Sliding Alignment Windows for Real-Time Crowd Captioning

Mohammad Kazemi, **Rahman Lavaee**, **Iftekhar Naim** and **Daniel Gildea**
Dept. of Electrical and Computer Engineering and
Dept. of Computer Science
University of Rochester
Rochester, NY 14627

Abstract

The primary way of providing real-time speech to text captioning for hard of hearing people is to employ expensive professional stenographers who can type as fast as natural speaking rates. Recent work has shown that a feasible alternative is to combine the partial captions of ordinary typists, each of whom is able to type only part of what they hear. In this paper, we extend the state of the art fixed-window alignment algorithm (Naim et al., 2013) for combining the individual captions into a final output sequence. Our method performs alignment on a sliding window of the input sequences, drastically reducing both the number of errors and the latency of the system to the end user over the previously published approaches.

1 Introduction

Real-time captioning provides deaf or hard of hearing people access to speech in mainstream classrooms, at public events, and on live television. Studies performed in the classroom setting show that the latency between when a word was said and when it is displayed must be under five seconds to maintain consistency between the captions being read and other visual cues (Wald, 2005; Kushalnagar et al., 2014). The most common approach to real-time captioning is to recruit a trained stenographer with a special purpose phonetic keyboard, who transcribes the speech to text with less than five seconds of latency. Unfortunately, professional captionists are quite expensive ($150 per hour), must be recruited in blocks of an hour or more, and are difficult to schedule on short notice. Automatic speech recognition (ASR) systems (Saraclar et al., 2002), on the other hand, attempts to provide a cheap and fully automated solution to this problem. However, the accuracy of ASR quickly plummets to below 30% when used on an untrained speaker's voice, in a new environment, or in the absence of a high quality microphone (Wald, 2006). The accuracy of the ASR systems can be improved using the 're-speaking' technique, which requires a person that the ASR has been trained on to repeat the words said by a speaker as he hears them. Simultaneously hearing and speaking, however, is not straightforward, and requires some training.

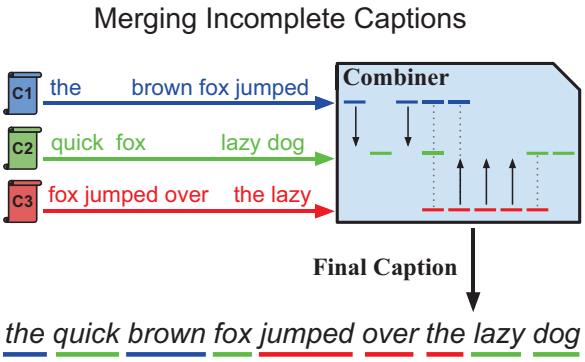


Figure 1: General layout of crowd captioning systems. Captionists (C1, C2, C3) submit partial captions that are automatically combined into a high-quality output.

An alternative approach is to combine the efforts of multiple non-expert captionists (anyone who can type), instead of relying on trained workers (Lasecki et al., 2012; Naim et al., 2013). In this approach, multiple non-expert human workers transcribe an audio stream containing speech in real-time. Workers type as much as they can of

Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Short Papers), pages 236–240,
Baltimore, Maryland, USA, June 23-25 2014.

the input, and, while no one worker's transcript is complete, the portions captured by various workers tend to overlap. For each input word, a timestamp is recorded, indicating when the word is typed by a worker. The partial inputs are combined to produce a final transcript (see Figure 1). This approach has been shown to dramatically outperform ASR in terms of both accuracy and Word Error Rate (WER) (Lasecki et al., 2012; Naim et al., 2013). Furthermore, recall of individual words irrespective of their order approached and even exceeded that of a trained expert stenographer with seven workers contributing, suggesting that the information is present to meet the performance of a stenographer (Lasecki et al., 2012). However, aligning these individual words in the correct sequential order remains a challenging problem.

Lasecki et al. (2012) addressed this alignment problem using off-the-shelf multiple sequence alignment tools, as well as an algorithm based on incrementally building a precedence graph over output words. Improved results for the alignment problem were shown using weighted A* search by Naim et al. (2013). To speed the search for the best alignment, Naim et al. (2013) divided sequences into chunks of a fixed time duration, and applied the A* alignment algorithm to each chunk independently. Although this method speeds the search for the best alignment, it introduces a significant number of errors to the output of the system due to inconsistency at the boundaries of the chunks. In this paper, we introduce a novel sliding window technique which avoids the errors produced by previous systems at the boundaries of the chunks used for alignment. This technique produces dramatically fewer errors for the same amount of computation time.

2 Problem Overview and Background

The problem of aligning and combining multiple transcripts can be mapped to the well-studied Multiple Sequence Alignment (MSA) problem (Edgar and Batzoglou, 2006). Let $S_1, \ldots, S_K, K \geq 2$, be the K sequences over an alphabet Σ, and having length $N_1, \ldots, N_K$. For the caption alignment task, we treat each individual word as a symbol in our alphabet Σ. The special gap symbol '$-$' represents a missing word and does not belong to Σ. Let $A = (a_{ij})$ be a $K \times N_f$ matrix, where $a_{ij} \in \Sigma \cup \{-\}$, and the i^{th} row has exactly $(N_f - N_i)$ gaps and is identical to S_i if we ignore

Algorithm 1 MSA-A* Algorithm

Require: K input sequences $\mathcal{S} = \{S_1, \ldots, S_K\}$ having length $N_1, \ldots, N_K$, heuristic weight w, beam size b
input $start \in \mathbb{N}^K$, $goal \in \mathbb{N}^k$
output an $N \times K$ matrix of integers indicating the index into each input sequence of each position in the output sequence

$g(start) \leftarrow 0$, $f(start) \leftarrow w \times h(start)$.
$Q \leftarrow \{start\}$
while $Q \neq \emptyset$ **do**
  $n \leftarrow$ EXTRACT-MIN(Q)
  for all $s \in \{0,1\}^K - \{0^K\}$ **do**
    $n_i \leftarrow n + s$
    if $n_i = goal$ **then**
      Return the alignment matrix for the reconstructed path from $start$ to n_i
    else if $n_i \notin Beam(b)$ **then**
      continue;
    else
      $g(n_i) \leftarrow g(n) + c(n, n_i)$
      $f(n_i) \leftarrow g(n_i) + w \times h(n_i)$
      INSERT-ITEM($Q, n_i, f(n_i)$)
    end if
  end for
end while

the gaps. Every column of A must have at least one non-gap symbol. Therefore, the j^{th} column of A indicates an alignment state for the j^{th} position, where the state can have one of the $2^K - 1$ possible combinations. Our goal is to find the optimum alignment matrix A_{OPT} that minimizes the sum of pairs (SOP) cost function:

$$c(A) = \sum_{1 \leq i \leq j \leq K} c(A_{ij}) \quad (1)$$

where $c(A_{ij})$ is the cost of the pairwise alignment between S_i and S_j according to A. Formally, $c(A_{ij}) = \sum_{l=1}^{N_f} \mathrm{sub}(a_{il}, a_{jl})$, where $\mathrm{sub}(a_{il}, a_{jl})$ denotes the cost of substituting a_{jl} for a_{il}. If a_{il} and a_{jl} are identical, the substitution cost is zero. The substitution cost for two words is estimated based on the edit distance between two words. The exact solution to the SOP optimization problem is NP-Complete (Wang and Jiang, 1994), but many methods solve it approximately. Our approach is based on weighted A* search for approximately solving the MSA problem (Lermen and Reinert, 2000; Naim et al., 2013).

2.1 Weighted A* Search for MSA

The problem of minimizing the SOP cost function for K sequences is equivalent to estimating the shortest path between a single source node and a single sink node in a K-dimensional mesh graph, where each node corresponds to a distinct position in the K sequences. The source node is $[0, \ldots, 0]$

Algorithm 2 Fixed Window Algorithm

Require: K input sequences $\mathcal{S} = \{S_1, \ldots, S_K\}$ having length $N_1, \ldots, N_K$, window parameter $chunk\_length$.

```
start_time ← 0
while goal ≺ [N_1, ..., N_K] do
    for all i do
        start[i] ← closest_word(i, start_time)
    end for
    end_time ← start_time + chunk_length
    for all i do
        goal[i] ← closest_word(i, end_time) − 1
    end for
    alignmatrix ←MSA-A*(start, goal)
    concatenate alignmatrix onto end of finalmatrix
    start_time ← end_time
end while
Return finalmatrix
```

and the sink node is $[N_1, \ldots, N_K]$. The total number of nodes in the lattice is $(N_1+1) \times (N_2+1) \times \cdots \times (N_K+1)$, and each node has $2^K - 1$ possible successors and predecessors. The A* search algorithm treats each node position $n = [n_1, \ldots, n_K]$ as a search state, and estimates the cost function $g(n)$ and the heuristic function $h(n)$ for each state. The cost function $g(n)$ represents the exact minimum SOP cost to align the K sequences from the beginning to the current position. The heuristic function represents the approximate minimum cost of aligning the suffixes of the K sequences, starting after the current position n. The commonly used heuristic function is $h_{pair}(n)$:

$$h_{pair}(n) = L(n \rightarrow t) = \sum_{1 \leq i < j \leq K} c(A_p^*(\sigma_i^n, \sigma_j^n)) \quad (2)$$

where $L(n \rightarrow t)$ denotes the lower bound on the cost of the shortest path from n to destination t, A_p^* is the optimal pairwise alignment, and σ_i^n is the suffix of node n in the i-th sequence. The weighted A* search uses a priority queue Q to store the search states n. At each step of the A* search algorithm, the node with the smallest evaluation function, $f(n) = g(n) + wh_{pair}(n)$ (where $w \geq 1$), is extracted from the priority queue Q and expanded by one edge. The search continues until the goal node is extracted from Q. To further speed up the search, a beam constraint is applied on the search space using the timestamps of each individual input words. If the beam size is set to b seconds, then any state that aligns two words having more than b seconds time lag is ignored. The detailed procedure is shown in Algorithm 1. After the alignment, the captions are combined via majority voting at each position of the alignment matrix. We ignore the alignment columns where the majority vote is below a certain threshold t_v (typically $t_v = 2$), and thus filter out spurious errors and spelling mistakes.

Although weighted A* significantly speeds the search for the best alignment, it is still too slow for very long sequences. For this reason, Naim et al. (2013) divided the sequences into chunks of a fixed time duration, and applied the A* alignment algorithm to each chunk independently. The chunks were concatenated to produce the final output sequence, as shown in Algorithm 2.

2.2 Limitations of Fixed Window Algorithm

The fixed window based alignment has two key limitations. First, aligning disjoint chunks independently tends to introduce a large number of errors at the boundary of each chunk. This is because the chunk boundaries are defined with respect to the timestamps associated with each word in the captions, but the timestamps can vary greatly between words that should in fact be aligned. After all, if the timestamps corresponded precisely to the original time at which each word was spoken, the entire alignment problem would be trivial. The fact that the various instances of a single word in each transcription may fall on either side of a chunk boundary leads to errors where a word is either duplicated in the final output for more than one chunk, or omitted entirely. This problem also causes errors in ordering among the words remaining within one chunk, because there is less information available to constrain the ordering relations between transcriptions. Second, the fixed window alignment algorithm requires longer chunks ($\geq$ 10 seconds) to obtain reasonable accuracy, and thus introduces unsatisfactory latency.

3 Sliding Alignment Windows

In order to address the problems described above, we explore a technique based on a sliding alignment window, shown in Algorithm 3. We start with alignment with a fixed chunk size. After aligning the first chunk, we use the information derived from the alignment to determine where the next chunk should begin within each transcription. We use a single point in the aligned output as the starting point for the next chunk, and determine the corresponding starting position within each original transcription. This single point is determined by a tunable parameter $keep\_length$

Algorithm 3 Sliding Window Algorithm

Require: K input sequences $\mathcal{S} = \{S_1, \ldots, S_K\}$ having length $N_1, \ldots, N_K$, window parameters $chunk\_length$ and $keep\_length$.

1: $start \leftarrow 0^K$, $goal \leftarrow 0^K$
2: **while** $goal \prec [N_1, \ldots, N_K]$ **do**
3: $endtime \leftarrow chunk\_length + \max_i time(start[i])$
4: **for all** i **do**
5: $goal[i] \leftarrow closest\_word(i, endtime)$
6: **end for**
7: $alignmatrix \leftarrow$MSA-A$^*(start, goal)$
8: concatenate first $keep\_length$ columns of $alignmatrix$ onto end of $finalmatrix$
9: **for all** i **do**
10: $start[i] \leftarrow alignmatrix[keep\_length][i]$
11: **end for**
12: **end while**
13: Return $finalmatrix$

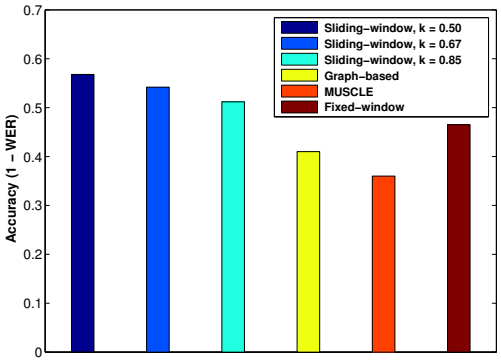
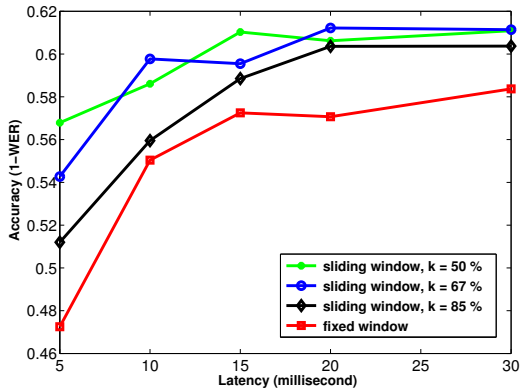
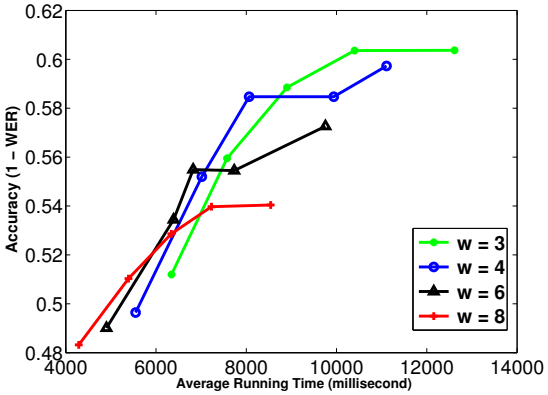


Figure 2: Evaluation of different systems on using WER metric for measuring transcription quality.



(a) varying keep-lengths for fixed heuristic weight



(b) varying heuristic weights for fixed keep-length

Figure 3: Tradeoff between speed and accuracy for different heuristic wights and keep-lengths

(line 10 of Algorithm 3). The materials in the output alignment that follow this point is thrown away, and replaced with the output produced by aligning the next chunk starting from this point (line 8). The process continues iteratively, allowing us to avoid using the erroneous output alignments in the neighborhood of the arbitrary endpoints for each chunk.

4 Experimental Results

We evaluate our system on a dataset of four 5-minute long audio clips of lectures in electrical engineering and chemistry lectures taken from MIT OpenCourseWare. The same dataset used by (Lasecki et al., 2012) and (Naim et al., 2013). Each audio clip is transcribed by 10 non-expert human workers in real time. We measure the accuracy in terms of Word Error Rate (WER) with respect to a reference transcription.

We are interested in investigating how the three key parameters of the algorithm, i.e., the chunk size (c), the heuristic weight (w) and the keep-length (k), affect the system latency, the search speed, and the alignment accuracy. The chunk size directly determines the latency of the system to the end user, as alignment cannot begin until an entire chunk is captured. Furthermore, the chunk size, the heuristic weight, and the keep-length help us to trade-off speed versus accuracy. We also compare the performance of our algorithm with that of the most accurate fixed alignment window algorithm (Naim et al., 2013). The performance in terms of WER for sliding and fixed alignment windows is presented in Figure 2. Out of the systems in Figure 2, the first three systems consist of sliding alignment window algorithm with different values of keep-length parameter: (1) keep-length = 0.5; (2) keep-length = 0.67; and (3) keep-length = 0.85. The other systems are the graph-based algorithm of (Lasecki et al., 2012), the MUSCLE algorithm of (Edgar, 2004), and the most accu-

rate fixed alignment window algorithm of (Naim et al., 2013). We set the heuristic weight parameter (w) to 3 and the chunk size parameter (c) to 5 seconds for all the three sliding window systems and the fixed window system. Sliding alignment window produces better results and outperforms the other algorithms even for large values of the keep-length parameter. The sliding alignment window with keep-length 0.5 achieves 0.5679 average accuracy in terms of (1-WER), providing a 18.09% improvement with respect to the most accurate fixed alignment window (average accuracy 0.4857). On the same dataset, Lasecki et al. (2012) reported 36.6% accuracy using the Dragon Naturally Speaking ASR system (version 11.5 for Windows).

To show the trade-off between latency and accuracy, we fix the heuristic weight ($w = 3$) and plot the accuracy as a function of chunk size in Figure 3. We repeat this experiment for different values of keep-length. We observe that the sliding window approach dominates the fixed window approach across a wide range of chunk sizes. Furthermore, we can see that for smaller values of the chunk size parameter, increasing the keep-length makes the system less accurate. As the chunk size parameter increases, the performance of sliding window systems with different values of keep-length parameter converges. Therefore, at larger chunk sizes, for which there are smaller number of boundaries, the keep-length parameter has lower impact.

Next, we show the trade-off between computation speed and accuracy in Figure 3, as we fix the heuristic weight and vary the chunk size over the range [5, 10, 15, 20, 30] seconds. Larger chunks are more accurately aligned but require computation time that grows as N^K in the chunk size N in the worst case. Furthermore, smaller weights allow faster alignment, but provide lower accuracy.

5 Conclusion

In this paper, we present a novel sliding window based text alignment algorithm for real-time crowd captioning. By effectively addressing the problem of alignment errors at chunk boundaries, our sliding window approach outperforms the existing fixed window based system (Naim et al., 2013) in terms of word error rate, particularly when the chunk size is small, and thus achieves higher accuracy at lower latency.

Acknowledgments Funded by NSF awards IIS-1218209 and IIS-0910611.

References

Robert C Edgar and Serafim Batzoglou. 2006. Multiple sequence alignment. *Current opinion in structural biology*, 16(3):368–373.

Robert C Edgar. 2004. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797.

Raja S Kushalnagar, Walter S Lasecki, and Jeffrey P Bigham. 2014. Accessibility evaluation of classroom captions. *ACM Transactions on Accessible Computing (TACCESS)*, 5(3):7.

Walter Lasecki, Christopher Miller, Adam Sadilek, Andrew Abumoussa, Donato Borrello, Raja Kushalnagar, and Jeffrey Bigham. 2012. Real-time captioning by groups of non-experts. In *Proceedings of the 25rd annual ACM symposium on User interface software and technology, UIST '12*.

Martin Lermen and Knut Reinert. 2000. The practical use of the A* algorithm for exact multiple sequence alignment. *Journal of Computational Biology*, 7(5):655–671.

Iftekhar Naim, Daniel Gildea, Walter Lasecki, and Jeffrey Bigham. 2013. Text alignment for real-time crowd captioning. In *Proceedings of the 2013 Meeting of the North American chapter of the Association for Computational Linguistics (NAACL-13)*.

Murat Saraclar, Michael Riley, Enrico Bocchieri, and Vincent Goffin. 2002. Towards automatic closed captioning: Low latency real time broadcast news transcription. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, pages 1741–1744.

Mike Wald. 2005. Using automatic speech recognition to enhance education for all students: Turning a vision into reality. In *Proceedings 35th Annual Conference on Frontiers in Education, 2005. FIE '05.*, pages S3G–S3G, Oct.

Mike Wald. 2006. Creating accessible educational multimedia through editing automatic speech recognition captioning in real time. *Interactive Technology and Smart Education*, 3(2):131–141.

Lusheng Wang and Tao Jiang. 1994. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4):337–348.