walk Backwards	Walk Sideways	Walk Cross-over
					
recall (average)	26.1/28 = 0.93	14.5/16 = 0.91	11.4/15 = 0.76	14.8/16 = 0.93	11.1/13 = 0.85
precision (average)	0.96	0.43	0.52	0.97	0.78
recall (best)	27/28 = 0.96	15/16 = 0.94	12/15 = 0.8	15/16 = 0.94	13/13 = 1.00
precision (best)	27/27 = 1.00	15/30 = 0.50	12/19 = 0.63	15/15 = 1.00	13/16 = 0.81
recall (worst)	25/28 = 0.89	14/16 = 0.88	12/15 = 0.80	14/16 = 0.88	11/13 = 0.85
precision (worst)	25/29 = 0.86	14/52 = 0.27	12/43 = 0.28	14/14 = 1.00	11/22 = 0.50
recall (manual)	22/28 = 0.79	15/16 = 0.94	8/15 = 0.53	16/16 = 1.00	10/13 = 0.77
precision (manual)	22/23 = 0.96	15/33 = 0.45	8/22 = 0.36	16/17 = 0.94	10/13 = 0.77

Table 1. Retrieval results for motion retrieval based on ten automatically learned motion class patterns for each class (average, best, worst) for motion retrieval based on an optimized manual query specification (manual).

0.92 were recovered by the motion class patterns. The best pattern could recover 26 of the 27 correct motions with a precision of 1.00. In case of the optimized manual query specification, we obtained a recall of $24/27 = 0.89$ and a precision of 0.92. From our experiments, we can say that for a large number of whole body movements our automatically learned motion class patterns are capable to produce qualitatively similar retrieval results as manual feature selection and query specifications.

5 Conclusions

In this paper, we have presented an evolutionary approach for learning motion class patterns, which offer an automated alternative to manual query specification without loss of retrieval quality. We introduced a novel genetic learning algorithm with a model for individuals, rules for selection, recombination and mutation as well as a suitable fitness function. Opposed to previous automated query specification approaches as described in [5], our motion class patterns can be directly plugged-in into the index-based matching scenario of [6] affording very efficient motion retrieval. For the future, we plan to employ similar genetic algorithms for automated keyframe selection as required in [5]. Another application we have in mind is the combination of efficient retrieval methods with learning approaches in order to generate suitable prior knowledge as needed to stabilize and support human motion tracking [1].

References

1. T. Brox, B. Rosenhahn, D. Cremers, and H.-P. Seidel. Nonparametric density estimation with adaptive anisotropic kernels for human motion tracking. In *Proc. 2nd Workshop on Human Motion*, LNCS **4814**:152–165, Springer, 2006.
2. L. Kovar and M. Gleicher. Automated extraction and parameterization of motions in large data sets. *ACM Trans. Graph.*, 23(3):559–568, 2004.
3. T. B. Moeslund, A. Hilton, and V. Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104(2):90–126, 2006.
4. M. Müller. *Information Retrieval for Music and Motion*. Springer, 2007.
5. M. Müller and T. Röder. Motion templates for automatic classification and retrieval of motion capture data. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 137–146. ACM Press, 2006.
6. M. Müller, T. Röder, and M. Clausen. Efficient content-based retrieval of motion capture data. *ACM Trans. Graph.*, 24(3):677–685, 2005.
7. M. Müller, T. Röder, M. Clausen, B. Eberhardt, B. Krüger, and A. Weber. Documentation: Mocap Database HDM05. Computer Graphics Technical Report CG-2007-2, Universität Bonn, 2007.
8. H. Pohlheim. *Evolutionäre Algorithmen: Verfahren, Operatoren und Hinweise*. Springer, 1999.
9. B. Rosenhahn, R. Klette, and D. Metaxas. *Human Motion Understanding, Modeling, Capture, and Animation*. Springer, 2007.
10. H. Sidenbladh, M. J. Black, and L. Sigal. Implicit probabilistic models of human motion for synthesis and tracking. In A. Heyden, G. Sparr, M. Nielsen, and P. Johansen, editors, *Proc. European Conference on Computer Vision*, LNCS **2353**:784–800, Springer, 2002.
11. C. Sminchisescu and A. Jepson. Generative modeling for continuous non-linearly embedded visual inference. In *Proc. Int. Conf. on Machine Learning*, 2004.
12. R. Urtasun, D. J. Fleet, and P. Fua. 3D people tracking with Gaussian process dynamical models. In *Proc. International Conference on Computer Vision and Pattern Recognition*, 238–245, IEEE Computer Society Press, 2006.