Activity Recognition with Finite State Machines

Wesley Kerr and **Anh Tran** and **Paul Cohen**
Computer Science Department
University of Arizona
{wkerr, trananh, cohen}@cs.arizona.edu

Abstract

This paper shows how to learn general, Finite State Machine representations of activities that function as recognizers of previously unseen instances of activities. The central problem is to tell which differences between instances of activities are unimportant and may be safely ignored for the purpose of learning generalized representations of activities. We develop a novel way to find the "essential parts" of activities by a greedy kind of multiple sequence alignment, and a method to transform the resulting alignments into Finite State Machine that will accept novel instances of activities with high accuracy.

1 Introduction

This work develops a representation of the structure of activities that is learnable and is general enough to correctly recognize previously unseen instances of activities. Generalizing over instances of activities requires a way to discard "inessential" elements of instances. This is the central problem addressed in this paper. Section 2 describes how we represent instances of activities and Section 3 presents a method for generalizing over instances. Section 3.3 shows how to represent these generalizations as Finite State Machines which may be used as recognizers. Section 4 presents empirical results on recognition accuracy. Related work is summarized in Section 5.

2 Representing Instances of Activities

We describe activities in terms of propositions that have truth values. For example, when approaching a box, the proposition *move-forward(agent)* is true at times. We can represent an activity as a *propositional multivariate time series* (PMTS), a two-dimensional array of binary elements, where $P_{i,j} = [0, 1]$ means that proposition P_i is false or true (0 or 1) at time j. An interval $[j, j + \tau]$ during which a proposition is true is called a fluent.

To illustrate our representations (and to keep further examples short), consider four fluents that might occur when an agent approaches a box: **C** = *collision(agent,box)*, **D** = *distance-decreasing(agent,box)*, **F** = *move-forward(agent)*, **S** = *speed-decreasing(agent)*, **V** = *visible(object,agent)*. A PMTS representation of *approach* is shown in Figure 1. In the first instant (the first column of the array), the *visible(object,agent)* and *move-forward* fluents are initated. In the next, the *distance-decreasing(agent-box)* is initiated. This fluent continues until the penultimate instant. In the final instant, the *collision(agent,box)* fluent is initiated.

C	00000000000000000001
D	01111111111111111110
F	11111111111000000000
S	00000000000111111110
V	11111111111111111111

Figure 1: The PTMS for an instance of an *approach*.

This example is very simple. In general, we model activities in terms of dozens of propositions, each of which may occur during several contiguous intervals, or fluents. We also have achieved good results with real-valued propositions, $P_{i,j} \in \Re$ (Sec. 4).

If we will be satisfied with an *ordinal* time scale for patterns — a scale the preserves the order of changes to a proposition's status but not the durations of fluents — then we can compress the PMTS into a *compressed bit array* (CBA) structure by removing identical consecutive columns, as shown in Figure 2. The purpose of this compression is less to save space than to produce an abstraction of the PMTS in which patterns of changes that are identical but for their durations are represented identically. Figure 3 is a CBA representation of the PMTS in Figure 1 .

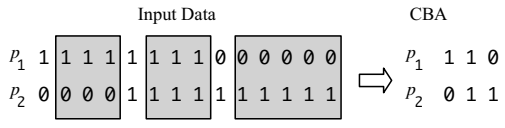

Figure 2: Removing consecutive identical columns in a bit array produces a compressed bit array (CBA).

3 Learning General Signatures of Activities

In many domains, we will observe variations in instances of activities. Even simple activities, such as an agent approach-

C	0001
D	0110
F	1100
S	0010
V	1111

Figure 3: The CBA representation for the *approach* example.

ing an object, may vary. Perhaps the object will not be visible at the outset, or become occluded during the activity. Perhaps the agent won't slow down as it nears the object. In general, instances of an activity will not have identical PMTSs or CBAs. Some might include fluents that others do not include, or the temporal relationship between fluents in one might be different in another. These differences are increasingly likely as we increase the number of propositions in the PMTSs.

If our only task is to accurately classify instances of activities, then the variations in instances need not be a problem. However, our approach is to extract the "essence" or "prototype" of any activity from instances, not merely classify activities.

Our approach will be to learn generalizations of instances of activities. These generalizations will eventually be represented as finite state machines, and recognizing an activity will be treated as transitions through a finite state machine to reach an accepting state (Sec. 3.3). But before we get to this representation, we will describe how to extract generalizations of activities represented as CBAs.

We write a CBA for an instance of an activity as a *qualitative sequence*, then use sequence alignment to find a *signature* that best fits all the instances of the activity.

3.1 From CBAs to Qualitative Sequences

Qualitative sequences have two useful properties: They represent the temporal relationships between fluents, and they are canonical in the sense that instances of an activity with identically ordered temporal relationships between fluents will have identical qualitative sequence representations.

Relationships between fluents can be described by *Allen relations* [Allen, 1983]. Allen recognized that, after eliminating symmetries, there are only seven possible relationships between two fluents, shown in Figure 4. Allen relations are qualitative in the sense that they represent the temporal order of events, specifically, the beginnings and endings of fluents, but not the durations of fluents.

The fluents in an instance may be sorted to generate a canonical representation of the instance. Ordered fluents (also called *normalized* fluents in [Winarko and Roddick, 2007]), are sorted according to earliest end time. If two fluents finish at the same time, they are further sorted by earliest start time, and if the start and end times are identical, then they are sorted alphabetically by proposition name.

After ordering fluents as we just described, we construct the Allen relations between *all* of the pairs of fluents in order, as described in Algorithm 1. An illustrative instance and the resulting qualitative sequence is shown in Table 1. The letters **F, D, S** and **C** denote propositions, and an assertion such as (**F** 0 11) means that proposition **F** was true in the interval [0,11).

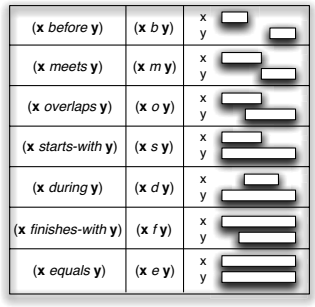

Figure 4: Allen Relations

Algorithm 1: MAKE-SEQUENCE(I)

```
S = ()
for i = 1 to size(I) do
   for j = (i + 1) to size(I) do
      S ← S + allen(I[i], I[j])
   end for
end for
return S
```

A qualitative sequence can be shortened by defining a window that determines how close any two intervals must be to consider one *before* the other. This is known as the *interaction window*. Assume that we have two intervals $(p_1\ s_1\ e_1)$ and $(p_2\ s_2\ e_2)$ such that p_1 and p_2 are proposition names, s_1 and s_2 are start times, e_1 and e_2 are end times, and $e_1 < e_2$. The two intervals are said to interact if $s_2 \leq e_1 + w$ where w is the interaction window. If we set $w = 1$ in Table 1, then we would remove the relation (**F** *before* **C**) .

Intervals	Qualitative Sequence
	(**F** *overlaps* **D**)
(**F** 0 11)	(**F** *meets* **S**)
(**D** 1 20)	(**F** *before* **C**)
(**S** 11 20)	(**D** *finishes-with* **S**)
(**C** 20 21)	(**D** *meets* **C**)
	(**S** *meets* **C**)

Table 1: An instance of an activity comprising four intervals, and the corresponding qualitative sequence.

3.2 From Qualitative Sequences to Signatures

Let $\mathcal{E}_c = \{S_1, S_2, \ldots, S_k\}$ be a set of qualitative sequences with the same activity label, $\mathcal{C}$. We define the *signature* of the activity label, $\mathcal{S}_c$, as an ordered sequence of *weighted* Allen relations. (The only difference between a signature and a qualitative sequence is these weights.)

We select a sequence at random from $\mathcal{E}_c$ to serve as the initial signature, $\mathcal{S}_c$, and initialize all of its weights to 1. After this, $\mathcal{S}_c$ is updated by combining it with the other sequences in $\mathcal{E}_c$, processed one at a time.

Two problems are solved during the processing of the sequences in $\mathcal{E}_c$. First, the sequences are not identical, so $\mathcal{S}_c$ must be constructed to represent the most frequent relations in the sequences. The weights in $\mathcal{S}_c$ are used for this purpose. Second, because a relation can appear more than once in a sequence S_i, there can be more than one way to align S_i with $\mathcal{S}_c$. These problems are related because the frequencies of relations in $\mathcal{S}_c$ depend on how sequences are successively aligned with it.

Updating the signature $\mathcal{S}_c$ with a sequence S_i occurs in two phases. In the first phase, S_i is optimally aligned with $\mathcal{S}_c$. The alignment algorithm, described below, penalizes candidate alignments for elements in $\mathcal{S}_c$ that are not matched by elements in S_i, and rewards matches. These penalties and rewards are functions of the weights stored with the signature. In the second phase, the weights in the signature $\mathcal{S}_c$ are updated. If an element in S_i is aligned with one from $\mathcal{S}_c$, then the weight of this element is incremented by one. Otherwise the weight of the element is initialized to one and it is inserted into $\mathcal{S}_c$ at the location selected by the alignment algorithm.

Updating the signature relies on the Needleman-Wunsch global sequence alignment algorithm [Needleman and Wunsch, 1970]. The algorithm uses dynamic programming to find an optimal alignment between two sequences. Alignments are constructed by selecting operators that modify each sequence to look more like the other. Conventionally, sequence alignment is based on three operators: Elements from two sequences may *match*; an element may be *deleted*, or an element may be *inserted*. However, to ensure that no information from sequences in $\mathcal{E}_c$ is lost, we allow the sequence alignment only to insert or match, not delete, elements. The costs of insertion and matching are determined by the weights that are stored with the signature as mentioned above.

$\mathcal{S}_c$ Aligned	S_i Aligned	$\mathcal{S}_c$ Updated	
–	(**C** *meets* **A**)	(**C** *meets* **A**)	1
–	(**C** *before* **B**)	(**C** *before* **B**)	1
–	(**C** *before* **C**)	(**C** *before* **C**)	1
(**A** *finishes* **D**)	–	(**A** *finishes* **D**)	1
(**A** *overlaps* **B**)	(**A** *overlaps* **B**)	(**A** *overlaps* **B**)	6
(**A** *meets* **C**)	(**A** *meets* **C**)	(**A** *meets* **C**)	6
(**D** *overlaps* **B**)	–	(**D** *overlaps* **B**)	1
(**D** *meets* **C**)	–	(**D** *meets* **C**)	1
(**B** *overlaps* **C**)	(**B** *overlaps* **C**)	(**B** *overlaps* **C**)	6

Table 2: Updating the signature $\mathcal{S}_c$.

The process is illustrated in Table 2. Suppose that some sequences have already been aligned with the signature $\mathcal{S}_c$. The sequence in column "S_i Aligned" is first aligned optimally with $\mathcal{S}_c$, as shown; notice that this involves inserting some "empty space" into both $\mathcal{S}_c$ and S_i. Then the two are merged, as shown in column "$\mathcal{S}_c$ Updated," and the weights associated with each Allen relation are updated.

Because the process of updating signatures first aligns and then merges new sequences into $\mathcal{S}_c$, signatures become stuffed with large numbers of elements that occur very infrequently, and thus have low weights. We use a simple heuristic to clean up the signature: After K training instances, all of the relations in the signature with weights less than or equal to n are removed. All of our experiments use $K = 10$ and $n = 3$, meaning that the signature is pruned of all elements occurring fewer than 4 times after a total of 10 training instances. The signature is again pruned after 20 training instances, and so forth.

3.3 From Signatures to Finite State Machines

The CBA in Figure 3 can be viewed as a finite state machine (FSM) in which each column is a state, as shown in Figure 5. However, it represents only a single instance of an activity. What we want is a finite state machine that represents a generalization over multiple instances of an activity. Signatures almost get us there, but signatures are sequences of Allen relations between fluents, not sequences of states comprising one or more fluents. We need a way to transform signatures back into CBA-like sequences of states, where "unimportant" fluents are discarded.

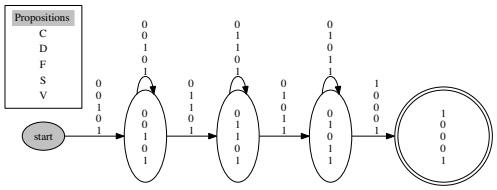

Figure 5: An example of the activity *approach* CBA in Figure 3 represented as FSM recognizer.

Recall that a signature $\mathcal{S}_c$ is built from qualitative sequences $\mathcal{E}_c = \{S_1, S_2, \ldots, S_k\}$. Each S_i is a sequence of Allen relations, $R(p, q)$, between fluents p and q, and each relation $R(p, q)$ has a weight in $\mathcal{S}_c$. For each fluent p, find all of the Allen relations $R(p, q)$ or $R(q, p)$, that is, all the Allen relations in S_i for which p is an argument. For each of these Allen relations, find its weight in the signature $\mathcal{S}_c$. If any of these weights exceeds a threshold, then we "keep" p. But if we cannot find a single Allen relation between p and another fluent whose weight in $\mathcal{S}_c$ exceeds a threshold, then this tells us that p is unimportant in $\mathcal{S}_c$. Not only may it be discarded from the qualitative sequence S_i but, more importantly, it can be discarded from the CBA and PMTS that gave rise to S_i. This procedure gives us a way to remove propositions from CBAs like the one in Figure 3.

Having removed unimportant propositions from the CBAs for each instance of an activity, and represented each of these edited CBAs as a FSM as in Figure 5, we are likely to see that these machines share states. It is then straightforward to build a single FSM that merges the FSMs for individual instances. This merged FSM is used as a *recognizer* for the activity. The FSMs "accept" unlabeled activities as soon as they reach a state which has no further transitions except back to the start state. Every branch in the FSM will lead to one final state reached at the end of the training instance, and all of these states become accepting states for the FSM recognizer.

The number of states in the merged FSM can be further reduced by applying a slightly modified NFA-to-DFA con-

version algorithm on the FSM. The savings in the number of states varies by activity, and reducing the states does not affect the accuracy of the recognizer. An added benefit of the optimization is that it does make the FSM significantly easier to comprehend and manipulate.

4 Evaluation

The experiments reported here involve building FSM recognizers for training instances of activities, then testing how accurately the FSMs recognize previously-unseen activities. An FSM is said to recognize an instance once it reaches any of the accepting states.

For each learned FSM we count true positives, false positives, true negatives, and false negatives. True positives (tp) occur when an FSM for activity i recognizes an instance of activity i. False positives (fp) occur when an FSM for activity i recognizes an instance of activity j. When an FSM for activity i fails to accept an instance of activity j, this is a true negative (tn). A false negative (fn) occurs when an FSM rejects an instance that it should have accepted. Recall is defined as $rec = tp/(tp + fn)$ and precision as $prec = tp/(tp + fp)$. We report the F_1 measure: $2(prec \times rec)/(prec + rec)$.

For each experiment, data are randomly split into training and testing sets of instances. The proportion of the instances put into the training set varied from 10% to 90%. The experiment was ran 100 times for each of the following datasets.

Data

The first dataset is collected from Wubble World 3D (*ww3d*), a virtual environment with simulated physics in which softbots, called wubbles, interact with objects [Kerr *et al.*, 2008]. The simulator collects distances, velocities, locations, colors, sizes, and other sensory information about the world, and represents them as propositions.

We collected a dataset of different activities, each of which consisted of several instances, e.g. several instances of a wubble *jumping over* a box, *jumping on* a box, *approaching* a box, *pushing* a box, *moving around* a box *to the left*, and *moving around* a box *to the right*. Instances were generated by manually controlling a wubble to repeatedly perform one of these activities. Each instance was unique, in that the wubble would start closer or farther from the box, move more or less quickly, and so on. The *ww3d* dataset consists of 37 instances labeled *jump over*, 20 instances labeled *jump on*, and 25 instances for each of the remaining activities. On average, each instance lasts for 445 time steps, and consists of 40 propositions and 73 fluents. This results in qualitative sequences of 2,177 Allen relations, on average.

The second dataset is gathered from a two-dimensional virtual environment – Wubble World 2D (*ww2d*) – in which agents interact with each other, autonomously. We gave the agents just four activities, which we selected to be difficult to discriminate. The agents could engage in *passing*, *colliding* and two kinds of *talking* activities. (The agents do not really talk, but they approach each other and pause for a bit, where presumably a conversation would take place, and then continue on their way.) In one kind of talk, the agents leave the conversation on the same trajectories on which they entered; in the other, they exit in the direction from which they entered. The dataset *ww2d* consists of twenty unique instances for each of the four activities. On average, each instance lasts for 1073 time steps, consists of seven propositions and twelve fluents. The average qualitative sequence length for an instance is 46 Allen relations.

The third dataset comes from an individual's *handwriting*. A single subject was asked to write the letters of the alphabet multiple times [Kerr, 2010]. The subject wrote on a Wacom pen tablet and the x and y positions of the pen were sampled at uniform time intervals. This dataset differs from the others because the data is real-valued, not propositional. We converted the data into propositional time series by applying the SAX algorithm [Lin *et al.*, 2007] as well as an algorithm that outputs the shape of the time series, similar to the SDL library [Agrawal *et al.*, 1995]. In the *handwriting* dataset, there are 21 instances of each of the 26 activities (one for each letter of the alphabet). The average instance lasts over 69 time steps and comprises of 17 propositions and 33 fluents. The average qualitative sequence length for an instance is 304 Allen relations.

4.1 Results

The average recognition performance, summarized by the F_1 score, is shown in Figure 6. Across all datasets, performance increases very quickly as more training data is provided. For example the FSM recognizers constructed on the *ww3d* dataset achieve an F_1 score of 0.9, with narrow confidence intervals, training on as few as four examples of each activity. Activities in *ww2d* focus on the interaction between two agents, and, as noted were designed to be difficult to discriminate. As seen in Figure 6, we are able to reach F_1 scores around 0.8 when averaged across activity. The *handwriting* dataset proved to be the most challenging. One reason for this is that there are many more classes to learn, and the learning algorithm is not discriminative, but, rather, learns common patterns found within the positive examples of activities. Another reason has to do with the composite nature of activities.

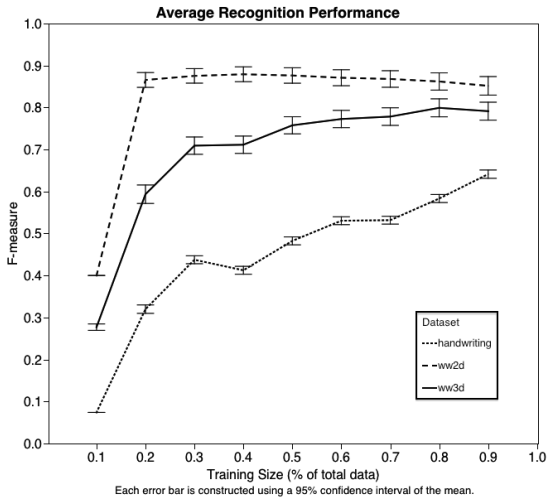

Figure 6: Average F_1 scores for all datasets.

Suppose a test corpus contains instances of activities, but one (or more) of these are components of others. For example, one activity might be *approach*, which is a component of another, such as *jump-over*. Or a looping movement in handwriting could be the letter "a" or a component of the letter "d." If the recognizer for the smaller activity fires during the larger activity, then it will be counted as a false positive. This effect is shown in Figure 7. We present the F_1 score for each of the activities in the *ww3d* dataset. We can clearly see that the F_1 score for the activity *approach* is constant and low, regardless of the amount of training. This happens because all of the activities in *ww3d* have *approach* as a component, so whenever one of these activities happens and the *approach* recognizer fires, it is penalized with a false positive.

One solution to the problem is to ignore all of the activities that are components of more complex activities. This does not solve the problem, it only sweeps it under the rug. A better solution is to track the activities of all the recognizers and combine or compare them over time. This solution is discussed more in Section 6.

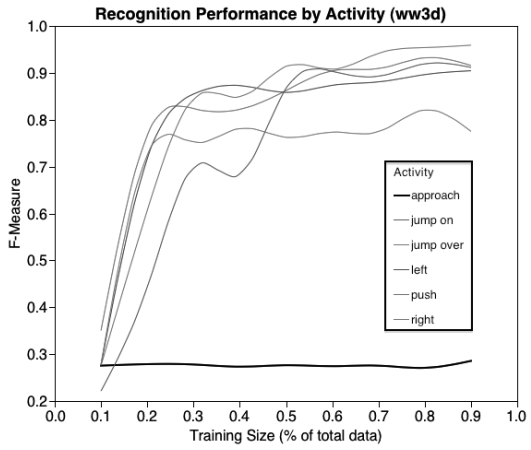

Figure 7: Average F_1 scores for activities in *ww3d*.

5 Related Work

Most of the research on temporal patterns in interval time series focuses on extracting patterns that occur with frequencies greater than some threshold, also known as *support*. The temporal patterns are commonly described using Allen relations and an Apriori-like algorithm [Agrawal and Srikant, 1994] is employed to extract the core structure of the activity by building larger patterns from smaller ones [Winarko and Roddick, 2007; Cohen *et al.*, 2002; Fleischman *et al.*, 2006; Papapetrou *et al.*, 2009].

Two groups have extended pattern mining to classification by explicitly using the most informative patterns to build classifiers [Patel *et al.*, 2008; Batal *et al.*, 2009].

All of the bottom up methods we describe here do not scale well because they generate and evaluate all patterns. This is prohibitive for even modest datasets. For instance, in the *ww3d* dataset, the common temporal pattern for the *jump over* activity consists of twenty different fluents. Previous algorithms cannot handle the enormous number of combinations of fluents (e.g., $\binom{20}{5}$ patterns consisting of only five of 20 fluents) required to find larger patterns.

The problem of online handwriting recognition is an active research problem, and has been so since the beginning of the sixties [Plamondon, 2000]. Special-purpose handwriting recognition methods have achieved very high accuracy (e.g., [Biem, 2006; Jaeger *et al.*, 2001] both achieve accuracy around or above 90%). Other researchers have applied somewhat more general algorithms to the problem; for example, [Bahlmann and Burkhardt, 2004] reported a general, scalable, HMM-based method called *cluster generative statistical dynamic time warping* (CSDTW) that holistically combines cluster analysis and statistical sequence modeling. Our approach is more general, still. It requires no external knowledge of the domain, or structural knowledge such as the number of hidden states in an HMM.

6 Discussion

In this paper, we introduced a new novel approach to activity recognition that uses FSM as the underlying representation. Recognizers are built by first extracting the signature of the activity, which consists of the essential components shared between all training instances of an activity. Once learned, the signature determines which fluents to attend to in order to generate a general FSM recognizer for the activity. The construction of the signature and the FSM employed little domain knowledge, allowing it to work well across several different activity types. In particular, we presented the recognition performance for three different activity types: *ww3d*, *ww2d*, and *handwriting*.

Overall, the recognizers perform with F_1 scores at or above 0.7 across all three datasets. Though the scores are good, we believe that that performance can be improved in two ways. One way to increase scores is to show that the learned FSMs match what human subjects would say about the test instance. For example, many subjects would probably say that *approach* occurs during all of the *jump over* test instances. Another option is to selectively choose a single activity per test instance rather than allowing all recognizers to accept the same test instance.

One possible heuristic for recognizing a single activity per test instance would be to select the FSM recognizer that covers the largest amount of time. A way to achieve this is to track the states of each FSM as it "plays" through an instance, and measure the *depth* of the state furthest away from the start state for each time step. We refer to this as the *information depth* of the recognizer and it should tell us something about the amount of information that the recognizer can "explain" about the instance. Figure 8 shows the information depth measurement for four different FSMs on an example of *jump over* from the *ww3d* dataset. The maximum depth ratio, measured as the ratio between the depth of the furthest active state and the total length of its path to the nearest accepting state, was recorded at each time step of the playback. A ratio of 1 implies that the FSM has reached an accepting state.

In this example, the correct label for the test instance is *jump-over*, and it is correctly recognized by the FSM for *jump*

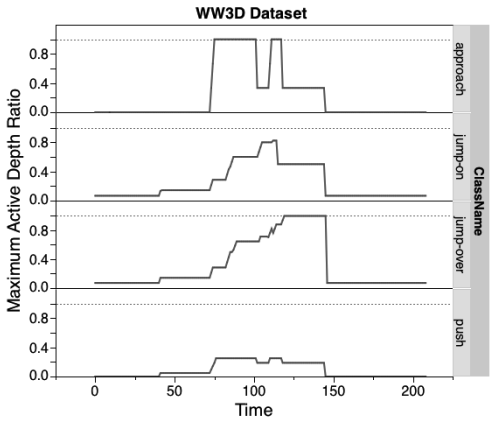

Figure 8: Results of a sample *information depth* experiment for the *ww3d* dataset.

over. In addition, the *approach* recognizer also reached an accepting state for the test, not once, but twice. The difference between the two explanations is that the *approach* recognizer only accounted for about 75 time steps, meaning that the wubble was not approaching the block for 125 time steps. On the other hand, the *jump over* recognizer accommodated most, if not all, of the activity. Selecting the FSM recognizer that covered more of test instance would mean selecting *jump over*, which is the correct activity.

7 Acknowledgements

This work was supported by Defense Advanced Research Projects Agency (DARPA) under contract W911NF1020064. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of DARPA.

References

[Agrawal and Srikant, 1994] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules. In *Proceedings of the 20th VLDB Conference*, 1994.

[Agrawal *et al.*, 1995] Rakesh Agrawal, Giuseppe Psaila, Edward L Wimmers, and Mohamed Zaït. Querying Shapes of Histories. *Proceedings of the 21th International Conference on Very Large Data Bases*, pages 502–514, 1995.

[Allen, 1983] James F. Allen. Maintaining Knowledge About Temporal Intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[Bahlmann and Burkhardt, 2004] Claus Bahlmann and Hans Burkhardt. The Writer Independent Online Handwriting Recognition System frog on hand and Cluster Generative Statistical Dynamic Time Warping. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(3):299–310, 2004.

[Batal *et al.*, 2009] Iyad Batal, Lucia Sacchi, Riccardo Bellazzi, and Milos Hauskrecht. Multivariate Time Series Classification with Temporal Abstractions. *Florida Artificial Intelligence Research Society Conference*, 2009.

[Biem, 2006] Alain Biem. Minimum Classification Error Training for Online Handwriting Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(7):1041–1051, 2006.

[Cohen *et al.*, 2002] Paul R Cohen, Charles Sutton, and Brendan Burns. Learning Effects of Robot Actions using Temporal Associations. *International Conference on Development and Learning*, 2002.

[Fleischman *et al.*, 2006] Michael Fleischman, Phillip Decamp, and Deb K Roy. Mining Temporal Patterns of Movement for Video Content Classification. *Proceedings of the 8th ACM SIGMM International Workshop on Multimedia Information Retrieval*, 2006.

[Jaeger *et al.*, 2001] S. Jaeger, S. Manke, J. Reichert, and A. Waibel. Online handwriting recognition: the npen++ recognizer. *International Journal on Document Analysis and Recognition*, 3:169–180, 2001.

[Kerr *et al.*, 2008] Wesley Kerr, Paul Cohen, and Yu-Han Chang. Learning and Playing in Wubble World. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 66–71, 2008.

[Kerr, 2010] Wesley N. Kerr. *Learning to Recognize Agent Activities and Intentions*. PhD thesis, University of Arizona, 2010.

[Lin *et al.*, 2007] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing SAX: a Novel Symbolic Representation of Time Series. *Data Mining and Knowledge Discovery*, 15:107–144, 2007.

[Needleman and Wunsch, 1970] Saul B. Needleman and Christian D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *Journal of molecular biology*, 48(3):443–453, March 1970.

[Papapetrou *et al.*, 2009] Panagiotis Papapetrou, George Kollios, Stan Sclaroff, and Dimitrios Gunopulos. Mining Frequent Arrangements of Temporal Intervals. *Knowledge and Information Systems*, 21(2):133–171, 2009.

[Patel *et al.*, 2008] Dhaval Patel, Wynne Hsu, and Mong Li Lee. Mining Relationships Amont Interval-based Events for Classification. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD'08)*, pages 393–404, 2008.

[Plamondon, 2000] Rejean Plamondon. On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):63–84, 2000.

[Winarko and Roddick, 2007] Edi Winarko and John F Roddick. ARMADA – An Algorithm for Discovering Richer Relative Temporal Association Rules from Interval-Based Data. *Data & Knowledge Engineering*, 63:76–90, 2007.