# Activity Discovery Using Compressed Suffix Trees

Prithwijit Guha[1], Amitabha Mukerjee[2], and K.S. Venkatesh[2]

[1] TCS Innovation Labs, New Delhi
prithwijit.guha@tcs.com
[2] Indian Institute of Technology, Kanpur,
{amit,venkats}@iitk.ac.in

**Abstract.** The area of unsupervised activity categorization in computer vision is much less explored compared to the general practice of supervised learning of activity patterns. Recent works in the lines of activity "discovery" have proposed the use of probabilistic suffix trees (PST) and its variants which learn the activity models from temporally ordered sequences of object states. Such sequences often contain lots of object-state self-transitions resulting in a large number of PST nodes in the learned activity models. We propose an alternative method of mining these sequences by avoiding to learn the self-transitions while maintaining the useful statistical properties of the sequences thereby forming a "compressed suffix tree" (CST). We show that, on arbitrary sequences with significant self-transitions, the CST achieves a much lesser size as compared to the polynomial growth of the PST. We further propose a distance metric between the CSTs using which, the learned activity models are categorized using hierarchical agglomerative clustering. CSTs learned from object trajectories extracted from two data sets are clustered for experimental verification of activity discovery.

## 1 Introduction

Activities are the manifestations of transitions of spatio-temporal relations of the scene objects among themselves and/or the scene (Figure 1); and are modeled as sequences of object appearance/motion features – e.g. human pose sequences, vehicle trajectories etc. Such features are often selected using domain specific priors (e.g. human body models using articulated cylinders/ellipsoids, stick models etc. for gesture recognition [1]) or constructed directly from object features obtained from image data (e.g. human contour descriptors using Point Distribution Models [2]).

The selection of these object states dictate the classes of the activities that they might describe. However, we further require efficient statistical sequence modeling techniques for representing significant temporal patterns from the time-series data of the object states. Existing literature on activity analysis is vast and is mostly based on the supervised framework where, the hidden Markov models [3] and its variants (e.g. parameterized HMM [4], coupled HMM [5] etc.)

G. Maino and G.L. Foresti (Eds.): ICIAP 2011, Part II, LNCS 6979, pp. 69–78, 2011.

(a) (b) (c) (d) (e)

**Fig. 1.** Activity discovery – (a)-(d) Multiple object tracking performed on traffic video to extract (e) object trajectories which form important descriptors of object actions. These trajectories are modeled as *compressed suffix trees* (Section 2) which are clustered (Section 3) further to discover the action categories.

have been widely used for activity analysis. In comparison, much less has been explored in the domain of unsupervised activity modeling and recognition. An early work in this direction was proposed through categorizing object actions by trajectory clustering [6]. The use of variable length Markov models (VLMM) in the domain of activity analysis was introduced in [2] for modeling interactions. These approaches propose to perform a vector quantization over the object feature space to generate temporally indexed object-state sequences from video data. These sequences are parsed further to learn probabilistic suffix trees (PST) leading to the discovery of behavioral models of varying temporal durations. A good overview on contributions in such prior free activity modeling can be found in [7]. In the similar lines, activities have been modeled by stochastic context free grammars (SCFG) from sequences of scene landmarks traversed by the object [8]. These approaches have so far focused only on the activity modeling using VLMM, SCFG or PST. The issue of clustering such models have been addressed in [9] where activities were modeled from object state sequences in terms of PST which were further categorized through a graph based clustering using a similarity measure between the PSTs.

Our work develops along the lines of the approaches of "discovery" rather than supervised learning from manually labeled exemplars. We observe that the time indexed object descriptor sequences used for activity modeling are mostly populated with self-transitions. We argue that learning such self-transitions into probability suffix trees has minimal usage in the context of activity structure discovery and hence propose to learn "compressed suffix trees" (CST, Section 2). We further propose a semi-metric measure of dissimilarity between two CSTs (Section 3) and the *activity discovery* is performed by grouping the learned models using average linkage hierarchical agglomerative clustering. The experimental results on 2 data sets are presented for object actions modeled as sequences of quantized motion directions (Section 4).

## 2 Compressed Suffix Tree

Activity analysis applications have traditionally used the Hidden Markov Model (HMM) or its variants for classifying temporal patterns of action descriptors.

However, from the viewpoint of activity "*discovery*", this approach has two significant drawbacks – first, the supervised learning framework of HMM limits it from discovering the temporal pattern structures of the activities and second, the first order Markovian assumption adopted in the HMM fails to capture the essence of variable length constituent sub-events of the activities. This led the researchers to use variants of sequence mining algorithms such as the variable length Markov model [2], stochastic context free grammar [8] and probabilistic suffix tree [9] which showed satisfactory performance in discovering variable length structures of activities.

Given an input symbol sequence like "*a a a b b c*", a sequence mining algorithm aims at finding all sub-sequences of varying length and frequency – say, "*a*" occurred thrice, "*b*" twice, "*a a*" twice, "*a b*" once etc. When applied to activity discovery, these symbols are nothing but the object states - e.g. quantized motion directions of traffic objects, human body poses, spatio-temporal relations between interacting objects etc.

We observe that, the sequence mining algorithms encode the self-transitions as well – i.e. the self-transition $a \rightarrow a$ is also learned along with $a \rightarrow b$. In most practical cases, however, the same state may continue for a long time in such activity descriptor sequences. For example, consider the action of a car turning – moving westwards in the scene for 140 frames, north-westwards for 10 frames and northwards for the next 100 frames. Mining this motion direction sequence will result to the discovery of a lot of variable length sub-sequences like "westwards-westwards", "northwards-northwards-northwards" etc.

Object state transitions plays a major role in defining the temporal structure of an activity. Thus, we believe that learning only the transitions between different states is more useful compared to the maintenance of information regarding the self transitions. The direct advantage of avoiding to learn the self-transitions is reflected in the memory saving as the sub-sequences like $a \rightarrow a \rightarrow a$, $b \rightarrow b$ etc. need not be stored any more. This involves the removal of the redundant symbol repetitions from the original sequence by editing it to a "*self-transitionless*" sequence. Thus, We propose to perform sequence mining by avoiding self-transitions but preserving the relative frequencies of the symbols themselves and their transition sub-sequences. A formal description of the proposed approach of learning the "compressed suffix tree" (CST) and its operational dissimilarity from a sequence mining algorithm (say, PST) is presented next.

Consider a temporally ordered symbol sequence $\mathbf{S} = \{\epsilon(t), t = 1, \ldots\}$ where the symbols belong to some alphabet $\mathcal{E}$. The results of sequence mining are represented by a tree $\mathcal{T}$ rooted at a node $\rho$, say. Each node of $\mathcal{T}$ is a 2-tuple $\mathbf{T}_n \equiv (\epsilon, \pi)$ containing the symbol $\epsilon \in \mathcal{E}$ and a real number $\pi \in (0, 1]$ signifying the probability of occurrence of the path $\{\rho, \ldots \mathbf{T}_n\}$ among the set of all possible paths of the same length. However, the sequence mining is performed to discover variable length sequences only up to a certain maximum size $L$ to maintain finite size of the tree. In other words, the maximum depth of the tree is $L$.

The process of sequence mining initializes with an empty (first in first out) buffer $\beta(0)$ (of length $L$, the maximum allowable sequence length) and the null tree $\mathcal{T}(0)$ (containing only the root node $\rho$). At the $(t-1)^{th}$ instant ($t > L$), the buffer will contain the most recent $L$ symbols from the sequence $\mathbf{S}$, i.e. $\beta(t-1) = \{\epsilon(t-L), \ldots, \epsilon(t-2), \epsilon(t-1)\}$. As the next symbol $\epsilon(t)$ is pushed to the buffer, the oldest symbol in the buffer is thrown out, i.e. $\beta(t) = \{\epsilon(t-L+1), \ldots \epsilon(t-1), \epsilon(t)\}$. Thus, the buffer ($\beta$) acts as a moving window (of size $L$) over the symbol stream $\mathbf{S}$. In case of a PST, the symbol $\epsilon(t)$ is directly pushed to the buffer $\beta$. However, in the proposed approach of transition sequence mining for learning a CST, $\epsilon(t)$ is pushed to $\beta$ only in case of a transition, i.e. $\epsilon(t) \neq \epsilon(t-1)$.

At every instant, the variable length sequences observed within the buffer are learned in $\mathcal{T}$. For example, if a buffer $\beta$ of size 4 contain symbols as $\beta = \{\ a,\ a,\ b,\ c\ \}$, then the 4 variable length sequences $c$, $b\ c$, $a\ b\ c$ and $a\ a\ b\ c$ are learned as branches in $\mathcal{T}$. Let the set of $l$-length paths (originating from $\rho$) of the tree $\mathcal{T}(t)$ at the $t^{th}$ instant be $B^{(l)}(t) = \{\alpha_u^{(l)}(t), u = 1, \ldots b_l\}$, where $b_l$ is the number of $l$-length branches in the tree. Now, if the $l$-length sequence in the buffer matches with the $b^{th}$ path of $B^{(l)}(t)$, then the probabilities $\{\pi_u^{(l)}(t), u = 1, \ldots b_l\}$ of the nodes of $\mathcal{T}(t)$ at the $l^{th}$ depth are updated as $\pi_u^{(l)}(t) = (1 - \eta_l(t))\pi_u^{(l)}(t-1) + \eta_l(t)\delta(u-b)$ where, $\eta_l(t)$ is the rate of learning $l$-length sequences at the $t^{th}$ instant and $\delta(\bullet)$ is the Kronecker delta function. However, in the current implementation a fixed learning rate $\eta$ is employed such that $\eta_l(t) = max\left(\frac{1}{t-l+1}, \eta\right) \forall l$.

The occurrence of a new symbol (with a transition in case of CST) results in the formation of newer variable length sequences in the buffer. Thus, new nodes signifying this symbol are added at different depths of the tree thereby growing newer branches. Each new node is initialized with an initial probability of $\eta_l(t)$, whereas the older node probabilities in the same depths are penalized by multiplying with a factor of $(1 - \eta_l(t))$. This ensures the self-normalizing nature of node probability updates such that they add up to 1.0 at each depth.

While learning these sequences, for the CST the probability of the 3 symbol sequence stays at 1.0 as it does not encounter any other transition sequence other than "$a\ b\ c$" till $t = 6$. On the other hand, the PST learns all the ordered sequences (of length 1/2/3 symbols) including the self-transitions. For example, at $t = 6$, the CST has only 6 nodes while the PST has 11 nodes. Thus, in practical applications of activity sequence modeling, where objects continue in the same state for long durations of time, the PST will learn a huge number of branches mostly due to self-transitions but the CST will remain compact in size for learning only branches with symbol-transitions. The memory efficiency of the CST is however reflected for higher values of $L$ and longer sequences.

We illustrate this by performing (transition) sequence mining on 100 artificial symbol sequences of sizes between 900 – 1100. The symbols are drawn randomly from an alphabet of size 20. Any drawn symbol is repeated a random number of times (between 50 and 150) and the sequence is constructed accordingly. We learn both CST and PST from this sequence by varying the maximum depth of

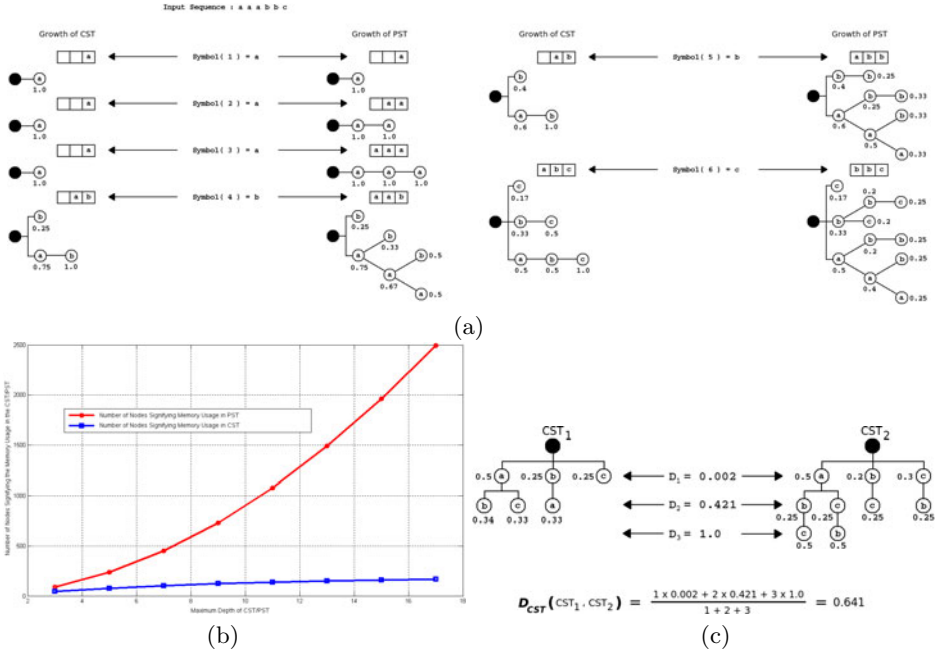


**Fig. 2.** (a) Learning the CST and the PST with a maximum depth of 3 from an input sequence of 6 symbols – "*a a a b b c*". The first three symbols being the same, the CST learns only a single node ("*a*" , 1.0 ) whereas the PST learns 3 nodes, i.e. ("*a*", 1.0 ), ("*a a*" , 1.0 ) and ("*a a a*" , 1.0). A transition is registered by the CST at $t = 4$ (new symbol: "*b*") and it learns two single symbol sequences ("*a*",0.75) and ("*b*",0.25) (*a* occurred thrice and *b* once in four symbols) and one 2-length sequence ("*a b*", 1.0). However, the PST learns two single symbol sequences ("*a*",0.75) and ("*b*",0.25); two sequences of size 2, i.e. ("*a a*", 0.67) and ("*a b*", 0.33); and, two sequences of length 3, i.e. "*a a a*" and "*a a b*" with probabilities 0.5 each. The learning process continues in the similar manner till $t = 6$. Note that, at $t = 6$, the CST has only 6 nodes while the PST has 11 nodes. (b) Memory usage in CST and PST for different tree depths. The maximum depth of the trees are varied from 3 to 17 in steps of 2. As the maximum depth of the tree increases, the number of nodes grow almost linearly in the CST as compared to the polynomial growth in case of the PST. (c) Illustrating the computation of distance between CSTs.

the tree ($L$) from 3 to 17 in steps of 2. The number of nodes added to the trees signify their memory usage during the process of (transition) sequence mining. Figure 2(b) shows the average number of nodes added to the CST/PST while the maximum depth of the tree is varied. As the maximum depth of the trees increase, the number of nodes grow almost linearly in the CST as compared to the polynomial growth in the case of PST. Notably, the CST presents a memory efficient alternative to the PST while mining sequences with large number of self-transitions as is frequently observed in the area of activity analysis.

Note that, the CST preserves the distribution of the symbols in the input, i.e. the probability distributions at the first depth of both CST and PST are the same and encodes the information relevant to transitions while using much lesser memory as compared to the PST. However, the PST maintains certain stochastic properties of the input sequences like variable order conditional probabilities and thus its branches can also be used as variable length symbol predictors. However, none of the earlier unsupervised activity analysis approaches have used these predictor properties as they were of no use in the context of activity structure discovery [2].

These stochastic properties are certainly lost in the CST as during the process of learning only transitions we are effectively editing the input sequence by removing redundant repetitions. However, we still maintain the relative frequencies of the states themselves and the self-transitionless sub-sequences. For example, in case of our input sequence "*a a a b b c*", we do not perform a buffer update operation at $t = 5$ when the symbol "*b*" repeats (Figure 2(a)), but we learn the fact that the transition sequence "*a b*" has occurred 2 times. This information is relevant in discriminating sequences with similar symbol transition structures but different symbol repetition durations. For example, the sequences "*a a a b b c*" and "*a a b b b c c*" have similar transition structure $a \rightarrow b \rightarrow c$ but different symbol repetitions. The procedure for computing (dis)similarities between symbol sequences modeled as CST are described next.

## 3 CST Clustering

We have proposed to learn CST from object state sequences where each tree as a whole characterizes each activity. To cluster these activities, we need to define a measure between two CSTs to reflect the dissimilarity between the activities that they have modeled. Motivated by the fact that the CST hosts probability distributions at each of its depth, we propose to use the Bhattacharya distance between two probability distributions $\mathcal{P}_1, \mathcal{P}_2$, given by $\mathcal{D}_b(\mathcal{P}_1, \mathcal{P}_2) = 1 - \sum_i \sqrt{\mathcal{P}_1(i)\mathcal{P}_2(i)}$. We define the dissimilarity $\mathcal{D}_{cst}(\mathcal{T}_1, \mathcal{T}_2)$ between the two CSTs $\mathcal{T}_1$ and $\mathcal{T}_2$ as $\mathcal{D}_{cst}(\mathcal{T}_1, \mathcal{T}_2) = \frac{\sum_{l=1}^{L} g(l)\mathcal{D}_b(\mathcal{P}_1^{(l)}, \mathcal{P}_2^{(l)})}{\sum_{l=1}^{L} g(l)}$ where $\mathcal{P}_k^{(l)}$ is the probability distribution hosted at the $l^{th}$ depth of the $k^{th}$ tree ($k = 1, 2$) and $g(l)$ is a monotonically increasing positive function of the depth $l$ (in our case $g(l) = l$). The proposed dissimilarity measure (Figure 2(c)) exploits the distributions of all variable length subsequences of a temporal pattern and we choose to assign more weight to the longer sub-sequences as compared to the shorter ones to give more importance to the structure of the whole event rather than it's minute details (shorter sub-sequences). However, note that $\mathcal{D}_{cst}$ is only a semi-metric as it does not satisfy the triangular inequality. CSTs learned from time indexed object state sequences are further grouped using average linkage hierarchical clustering algorithm whose performance depends on the number of clusters. We next present the methodology adopted for computing the clustering sensitivity.

### 3.1 Clustering Performance Analysis

Consider the case of clustering $N$ activities belonging to $M$ different categories, such that the $m^{th}$ category contain $N_m$ activity instances ($N = \sum_{m=1}^{M} N_m$). Consider an unsupervised classification algorithm to form $Q$ clusters over these activities. We can form a $M \times Q$ "*cluster-category distribution matrix*" ($CCDM$) such that, $CCDM[m][q]$ ($m = 1, \ldots M$, $q = 1, \ldots Q$) denotes the number of activities of the $m^{th}$ category which belong to the $q^{th}$ cluster. The category label $l(q)$ of the $q^{th}$ cluster is assigned based on the maxima of the frequencies of the activity categories present in it. Thus, if instances of the $i^{th}$ category are present with a maximum frequency in the $q^{th}$ cluster, then $l(q) = i = argmax_j CCDM[j][q]$. This makes us to consider all the other activities in the $q^{th}$ cluster which do not belong to $l(q) = i$ as false detections with respect to the $i^{th}$ category.

A $M \times M$ confusion matrix ($CM$) of (unsupervised) classification can be constructed from $CCDM$ using the cluster-category labels. Let, $CM[r][k]$ ($r, k = 1, \ldots M$) denote the number of activities actually belonging to the $r^{th}$ class, which have been classified to be belonging to the $k^{th}$ category. It can be shown that the confusion matrix entries can be computed as $CM[r][k] = \sum_{q=1}^{Q} CCDM[r][l(q) - k]$. The sensitivity $S(Q)$ of unsupervised classification for $Q$ number of clusters is defined as the fraction of the total number of activities which are classified correctly and is thus computed as $S(Q) = \frac{\sum_{r=1}^{M} CM[r][r]}{N}$. Sensitivity curves of activity categorization can be obtained by varying the desired number of clusters $Q$ in the hierarchical agglomerative clustering.

## 4 Results

We present our results on two outdoor surveillance data sets – first, the PETS2000 data set and second, a traffic video data set (Figure 1). We illustrate our results mainly on the PETS2000 data set as it contains a smaller number of objects and show only performance analysis results on the traffic video.

Pixel-wise mixture of Gaussian based scene background modeling is used to detect the scene objects as foreground blobs. Object features (color distribution and motion model) learned from these blobs are used to track multiple objects across the images. We have used the multi-object tracking algorithm proposed in [10] to extract the object trajectories[1].

For the purpose of characterizing object actions in terms of the image data, we use a characterization of the image plane motion of the object (presented below). The motion directions in the image plane are quantized to form 8 motion states as in compass directions – $M_1$ denoting "eastwards", $M_2$ signifying "north-east" and so on going anti-clockwise to $M_8$ representing the "south-east" direction.

[1] Due to space constraints, we do not provide the details of foreground extraction and multiple object tracking algorithm for trajectory extraction.

Additionally, we use the state $M_0$ in case the object is at rest. Object trajectories in image space form an important aspect of single object actions. We represent the trajectory as a sequence of quantized motion direction states. CSTs are learned over the motion direction sequences of the objects which form the motion behavior model. The CSTs are further subjected to average linkage hierarchical agglomerative clustering algorithm for discovering action categories.

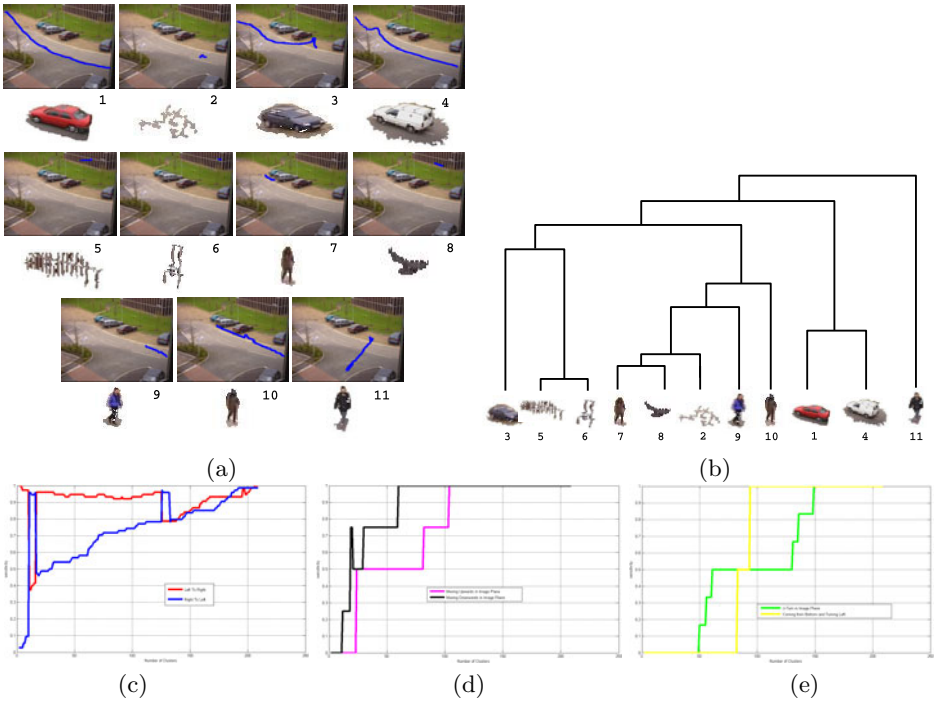


**Fig. 3.** Trajectory clustering – (a) Trajectories of objects extracted from PETS2000 data set. Note that the noise objects #2 , #5, #6 and persons #7, #9 have very short scene presences and insignificant trajectories. (b) The dendogram formed by hierarchical agglomerative clustering of the CST trees learned from these trajectories. Note that the trajectories of persons #7, #9 and #10 are distinct from those of the cars #1 and #4. However, Car #3's parking trajectory distinguishes it from the other cars. (c)–(e) Trajectory clustering sensitivity vs. number of clusters (traffic video). For clarity, the graphs are shown for two image plane trajectory categories at a time – (c) LEFT TO RIGHT and RIGHT TO LEFT; (d) MOVING UPWARDS and MOVING DOWNWARDS; (e) U-TURN and COMING FROM BOTTOM AND TURNING LEFT.

Foreground blob detection followed by multiple object tracking is used to extract 11 objects from the PETS2000 data set whose trajectories are shown in figure 3(a). The dendogram formed by clustering the CST learned from quantized motion direction state sequences (obtained from object trajectories) is shown in

figure 3(b). The red and white colored cars (objects #1 and #4) had almost similar trajectory which entered the scene from the right and drove to exit through the left boundary are clustered properly. Also, the objects #7, #8, #2, #9 and #10 had their trajectories directed from left to right albeit with varying frequencies and are seen to cluster in the same group. Also, it is worth noting that the trajectory (walking downwards in the scene and turning back) of object #11 (person in black dress) remained salient from the others in the process of clustering.

The processes of blob detection and tracking are used to extract 209 objects from the traffic video. The image plane trajectories of these objects characterize their actions. Manual inspection of these image plane trajectories show the existence of 6 different categories along with spurious ones (MISC) arising due to track losses – $\Gamma(\text{TRAJECTORY}) = \{$ LEFT TO RIGHT (75), RIGHT TO LEFT (74), MOVING UPWARDS (4), MOVING DOWNWARDS (4), U-TURN (3), COMING FROM BOTTOM AND TURNING LEFT (5), MISC (44) $\}$. CSTs are learned from the temporally ordered sequences of the quantized motion directions obtained from the trajectories which are further subjected to hierarchical agglomerative clustering. The classification sensitivities of each category are computed by the performance analysis procedure outlined in sub-section 3.1. The classification sensitivities for varying number of clusters are shown in figures 3(c)–(e).

Note that the sensitivity computed by our evaluation criterion increases with the number of clusters in hierarchical agglomerative clustering (figure 3(c)–(e)). However, the trajectory categories of LEFT TO RIGHT and RIGHT TO LEFT appear as the leading or most frequent members of the clusters (Sub-section 3.1) with even lower number of clusters. Also, the trajectories of the categories MOVING UPWARDS and MOVING DOWNWARDS show high sensitivities at lower number of clusters even with lower frequency of appearance. We believe that the trajectories MOVING UPWARDS and MOVING DOWNWARDS stand out as they do not have motion states in common with LEFT TO RIGHT or RIGHT TO LEFT. On the other hand, the trajectories of U-TURN and COMING FROM BOTTOM AND TURNING LEFT show low sensitivities on account of high sub-structural similarities with the other four trajectory categories.

## 5 Conclusion

We have proposed to model activity descriptor sequences using "*compressed suffix trees*" (CST) which are shown to be advantageous over the existing formulations of using the "*probabilistic suffix trees*" (PST). This has direct advantages in memory efficient model construction thereby avoiding the learning of large number of self-transitions present in activity descriptor sequences. It is shown that the use of CST is advantageous in cases of higher order activity modeling as the CST shows an almost linear growth in memory usage while the PST grows polynomially in size (number of nodes) as the maximum allowable sequence length is increased. We have further proposed a distance metric in the space of CSTs to perform hierarchical agglomerative clustering of activity models. The efficiency of the proposed approach is experimentally verified on 2 data

sets where activities are represented as quantized object motion direction state sequences.

Activities are generally described as temporally ordered sequences of the object states. However, another informative component of an activity is the number and nature of the participants in that event. Our future work aims at characterizing activities through the participant category (e.g. humans ride a bike), number of participants (group activity – people boarding a bus; single object action – running, walking etc. and two-object interactions – handshake, following, overtaking etc.) along with the state space features (pose/velocity/proximity etc.) of the participants. Discovery of such information beyond the analysis of only activity structures (i.e. models learned from time indexed object descriptor sequence) in a purely unsupervised framework will advance us a few steps further towards the key goals of a cognitive vision system.

## References

1. Moeslund, T., Hilton, A., Kruger, V.: A survey of advances in vision-based human motion capture and analysis. Computer Vision and Image Understanding 104, 90–126 (2006)
2. Galata, A., Johnson, N., Hogg, D.: Learning variable-length markov models of behavior. Computer Vision and Image Understanding 81, 398–413 (2001)
3. Gao, J., Hauptmann, A.G., Bharucha, A., Wactlar, H.D.: Dining activity analysis using a hidden markov model. In: IEEE International Conference on Pattern Recognition, pp. 915–918 (2004)
4. Bobick, A., Wilson, A.: A state-based technique for summarization and recognition of gesture. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 382–388 (1995)
5. Brand, M., Oliver, N., Pentland, A.: Coupled hidden markov models for complex action recognition. In: Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition, pp. 1–6 (1997)
6. Johnson, N., Hogg, D.: Learning the distribution of object trajectories for event recognition. In: British Machine Vision Conference, pp. 583–592 (1995)
7. Buxton, H.: Learning and understanding dynamic scene activity: a review. Image and Vision Computing 21, 125–136 (2003)
8. Veeraraghavan, H., Papanikolopoulos, N., Schrater, P.: Learning dynamic event descriptions in image sequences. In: IEEE International Conference on Computer Vision and Pattern Recognition (2007)
9. Hamid, R., Maddi, S., Bobick, A., Essa, M.: Structure from statistics - unsupervised activity analysis using suffix trees. In: IEEE International Conference on Computer Vision (2007)
10. Guha, P., Mukerjee, A., Venkatesh, K.S.: Efficient occlusion handling for multiple agent tracking with surveillance event primitives. In: Second Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance (2005)