Detecting Reduplication in Videos of American Sign Language

Zoya Gavrilov, Stan Sclaroff*, Carol Neidle*, Sven Dickinson

University of Toronto, *Boston University
6 King's College Road Toronto ON, 111 Cummington Street Boston MA
zoya.gavrilov@utoronto.ca, sclaroff@cs.bu.edu, carol@bu.edu, sven@cs.toronto.edu

Abstract

A framework is proposed for the detection of reduplication in digital videos of American Sign Language (ASL). In ASL, reduplication is used for a variety of linguistic purposes, including overt marking of plurality on nouns, aspectual inflection on verbs, and nominalization of verbal forms. Reduplication involves the repetition, often partial, of the articulation of a sign. In this paper, the apriori algorithm for mining frequent patterns in data streams is adapted for finding reduplication in videos of ASL. The proposed algorithm can account for varying weights on items in the apriori algorithm's input sequence. In addition, the apriori algorithm is extended to allow for inexact matching of similar hand motion subsequences and to provide robustness to noise. The formulation is evaluated on 105 lexical signs produced by two native signers. To demonstrate the formulation, overall hand motion direction and magnitude are considered; however, the formulation should be amenable to combining these features with others, such as hand shape, orientation, and place of articulation.

Keywords: sign recognition; reduplication; apriori algorithm

1. Introduction

In this paper, we propose an automatic framework for the detection of reduplication in digital videos of American Sign Language (ASL). The focus of our research is lexical signs, the largest morphological subclass in ASL; thus, we do not consider other types of signs, including fingerspelled signs and classifier constructions. The production of lexical signs in ASL involves particular hand configurations, orientations, places of articulation in the signing space, and types of movement (Stokoe et al., 1965; Klima and Bellugi, 1988). Reduplication–a full or partial repetition of the base form of a sign–can be involved in various constructions, including overt marking of plurality on nouns, aspectual inflection on verbs (frequency, intensity, repetition, duration, etc.), and nominalization of verbal forms (Fischer, 1973; Wilbur, 1973; Pfau and Steinbach, 2006; Cokely and Baker-Shenk, 1980; Perry, 2005; MacLaughlin et al., 2000).

Given the prevalence and linguistic importance of reduplication, it is clear that an automated ASL recognition system must include a model for reduplication. Moreover, modeling reduplication is fundamental in solving the problem of sign segmentation; for instance, in parsing a sequence of continuous signs, it is essential to know whether two sequential movements are contained within a single sign or two different signs.

The richness of reduplication presents interesting challenges for automated analysis. When reduplicated, the movement of the base form of a sign is sometimes reduced in magnitude and damped with subsequent repetitions. In some linguistic constructions, the reduplicated motion may be superimposed on a larger motion pattern (MacLaughlin et al., 2000). For example, in overt marking of plurality on nouns the reduplicated motion is frequently subject to a lateral translation, as shown in Fig. 1. All of these modifications to the base form present substantial challenges for the automated detection of reduplication. Despite these challenges, there are still significant underlying regularities

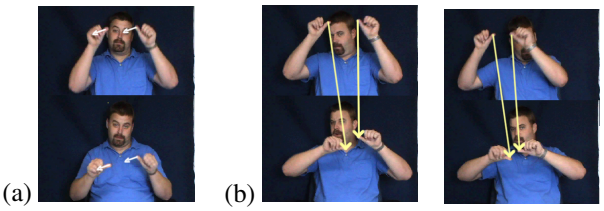

Figure 1: Example from (Neidle, 2007a) of singular and plural signs for POSTER, both bisyllabic.
(a) The singular version involves two forward movements;
(b) The plural version involves two downward movements, laterally displaced from one another.

that characterize reduplication, which form a basis for our approach: we recognize within a signing sequence inexact repetitions of the type typically found in reduplication.

To detect reduplication, we formulate an approach that finds inexact repetitions of a base form within a signing sequence. We adapt the apriori algorithm (Agrawal et al., 1993) for efficiently finding repeated patterns in data streams. An ASL video clip is first processed to obtain the movement parameters for the signer's hands in each video frame. These hand movement parameters are then mapped to a codebook to yield a sequence of symbols, such that similar hand movement parameters are mapped to the same code in the codebook. The apriori algorithm is then used to find temporally adjacent, repeated patterns within the sequence. Our formulation of the apriori algorithm allows for inexact matching of similar hand motion subsequences and thereby provides robustness to possible noise in hand movement parameters. Our approach is evaluated on 105 lexical signs produced by two native signers. For the purpose of demonstrating our formulation, overall hand motion direction and magnitude are considered; however, our formulation should be amenable to combining these features with others, such as hand shape, orientation, and place of articulation.

The plan for the rest of the paper is as follows. In Section 2, we review related work. To our knowledge, we are the first to propose a computational approach for the detection of reduplication in ASL. In Section 3, we describe our algorithm for finding inexact repeated subsequences of states, given an input of state and weight vectors. In Section 4, we give implementation details for processing ASL videos to obtain the movement parameters for the signer's hands. In Section 5, we describe our results in detecting reduplication given sequences of hand movement parameters. In Section 6, we outline the current limitations of our algorithm and directions for future research, and in Section 7 we conclude by outlining the contributions of this work.

2. Related work

2.1. String-based representations of human motion

A number of past works have proposed discretizing motion sequences into states on which string pattern matching algorithms can operate. For instance, (Beaudoin et al., 2008) convert motions to motion state strings by k-means clustering the pose space, and associating each pose with the nearest cluster. Sequential repetitions of letters are removed, and an adjacency matrix helps identify non-identical substrings that represent similar motions. Repeated motions are found by performing string matching with randomly generated seed substrings (of prespecified lengths). In contrast, we find repeated patterns by sequentially generating larger patterns from smaller patterns constrained by the apriori principle.

In (Fihl et al., 2006) a motion sequence is represented as a string of primitives from a predefined set. The string is pruned by removing isolated instances and collapsing all repeated letters, with a weight generated to reflect the number of repetitions. Probabilistic edit distances are used to compare a motion sequence to labelled motion sequences to identify the most likely candidate for recognition of the query sequence. This approach assumes that the patterns to be located in sequences are known in advance. Our framework does not make this assumption.

In (Bobick and Wilson, 1997) gestures are recognized by matching them to prototypes, where a prototype is represented as a curve through configuration space, parameterized according to arc length, and independent of time. The prototype is then partitioned into regions, or states, based on how its direction varies. For a gesture to be recognized, it must match sufficiently and pass through all the required states. Such an approach is unlikely to work well for natural human motions that exhibit wide variability.

Other approaches seek to match one motion to another by computing the similarity between video segments, e.g, by direct frame-to-frame similarity computation (Schödl et al., 2000; Efros et al., 2003), or dynamic time warping for sequence alignment (Tang et al., 2008). In our problem, we are looking for similar motion segments within a *single* video. Hence, utilizing an approach that compares video sequences would theoretically entail computing the pairwise similarity between all non-overlapping frame sequences within a single video –an approach that would be exponential in the length of the input sequence.

In bioinformatics, a related problem entails *motif finding*– the discovery of commonly recurring subunits among a set of sequences (for a survey see (Das and Dai, 2007)); however, such approaches generally assume a very high degree of regularity to the repeating units.

2.2. Apriori algorithm

The apriori algorithm was originally formulated for mining association rules in transactional databases (Agrawal et al., 1993), and then extended to handle temporal sequences (see (Laxman and Sastry, 2006) for a review). The apriori algorithm proceeds incrementally in levels, where each level comprises a candidate generation step followed by a frequency counting step. The *apriori property* that is used in generating candidate patterns of progressively larger sizes at each level guarantees that the frequency of a pattern is bounded above by the frequencies of its subpatterns; thus larger patterns are grown from smaller patterns, thereby providing computational savings. The complexity of candidate generation is polynomial in the size of the collection of frequent patterns, but is independent of the length of the input sequence (Mannila et al., 1997).

The frequency counting step tallies the number of times each candidate pattern occurs in the input. In (Mannila et al., 1997), where a finite state automaton is used to keep track of serial candidate patterns, this has time complexity $O(n|C|l)$, where n is the length of the sequence, C contains the candidate patterns for this level, and l is the maximum number of automata for each pattern. Laxman et al. (Laxman et al., 2007) propose a variation to this algorithm, where the frequency is a measure of *non-overlapped* occurrences. This reduces the complexity of frequency counting to $O(n|C|)$. We adapt this latter approach to our problem.

3. Approach

Our goal is to determine efficiently whether a given sign includes reduplication. The input to our system is a video sequence with n frames.

A tracker estimates the signer's hand locations in each video frame $f_i, i \in [1, n]$. Features are computed for the detected hands in each frame. The features for each hand are then mapped onto a codebook B of size m, resulting in a sequence of discrete states, $S = \{s_1, \ldots s_n\}$ representing the video sequence, where $s_i \in B, \forall i \in [1, n]$.

A variant of the apriori algorithm is then used to locate repeated patterns within the sequence of discrete states S. The apriori algorithm (Agrawal et al., 1993) was designed to enable efficient searching for repeated sequential patterns over large databases. Our application differs, in that we want to search efficiently for reduplication within individual ASL signs. Nevertheless, the basic apriori algorithm offers a systematic and efficient framework for detecting repeated patterns within signs, which we adapt to account for inexact matching of a reduplicated base form within a sign.

3.1. Apriori algorithm overview

Algorithm 1 summarizes our adaptation of the apriori algorithm. The input to the algorithm includes a sequence of n states S, an n-dimensional weighting vector $\mathbf{w}$ that gives

Algorithm 1 findRedup($S, \mathbf{w}, \mathbf{M}, thresholds$)

1: $C \leftarrow$ getSetOfStates(S)
2: **while** notEmpty(C) **do**
3: $[freq, cumValue, cumCost] \leftarrow$ countFrequencies($S, \mathbf{w}, \mathbf{M}, thresholds$)
4: $C \leftarrow$ generateCandidates($C, freq, cumValue, cumCost, thresholds$)
5: **end while**
6: return C

the certainty for each state in the sequence S, a similarity matrix $\mathbf{M}$ that determines the cost of matching any two states in the codebook B, and thresholds that control inexact matching and pruning of the search for repeated patterns. Each of the inputs to the algorithm will be explained in more detail below.

Alg. 1 first invokes the function getSetOfStates to produce an initial list of candidate patterns C. This initial list C contains all possible single-state patterns that can appear within an input string S (hence, at this point, $C \subseteq B$). In the original apriori algorithm frequency-based pruning occurs at this stage; however, our algorithm will provide inexact matches, so no pruning occurs initially. The algorithm then iterates, alternating between frequency counting (Alg. 2) and candidate generation (Alg. 3), with increasing pattern size at each iteration.

3.2. Frequency counting

Algorithm 2 traverses the input string S to compute the frequency of each candidate repeated pattern $C_j \in C$, where $j \in [1, |C|]$. An automaton, A_j is first initialized for each candidate pattern C_j. The automaton has internal states $c_1, ..., c_m$ corresponding to each state of C_j ($|C_j| = m$). The *match value*, v_j, for each automaton is initialized to 0. Alg. 2 then iterates over each s_i of the input string S, checking to see if any automata match to the state at position s_i. If there is a match, then the corresponding automaton A_j, its match cost $cost_j$, and match value v_j are updated.

To account for natural variation in signing, the matching of the states in Alg. 2 is inexact. The pairwise matching cost of states is defined by a similarity matrix $\mathbf{M}$, where $0 \leq m_{u,v} \leq 1$. Diagonal entries $m_{u,u} = 1$ (a state is most similar to itself) and non-zero, off-diagonal entries in $\mathbf{M}$ specify the degree of the match between the state an automaton was waiting for (c_k) and the state on the string at s_i. A poor match increases the match cost $cost_j$ for the given automaton. In our implementation, because of the coarse discretization of directions of hand motions, we allow an automaton waiting for a given bin also to accept its first-neighbor bins as valid, with an associated cost (see Alg. 2). The other entries in $\mathbf{M}$ are zero. A more general $\mathbf{M}$ would require further modifications to the algorithm.

To account for the varying certainty of (or confidence in) state labels in the sequence S, each state s_i has a corresponding scalar weight w_i. If an automaton matches a state, then the *match value*, v_j, for that automaton A_j is increased by the corresponding weight w_i; otherwise, a noise state decreases v_j by w_i. In our implementation, the weight w_i is proportional to the magnitude of the motion, although other measures of certainty are possible.

In the case of a successful match, an automaton advances from c_k to c_{k+1} ($1 \leq k < m$), to wait for the next state. Otherwise, the automaton continues to wait for the same state. Additionally, if an automaton processes the same state multiple times consecutively, it accumulates a match value and a cost, without advancing to the next state. As a result, a state that recurs over multiple frames but has low certainty each time may have an accumulated match value ($v_j = \sum_k w_k$) that is comparable to the case where a state occurs in only one frame but with a high certainty value ($v_j = w_k$). This allows us to robustly handle motion patterns that occur with varying speeds.

In our implementation, when an automaton processes a noise state (an input state with no similarity value to the state an automaton is waiting for), the noise state contributes a negative match value. Hence, the accumulated match value of an automaton can increase or decrease as matching to the input progresses. As part of each iteration of Alg. 2, an automaton A_j is restarted (reinitialized) if this accumulated match value v_j falls below a minimum threshold (lines 17-20 in Alg. 2).

If the final state c_m is reached for an automaton A_j, then the frequency count for the candidate pattern C_j is incremented by one, and the cumulative value and cost for the pattern are incremented by v_j and $cost_j$, respectively. The automaton is restarted so that search for the next possible match can continue in the remaining part of the string S. These steps are shown in Alg. 2, lines 21-27.

3.3. Candidate generation

Algorithm 3 generates an updated list of the candidate patterns to be used in matching. Updated candidate patterns are generated from the patterns that were matched with sufficient strength and a sufficient number of times by the automata in Alg. 2. To generate new larger patterns for the next iteration, overlapping patterns from the previous iteration are concatenated. This exploits the apriori property to guarantee that the frequency (and certainty) of a pattern is bounded above by the frequencies (and certainty values) of its subpatterns, thereby reducing the computational cost of brute search through all possible patterns.

The patterns that survive to the next iteration (as components of larger patterns) are those that are not pruned in lines 1-6 of Alg. 3. In pruning, we consider the following cumulative value measurements for a given pattern C_j:

$$costMeasure(C_j) = \frac{cumCost(C_j)}{|C_j| \times freq(C_j)} \quad (1)$$

$$valMeasure(C_j) = \frac{cumValue(C_j)}{freq(C_j)} \quad (2)$$

Here, $freq(C_j)$ is the number of times C_j has matched to the input string (via exact/inexact matches) and

Algorithm 2 countFrequencies$(S, C, \mathbf{w}, \mathbf{M}, thresholds)$

for all $j = 1$ **to** $|C|$ **do**
  $A_j \leftarrow$ makeAutomaton(C_j)
  $v_j \leftarrow 0$
end for
for all $i = 1$ **to** $|S|$ **do**
  for all $j = 1$ **to** $|A|$ **do**
    if waiting(A_j, s_i) **then**
      $cost_j \leftarrow cost_j + (1 - \mathbf{M}(A_j, s_i))$
      $v_j \leftarrow v_j + w_i$
      update(A_j)
    else if curState(A_j, s_i) **then**
      $cost_j \leftarrow cost_j + (1 - \mathbf{M}(A_j, s_i))$
      $v_j \leftarrow v_j + w_i$
    else
      $v_j \leftarrow v_j - w_i$
    end if
    if $v_j < weightThresh$ **then**
      reinitialize(A_j)
      $v_j \leftarrow 0$
    end if
    if finalState(A_j) **then**
      $freq(C_j) \leftarrow freq(C_j) + 1$
      $cumValue(C_j) \leftarrow cumValue(C_j) + v_j$
      $cumCost(C_j) \leftarrow cumCost(C_j) + cost_j$
      reinitialize(A_j)
      $v_j \leftarrow 0$
    end if
  end for
end for
return $freq, cumValue, cumCost$

$cumValue(C_j)$ is the sum of the weights of all states in the input string matched to C_j. Hence, $valMeasure$ is a measure of the certainty of a match to the pattern, averaged over all occurrences of the pattern in the input. The cumulative penalty $cumCost$ is the sum of the costs of matching C_j to the input string. In our implementation, costs are 0 for perfect matches, and 1 for neighboring bins. A pattern will have a high $costMeasure$ if the total cost of the matches is high, relative to the size and frequency of the pattern in the input. A pattern will be pruned if $valMeasure$ falls below a threshold, or if $costMeasure$ exceeds a threshold.

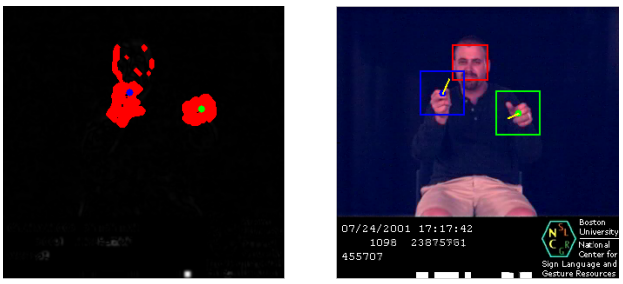

Figure 2: (a) Clustering of moving points; (b) Detected hands and face. Also displayed (in yellow) are the motion directions of the hands between successive frames.

4. Implementation details

4.1. Tracking

The input is a video clip corresponding to a sign, sampled at 30 fps, where each frame is 312×324 pixels. For hand tracking, we use a simple k-means clustering (where $k = 2$, for the hands) applied to difference images. A difference image provides a crude measure of motion but offers the benefits of being simple to compute and providing robustness to illumination and appearance changes (Fihl et al., 2006). A face tracker (Bradski, 1998) is used to subtract head motion. Fig. 2 shows an example processed frame from an ASL video sequence.

The initial hand and face locations are specified with bounding boxes in the first frame of each sequence. Pixels within the bounding boxes are used to initialize the skin color model and the thresholds used in frame differencing. The tracker then runs autonomously.

We have also built a tool that allows for user-directed track adjustment. At each frame, the tracker offers a hypothesis for the bounding boxes of the hands, which the user can then adjust. Tracker computations for subsequent frames then take into account the adjusted locations. This tool was used to help label video sequences that the tracker had trouble with, thereby increasing the amount of data that could be used to test the reduplication-detection module separately from the detection module. This provides us with the capability of testing our apriori algorithm on the results of a gold standard tracking of the hands, in order to exam-

Algorithm 3 generateCandidates($C, freq, cumValue, cumCost, thresholds$)

for all $i = 1$ **to** $|C|$ **do**
  $costMeasure(C_i) \leftarrow cumCost(C_i)/(|C_i| \times freq(C_i))$
  if $cumValue(C_i)/freq(C_i) \leq valThresh$ **or** $costMeasure(C_i) \geq costThresh$ **then**
    remove(C, C_i)
  end if
end for
$newC = \{\}$
for all $1 \leq j, k \leq |C|, j \neq k$ **do**
  if $C_j[2 : end] == C_k[1 : end - 1]$ **then**
    $newPat \leftarrow$ concatenate($C_j, C_k[end]$)
    insert($newC, newPat$)
  end if
end for
return $newC$

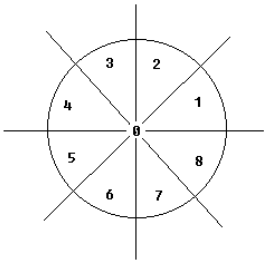

Figure 3: Hand motion projected onto the 2D image plane is discretized to one of 8 directions, or bin 0 when there is no motion (motion magnitude below threshold).

ine limitations that remain in detecting reduplication in the absence of tracker noise. Please see Sec. 6. for a discussion of some of these limitations.

4.2. Codebook Representation

For each frame, the direction of each hand's motion is computed. These directions are discretized into 8 bins, each spanning 45 degrees, with the exception that if no motion is observed, the motion is assigned to bin 0 (see Fig. 3). We process the sequence of each hand separately to produce its corresponding sequence of symbols S and weights w_i. In our implementation, the w_i are proportional to the magnitude of the hand's displacement from frame $i-1$ to i.

More precisely, for a video sequence $F = \{f_1, ..., f_n\}$, we compute as hand features H_i ($i \in [1, n]$), the displacement direction from the hand cluster center in f_{i-1} to the hand cluster center in f_i, mapping these onto the codebook of discrete directions $B = [0, 8]$ and weights w_i that capture the magnitudes of each displacement. The video is represented as a sequence of discrete states, S (where $s_i \in B$), with associated weight vector $\mathbf{w}$. Note that for the first frame in a sequence, the direction and magnitude of motion is defined to be 0.

This particular codebook design was chosen to demonstrate our method for detecting reduplication in ASL. We found that this codebook design works well in our experiments. However, the optimal number of states and codebook could be learned from training data. Learning the codebook for representing ASL hand motion parameters remains a topic for future investigation.

5. Experiments

The experiments were run on sequences from the National Center for Sign Language and Gesture Resources (NCSLGR) Corpus, a collection of ASL videos collected at Boston University from Deaf native signers and linguistically annotated using SignStream®(Neidle, 2002a). The corpus, available from http://www.bu.edu/asllrp/ (see also (Neidle and Vogler, 2012)), includes synchronized video files showing the signing from the front and side, as well as a close-up of the face (although only the front view was used for the current research), as well as linguistic annotations of both manual signs (represented by unique gloss labels) and nonmanual behaviors; annotation conventions are documented in (Neidle, 2002b; Neidle, 2007b). The dataset includes 19 short narratives (containing a total of 1002 utterances) plus 885 additional elicited utterances. This constitutes a total of 1,888 linguistically annotated utterances, with 1,920 distinct canonical signs (grouping together close variants) and 11,861 total sign tokens. The sign videos for this research are lexical signs taken from the corpus, and hence occur in context, with natural variability. Focusing on data from two of the signers, we obtained a subset of lexical signs that had been labeled as including reduplication (120 sign examples involving 58 distinct signs) and removed those in which we failed to see reduplication, probably because of the placement of the frontal camera, movements small in magnitude, or errors in labelling. What remained were 84 video sequences, each containing a reduplicated sign (a total of 58 distinct signs for which we had one or more examples with reduplication). We then queried the corpus for sequences involving those same two signers that contained non-reduplicated versions of the same signs, and found 21 sign examples (involving 5 distinct signs). Together, the 84 examples of signs with reduplication, and the 21 non-reduplicated examples of signs with reduplicated counterparts in the dataset, formed the 105 sequences that were used for testing our algorithm.

Of these 105 video sequences, 12 were grayscale sequences of one signer, and the rest were RGB sequences of another signer (based on the availability of data for each in the corpus). Moreover, the video sequences used came from 11

separately-recorded videos, thereby introducing some variability in the appearance of the signers. The fact that the video segments have been extracted from continuous signing means that the signs are occurring in different linguistic contexts, and are subject to a variety of contextually-sensitive linguistic processes, including coarticulation effects. The output of the hand tracker was mapped to the binned motion directions for each hand in each frame, as described in Sec. 4.. The apriori algorithm was then run on the motion sequence of each hand, separately, such that reduplication was claimed if at least one of the hands exhibited reduplication. This allowed the system to account for one-handed signs, as well as to provide greater robustness to tracker errors.

To separate the modules (and hence failure modes) of motion tracking and reduplication detection, another set of experiments was run with manually corrected tracks. This involved integrating the detector into an interactive system, whereby a user could correct the tracker's errors. The average percentage of overlap between the hand-corrected and automatically-tracked bounding boxes is 60%, taken over all frames of all sequences in our dataset. Fewer than 10% of the automatically-tracked bounding boxes had overlap of 10% or less with the hand-corrected bounding boxes.

Our apriori algorithm was run with the following parameter settings: $weightThresh = 0$, $costThresh = 0.5$ and $valThresh$ varying from 0 to 10. The threshold $weightThresh$ determines the amount of tolerance an automaton has for matching noise states in the patterns it detects; the setting $weightThresh = 0$ means that search is pruned whenever the weighted penalty for matching noise states exceeds the score for matching states in the automaton's pattern. The thresholds $valThresh$ and $costThresh$ control the pruning of patterns at the end of the computation; these two thresholds determine whether patterns have occurred with a high enough certainty and with a low enough match cost, respectively.

The ROC curve in Fig. 4 shows the performance of the reduplication detection system. The ROC measure plots the trade-off between the detection rate and the false positive rate. The false positive rate pictured is controlled by varying $valThresh$ in the range 0 to 10. The red curve shows detection accuracy given the output of our simple automatic hand tracker, whereas the blue curve shows results given hand-corrected tracks. Comparison of these curves shows that the reduplication detection formulation works relatively well in the presence of tracking errors for this dataset. The green curve in Fig. 4 shows performance on the corrected video sequences, but with no bin variations permitted within the apriori pattern matching, i.e., M is given as an identity matrix. As demonstrated by the graph, inexact matching is beneficial, as it compensates for the errors introduced by the discretization of a continuous space.

6. Discussion

The proposed framework discovers repeating patterns of states amidst noise and natural variation. We have thus demonstrated the capabilities of this framework for dealing with signs represented only by motion directions. For a more comprehensive analysis of reduplication, the algorithm presented here can be extended to incorporate additional features, including hand shape, orientation, and place of articulation.

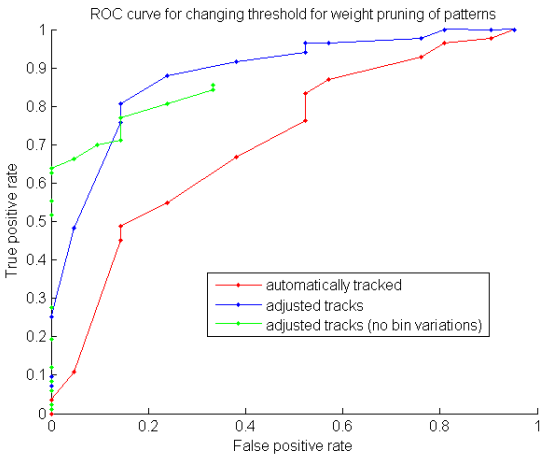

Figure 4: ROC curve as a function of changing the threshold for pattern scores.

Our feature representation of motion bins is only sufficient to represent signs where the motion parameter undergoes reduplication (in the frontal plane). This is not the case, for instance, for signs where reduplication is limited to change in handshape or local finger movement. In the latter case, reduplication may be difficult to detect using the methods described here. To handle all cases robustly, a different representation of hand motion may be necessary.

Motion repetition may be observed in the absence of reduplication; a particular motion pattern may repeat in an ASL utterance without the repetition being attributable to reduplication. Other features may also be useful for disambiguation. In particular, a repetition of motion is not sufficient to claim reduplication, unless the entire set of articulatory features (e.g., including handshape) is involved in the repeated motion.

The performance of the apriori algorithm in detecting repeating motion is fully dependent upon the input it receives; thus, a faulty tracker is likely to lead to an overall degradation of detection performance. For instance, a common failure mode of our simple tracker involves detecting the elbow or other part of the arm as a hand. Out of the 11 videos from which signs were obtained, 6 videos involved a signer with short sleeves. Better localization of the elbows and arms of the signer as in (Buehler et al., 2008) would be an important feature of a tracker for handling these sorts of videos.

7. Conclusion

In this paper we have formulated the problem of detecting reduplication in ASL video streams and we have presented a framework for doing so. We have extended the apriori algorithm to handle the natural variability of sign language and also the noise that is likely to be introduced by motion detectors. We have offered some promising initial results of the framework applied to the movement parameter of signs, and have discussed how the framework can be extended to allow for more complete representations.

A more general approach for detecting reduplication would

require a more complete representation of hand states and a corresponding distance metric. Mapping such a representation to a finite state space would permit treating the reduplication detection problem as a string pattern matching problem. The results presented in this paper merely reinforce our conviction that more features are required for a more comprehensive analysis of reduplication, but we have nevertheless offered hopeful results indicating that the methodology we have chosen is suitable for this purpose.

In this paper, hand motion features were estimated in the image plane, i.e., in two-dimensions. Three-dimensional measurements of hand motion parameters would provide a more complete representation of hand states. Three-dimensional sensing is becoming more affordable, given the advent of computer game sensors like the Microsoft Kinect depth-sensing system. Thus, an extension of our framework to exploit 3D hand motion features is an exciting possible future direction for our research.

8. Acknowledgments

We would like to acknowledge Ashwin Thangali of Boston University for his support, advice, and technical help, throughout the duration of this project. We would also like to thank Joan Nash of Boston University and Braden Painter of Gallaudet University for helpful discussions about reduplication. The research reported here has been partially funded by the Natural Sciences and Engineering Research Council (NSERC) of Canada and by grants from the U.S. National Science Foundation: 0705749, 0964385, 0958442, and 0964385.

9. References

R. Agrawal, T. Imielinski, and A. Swami. 1993. Mining association rules between sets of items in large databases. In *ACM SIGMOD*, pages 207–216.

P. Beaudoin, S. Coros, M. Van de Panne, and P. Poulin. 2008. Motion-motif graphs. In *ACM SIGGRAPH*, pages 117–126.

A.F. Bobick and A.D. Wilson. 1997. A state-based approach to the representation and recognition of gesture. In *PAMI*, volume 19, pages 1325–1337.

B. Bradski. 1998. Computer vision face tracking for use in a perceptual user interface. *Intel Technology Journal*.

P. Buehler, M. Everingham, D.P. Huttenlocher, and A. Zisserman. 2008. Long term arm and hand tracking for continuous sign language TV broadcasts. In *BMVC*.

D. Cokely and C. Baker-Shenk. 1980. *American Sign Language: A Teacher's Resource Text on Curriculum, Methods, and Evaluation*. Gallaudet University Press, Washington, DC.

M. Das and H. Dai. 2007. A survey of DNA motif finding algorithms. *BMC Bioinformatics*, 8(7).

A.A. Efros, A.C. Berg, G. Mori, and J. Malik. 2003. Recognizing action at a distance. In *ICCV*, volume 2, pages 726–733.

P. Fihl, M. B. Holte, T. B. Moeslund, and L. Reng. 2006. Action recognition using motion primitives and probabilistic edit distance. In *AMDO*, pages 375–384.

S. Fischer. 1973. Two processes of reduplication in the American Sign Language. In *Foundations of Language*, volume 9, pages 469–480.

E. Klima and U. Bellugi. 1988. *The Signs of Language*. Harvard University Press, Cambridge, MA.

S. Laxman and P.S. Sastry. 2006. A survey of temporal data mining. In *SADHANA, Academy Proceedings in Engineering Sciences*, volume 31, pages 173–198.

S. Laxman, P.S. Sastry, and K. P. Unnikrishnan. 2007. A fast algorithm for finding frequent episodes in event streams. In *ACM SIGKDD*, pages 410–419.

D. MacLaughlin, C. Neidle, B. Bahan, and R.G. Lee. 2000. Morphological inflections and syntactic representations of person and number in ASL. In *Recherches linguistiques de Vincennes*, volume 29, pages 73–100.

H. Mannila, H. Toivonen, and A. Verkamo. 1997. Discovery of frequent episodes in event sequences. *Data Mining Knowledge Discovery*, pages 259–289.

C. Neidle and C. Vogler. 2012. A new web interface to facilitate access to corpora: Development of the ASLLRP data access interface (DAI). In *Proc. 5th Workshop on the Representation and Processing of Sign Languages: Interactions between Corpus and Lexicon, LREC 2012*.

C. Neidle. 2002a. SignStream™: A database tool for research on visual-gestural language. *Journal of Sign Language and Linguistics*, pages 203–214.

C. Neidle. 2002b. SignStream™Annotation Conventions used for the American Sign Language Research Project. Technical report, Boston University, Boston, MA. Report 11, Boston University American Sign Language Linguistic Research Project.

C. Neidle. 2007a. Interplay between manual and non-manual expressions in American Sign Language (ASL). In A. Zaenen, J. Simpson, T. Holloway-King, J. Grimshaw, J. Maling, and C. Manning, editors, *Architectures, Rules, and Preferences: A Festschrift for Joan Bresnan*. CSLI.

C. Neidle. 2007b. SignStream™Annotation: Addendum to Conventions used for the American Sign Language Research Project. Technical report, Boston University, Boston, MA. Report 13, Boston University American Sign Language Linguistic Research Project.

D. Perry. 2005. The use of reduplication in ASL plurals. Master's Project, Boston University.

R. Pfau and M. Steinbach. 2006. Pluralization in sign and in speech: A cross-modal typological study. In *Linguistic Typology*, volume 10, pages 135–182.

A. Schödl, R. Szeliski, D. H. Salesin, and I. Essa. 2000. Video textures. In *SIGGRAPH*, pages 489–498.

W. C. Stokoe, D. C. Casterline, and C. G. Croneberg. 1965. *A dictionary of American Sign Language on linguistic principles*. Linstok Press, Silver Spring, MD.

K.T. Tang, H. Leung, T. Komura, and H. Shum. 2008. Finding repetitive patterns in 3D human motion captured data. In *Proc. 2nd Int. conference on ubiquitous information management and communication*, pages 396–403.

R. Wilbur. 1973. *The Phonology of Reduplication*. Ph.D. thesis, University of Illinois, Urbana-Champaign, IL. Published by the Indiana University Linguistics Club.