

# Computer-Assisted Disfluency Counts for Stuttered Speech

Peter A. Heeman,<sup>1,4</sup> Andy McMillin,<sup>2,4</sup> and J. Scott Yaruss<sup>3</sup>

<sup>1</sup>Center for Spoken Language Understanding, Oregon Health & Science University, Beaverton, OR

<sup>2</sup>Artz Center, Portland, OR

<sup>3</sup>Dept. of Communication Sciences and Disorders, University of Pittsburgh, Pittsburgh, PA

<sup>4</sup>BioSpeech, Lake Oswego, OR

heemanp@ohsu.edu

## Abstract

We present computer tools to help speech-language pathologists (SLP) in counting disfluencies, for both real-time and transcript-based counts. The latter tend to be more precise and show which words are involved in each disfluency. Our approach allows real-time counts to be used as the basis for transcript-based counts. We employ automatic speech recognition to generate a word transcript (for read-speech samples), and then automatically merge the disfluency annotations with the word transcript, and have the SLP review parts of the audio file where a disfluency annotation was placed. This approach results in improved disfluency annotations, and can be done in 3 times real-time.

**Index Terms:** stuttering, disfluency counts, user-interface

## 1. Introduction

Stuttering is a communication disorder characterized by disfluencies that are frequent and disruptive to communication. SLPs use disfluency counts to decide whether a client should be treated, to assess treatment progress, and to document treatment outcomes. They often do disfluency counts in real-time as a client is talking. However, these are not very specific, and cannot be re-examined. SLPs can also use a verbatim transcript approach, in which they first transcribe exactly what was said, and then mark up the transcript with disfluency codes. This method allows more detailed and accurate counts to be obtained.

Unfortunately, few tools exist to assist SLPs with disfluency counts. Automatic approaches to transcribe stuttered speech and count disfluency have been attempted, but Automatic Speech Recognizers (ASRs) have problems even with fluent speech. The only tools that have proven useful are ones for playing and annotating audio files. However, these tools are not optimized for annotating disfluencies, and cannot be used to annotate disfluencies in real-time. Transcribing the actual words is even worse. Thus, SLPs almost exclusively use real-time counts using pencil-and-paper. Our **goal** is to build computer tools that will assist SLPs in counting disfluency, both real-time and transcript-based. For transcript-based counts, we will use speech technology, especially ASR, to speed up this process, so that it is just several times real-time to produce an *annotated verbatim transcript*, rather being over 20 times.

We need to structure the tools around the reality of clinical practice. SLPs are very busy. They can perform real-time counts during the therapy session as the client is speaking; but they might not have extra time to perform transcript-based counts. It would be advantageous if SLPs do not have to decide

beforehand which type of counts they want to create; if they did, they would probably not commit to the extra time needed for transcript-based counts. After doing real-time counts, it would probably be too disheartening to throw out these counts and start over to produce transcript-based counts. Ideally, the effort expended in producing the real-time counts can be used as a starting point in producing the transcript-based counts.

In this paper, we present a computer tool that allows an SLP to perform real-time counts. We also present a method that can take the real-time counts, and assist the SLP to turn them into transcript-based counts. We then present a user-study in which we evaluate how well this approach works. This approach uses read-speech samples, which are extensively used in therapy.

## 2. Producing Transcript-Based Counts

Below, we give our 5 step process for creating transcript-based disfluency counts.

**Step 1:** The SLP annotates the disfluencies in real-time, perhaps as the client is speaking, using 8 categories: sound, word and phrase repetitions, revisions, interjections, blocks, prolongations, and other.

**Step 2:** An ASR uses the text from the story to produce a word transcription (possibly with errors in it).

**Step 3:** A computer program merges the ASR transcript with the SLP's annotations to produce an annotated verbatim transcript. Each of the SLP's annotations are placed on a word in the transcription that it is likely to have occurred on.

**Step 4:** A computer program determines a set of regions for the SLP to review and correct.

**Step 5:** The SLP reviews and corrects the word transcription and disfluency annotation for each region.

With this approach, a relatively complete word-by-word annotated transcription can be produced with a single real-time pass and a set of samples that are re-heard and corrected, if need be.

### 2.1. Real-Time Disfluency Counts (Step 1)

SLPs already do disfluency counts in real-time using pencil-and-paper. By using a computer tool, we can do automatic summarization of the disfluency counts. We can also keep the time-alignment of the disfluency annotations. We developed a prototype annotation tool that plays a pre-recorded audio file and allows the user to annotate disfluencies as they occur. The tool allows a single type of disfluency to be annotated, or multiple types, according to a configuration file. A touch sensitive computer screen can be used with the tool. Each disfluency type has a large labeled button on the screen, which the user can press when a disfluency of that type is heard. The annotation window is shown in the bottom left in Figure 1. The tool also displays the waveform of the audio file. As the user

---

This study was funded by the NIH under grant 1R41DC009944-01 to BioSpeech, where Dr. Heeman is an employee. This potential conflict of interest has been reviewed and managed by OHSU.

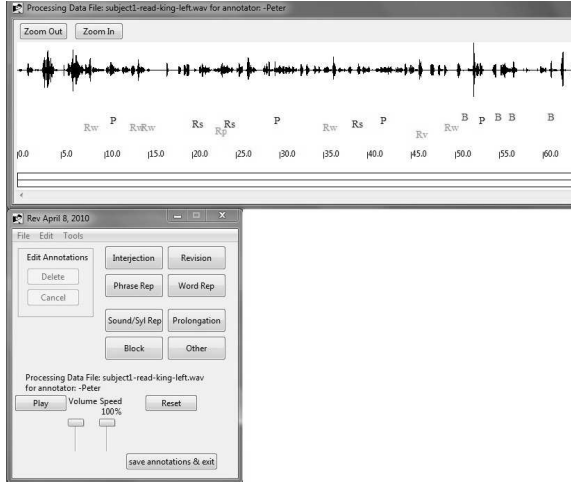


Figure 1: GUI for annotating disfluencies

annotates each disfluency, it is marked on the computer screen time-aligned with where it occurred in the audio file. Due to human reaction time [1], the annotation will not be exactly where the disfluency occurred, but it should be within several seconds.

## 2.2. Running the ASR (Step 2)

As we are using read-speech samples, we can use the story text to guide the ASR. Past approaches have used a grammar-based ASR to account for disfluencies in read-speech samples (cf. [2, 3]). Instead, we use the Sphinx 4 recognizer [4]. This ASR supports statistical bigram language models, which specify how probable any word is to follow any other word in the audio file. We wrote a computer program that builds a bigram model of the story text. Each word in the story is viewed as a different word. For example, in “it’s the crown that makes the king,” the two instances of ‘the’ are treated as two different words, e.g., “the<sub>1</sub>” and “the<sub>2</sub>”. The bigram model forces the ASR to recognize words in the order that they occur in the story by putting all of the probability for a word on the word that follows it in the story (e.g.,  $\text{Pr}(\text{king}|\text{the}_2) = 100\%$ ). To handle deviations from the story text, we could include a garbage model. Instead, we use smoothing by including in the language model a small unigram probability for each word of the story. We report the results of this using the development set and a test set (Table 1). The rate of the unigram probability affects the performance. For  $10^{-30}$ , the WCR is 74.99%. This increases to 87.84% for a probability of  $10^{-13}$ , which we found to be optimal on the development data. The higher probability allows Sphinx to recognize some disfluency patterns without any explicit modeling of them.

We also built a bigram model that allows the full range of backtracks [5]. We allow word repetitions, and multi-iteration word repetitions. We allow phrase repetitions of any words in the current sentence, and to the beginning of the previous two sentences as well. We also model word fragments by adding added dictionary entries for each word consisting of the first two phonemes. We also allow any arbitrary embedding of these disfluencies.

More importantly, we specify a probability for what a speaker might do next, either producing the next word in the story, or backtracking to a previous word in the story. This allows us to capture that it is more likely for a speaker to say the next word of the story, and less likely to make a word repetition, and even less likely to restart to the beginning of the

	Dev. Set	Test Set
Story Text. Unigram at $10^{-30}$	73.60%	74.99%
Story Text. Unigram at $10^{-13}$	85.67%	87.84%
with disfluency modeling	91.36%	90.26%

Table 1: Speech recognition using story text.

sentence. These probabilities were set empirically using the development data. This is a simplified version from our earlier research work [6]. We did not model revisions, as there are a lot of different words that a speaker might use in these disfluencies, and revisions can be partially accounted for by word and phrase repetitions. We experimented with incorporating filled pauses (interjections), but this did not improve performance. We also did not model prolongations or blocks. In all, modeling disfluencies further improved the WCR to 90.26%.

## 2.3. Placing the Annotation Codes (Step 3)

The next step is to guess which words in the ASR output (from Step 2) to place each of the SLP’s disfluency annotations (from Step 1). For example, if the SLP indicated a word repetition at 40.2s, and there was a word repetition in the ASR transcription at 39.6s, we might place the annotation on those words. The result will be an annotated verbatim transcript. We wrote a computer program, *PlaceAnnotations*, for this task. If *PlaceAnnotations* is able to guess correctly, the SLP will not have to adjust the annotation code, thus decreasing the effort needed in Step 5. This program is limited though, as interjections, prolongations, and blocks do not have any marker in the ASR output.

*PlaceAnnotations* first determines which words from the ASR transcription fall within a certain time window from when the SLP annotated it. To determine this window, we had an SLP annotate a set of audio files (the ones used as a *test set* for the ASR in Section 2.2). We used these files as a *development set*. We found that the SLP tended to lag the word that the disfluency occurred on by up to 5s. The lag times vary by the disfluency type. For example, for the revision “as long a long beard clings to his chin”, the SLP might not identify it until she hears the second instance of ‘long’. However, the disfluency should be marked after the first instance of ‘long’, which is where the *backtracking* occurred [5]. Thus, several seconds might have passed. Contrast this to a prolongation, in which an SLP might be able to identify the disfluency while the prolonged word is being said. Hence, we set the time window according to the disfluency type. We then enlarge this window so that the start and end are not in the middle of a word in the ASR transcription.

*PlaceAnnotations* next guesses which word in this range that the disfluency is on. Using the development set, we chose the following heuristics.

- Find a word repetition in the ASR transcription (for word and sound repetitions).
- Find a backtracking in the ASR transcription. A backtracking occurs where the transcription has  $w$  followed by  $w'$ , but where  $w'$  is earlier in the story than  $w$ .
- Find an inter-word silences (for blocks).
- Use word closest to average lag time for the disfluency type.

## 2.4. Determining the Regions (Step 4)

The output of *PlaceAnnotations* might have errors in it, due to its own limitations, errors in the ASR transcription, and mistakes made by the SLP (e.g., using the wrong disfluency code). Hence for each annotation, we determine a *region* of the audio file where the disfluency most likely is, so that the SLP, in Step

5, can correct it if necessary. We propose the following criteria that the regions should meet.

- Should be long enough so that the actual disfluency is in the region as well as enough context to allow the SLP to identify the disfluency from the audio.
- Should not overlap. This way the SLP will not correct the same part multiple times, and it will make it easier to merge the corrections from different regions.
- Should be as short as possible so that the SLP does not need to listen to extraneous speech.
- Should be short enough so that, as much as possible, a region only contains a single disfluency.
- Should start and stop at interword-silences or at word boundaries in the transcribed words.
- The first and last word in the region should be correctly recognized. This should make it easier for the SLP to match up the transcribed words with the audio.

We built a computer program, *MakeRegions*, to create regions that best meet the above criteria. For each disfluency, we start with the region used by *PlaceAnnotations*. We then determine which regions overlap in time with other regions: these are the regions that might need to be merged. We prioritize potential mergers by how many seconds of overlap there is between the regions. A potential merger is accepted if the resulting length is less than 14s or the amount of overlap is greater than 1 second. If the potential merger is rejected, the start time of the second region is adjusted so that it does not overlap with the first region. By prioritizing the potential mergers by amount of overlap, we should be better able to avoid long regions while still merging together regions that are difficult to separate.

### 2.5. Review and Correct (Step 5)

For Step 5, we built a computer tool that allows the SLPs to review and correct the annotated verbatim transcript from Step 3 for each of the regions from Step 4. The tool displays the waveform of the audio file, along with the ASR transcription time-aligned to it. The top window also shows the extent of the current region. Below the waveform window is a second window that has a text field that shows the transcription and annotation of the region. For the disfluency annotations, we use a special letter sequence following the word that it belongs to. For example, the text might be “was only fooling Rp she was only fooling herself she”; where ‘Rp’ is the code for a phrase repetition that *PlaceAnnotations* placed after the word “fooling”. The text interface is primitive, but is sufficient to demonstrate the feasibility of our overall approach.

## 3. Evaluation of Annotations

To evaluate the tools, we compared how well SLPs annotated disfluencies with our computer tools versus pencil-and-paper. The annotation task consists of identifying each disfluency and categorizing it into one of eight categories [7, 8]. We recruited five SLPs who regularly see clients who stutter and regularly use a computer. They received 4 hours of training, of which about one third was about applying the disfluency scheme in real-time with pencil-and-paper and the computer tools.

Subjects did two sessions, each lasting about 1.5 hours, separated by one week. Subjects annotated 20 audio files, each between one and two minutes in length. The audio files were from six different children who stutter. We grouped the audio files by speaker, and included a practice audio file at the beginning of each group, so as to familiarize the SLPs with the speaker’s

Subject	Paper		Annotation		Correction	
c1	46.7%	150	50.4%	162	52.6%	184
c2	48.8%	181	42.6%	163	48.2%	171
c3	50.3%	169	39.9%	134	41.4%	136
c4	48.0%	185	44.5%	165	48.6%	177
c5	40.5%	128	40.3%	131	44.5%	146
All	46.8%	813	43.6%	755	47.1%	814

Table 2: Comparison to reference.

disfluency patterns. With each audio file, SLPs alternated using paper and using the annotation and correction tools (Step 1 and 5). Half of the subjects used paper first, and the rest used the computer tools first. For the second session, subjects switched which method they used for each file.

**Agreement with reference annotations:** To analyze the performance of paper versus the computer tools, we compare the annotations produced using each method to the reference annotations. The reference annotations were generated automatically from another scheme [5], and were annotated by someone without clinical experience. Hence, it might contain mistakes. As we do not have time-stamps for the paper counts, we compare the annotations using *set matching*. We determine the number of annotations of each type that match. This is the number of hits. We divide this by the number of hits plus the number of annotations that do not have a match (from either file).

Table 2 gives the results. The score for each dialogue is weighted by the number of annotations. This metric indicates that the paper method is doing better than the annotation tool (Step 1): 46.8% agreement versus 43.6%. This difference is significant using the Wilcoxon signed-rank test with each story and subject as a data-point,  $N=83$ ,  $z=2.25$ ,  $p<.03$ , two-tailed. For this test, each dialogue is given the same weight, but we excluded one of the stories, as it just had three annotations in the reference annotation.

Now consider the agreement after the correction tool (Step 5), which increases from 43.6% to 47.1%. In fact, all 5 subjects improve their agreement with the correction pass. This improvement is statistically significant,  $N=72$ ,  $z=2.87$ ,  $p<0.005$ , two-tailed. In addition, subjects did better after the correction tool than paper, but the difference is not statistically significant.

**Inter-coder agreement:** We also compute inter-coder agreement for each method. Here, we determine the average pairwise agreement between each pair of subjects using a particular method. The rationale is that if a method results in higher agreement between subjects than they are probably agreeing on the correct answer. The advantage of this analysis is that it does not rely on the reference annotations. Averaging over all pairs of subjects, the average agreement for paper is 53.8% and 51.2% for the annotation tool. However, the difference is not statistically significant using the Wilcoxon Signed Rank Test for paired samples, in which each speech sample is a datapoint  $N=178$ ,  $z=1.22$ , NS, two-tailed.

After the correction-pass, the intercoder agreement is 55.7%. The difference with the annotation-pass is significant,  $N=163$ ,  $z=2.43$ ,  $p<0.02$ , two-tailed Wilcoxon signed-rank test. Just as with agreement with the reference annotations, this analysis indicates that the second-pass is resulting in improved annotations. We also compared paper against the correction-pass: 53.8% versus 55.7%. This difference is not significant via the Wilcoxon signed-rank test,  $N=179$ ,  $z=1.5$ , NS.

**Discussion:** The subjects performed worse with the annotation tool than with paper. By looking at the last columns of Table

	Total	Rate
Rs Sound Repetition	140	45.71
Rw Word Repetition	158	48.73
Rp Phrase Repetition	112	55.36
Rv Revision	62	58.06
B Block	47	34.04
P Prolongation	162	33.95
Int Interruption	0	n.a.
O Other	3	66.67
All	684	45.61

Table 3: Placing the Annotation.

2, it seems that part of the problem is that subjects are simply annotating fewer disfluencies (755 versus 813). In a post-experiment questionnaire, several subjects noted that they had difficulty locating the correct annotation button. More work is needed to address this. However, there was a statistically significant improvement in the annotations after the correction pass.

#### 4. Evaluation of Intermediate Steps

To give further insight into the functioning of the tools, we evaluate the individual steps.

**Step 3:** The ability to correctly place the annotation codes depends on the quality of the heuristics, but also on the ASR word transcription, and the quality of the subjects' annotations. To allow us to evaluate the heuristics, we built a computer program that scores the SLPs' annotations from Step 1. The program, *ScoreStep1*, aligns the SLP's annotations with the disfluencies in the reference annotation, and then scores each one as to whether it is the correct type, wrong type, or a false positive. *ScoreStep1* takes into account the distance from the annotation to the reference disfluency, whether they are the same annotation code, or are confusable types (such as word and sound repetitions). We verified this program on the development data.

*ScoreStep1* matched 684 of the subjects' disfluencies with ones in the reference annotations (Table 3). Of these, *PlaceAnnotations* placed 312 on a word in the ASR transcription that overlaps in time with the word in the reference transcription that has the disfluency marked on it, giving a rate of 45.6%. We also give the placement rate for each disfluency type (based on what the SLP coded). We see that the lowest rates are for blocks and prolongations, as the ASR does not model them.

**Step 4:** The 755 disfluencies that the SLPs annotated were placed into 600 regions (Table 4). The average length of each region is 4.1s. The average length of the longest 10% is 7.4s. The regions accounted for less than one third of the duration of the audio files. Of the 684 annotations that were matched to the reference annotations, 592 were inside of a region, and 92 were outside. Thus, *MakeRegions* seems to be accurate only 86.5% of the time (assuming that *ScoreStep1* is correct). More work is needed as incorrect regions limits the usefulness of Step 5.

Of the 600 regions that *MakeRegions* created, 560 (93.3%) had the first and last word in the region corrected transcribed

Average number of annotations per region	1.26
Average length of region	4.1s
Average length of top 10%	7.4s
Region includes words in disfluency	86.5%
First and last word of region correct	93.3%
Percentage of Audio Files	30.5%

Table 4: Analysis of regions.

	Single		Multiple	
	No change	Change	No Change	Change
Number of regions	161	317	19	103
Ave. length	3.7s	3.6s	5.9s	6.2s
Time spent	9.3s	30.4s	27.4s	51.7s
Real-time factor	2.5	8.5	4.6	8.4

Table 5: Time spent versus whether a change was made.

by the ASR; 11 had the first word correct; 25 had the last word correct; and 4 had neither.

**Step 5:** The SLPs spent on average 28.3s on each region in the correction pass, giving a real-time factor of 6.9. Averaging over the entire length of the audio file gives a real-time factor of 2.1. In a questionnaire, several SLPs commented that this time was well spent, as they are reviewing the critical areas of the speech. The entire time with both passes has a real-time factor of 3.1, which is substantially less than producing an annotated verbatim transcript by hand.

We also contrast regions where no change was needed that have a single versus multiple disfluencies (Table 5). Here the time-factor increases from 2.5 to 4.6. This shows that it is beneficial to isolate each disfluency into its own region if possible. For regions with a single disfluency, we see that if a change is needed, the time factor increases from 2.5 to 8.5. Thus, as Steps 2 and 3 improve, even less time will be needed by the SLP.

#### 5. Conclusion

In this paper, we presented the results from a study using computer tools to assist SLPs in annotating disfluencies in stuttered speech. We found that SLPs using the tool for annotating disfluencies in real-time performed slightly worse than using pencil-and-paper. This might be due to a deficit in the user interface. We also proposed a method in which these real-time counts can form the basis of transcript-based counts for read-speech samples. The transcript-based counts can be produced at a rate of 3 times real-time, and agreed better with a reference annotation than the real-time counts. These results show the feasibility of our tools. By allowing transcript-based counts to be done efficiently, word-level information about disfluencies can be made available to the SLP, which might result in improved therapy.

#### 6. References

- [1] D. B. Fry, "Simple reaction-times to speech and non-speech stimuli," *Cortex*, vol. 11, no. 4, pp. 355–360, 1975.
- [2] E. Nöth, H. Niemann, T. Haderlein, M. Decher, U. Eysholdt, F. Rosanowski, and T. Wittenberg, "Automatic stuttering recognition using hidden markov models," in *ICSLP*, Beijing, Oct. 2000.
- [3] K. Hollingshead and P. Heeman, "Using a uniform-weight grammar to model disfluencies in stuttered read speech: a pilot study," CSLU, OHSU, Tech. Rep. CSLU-04-002, Jul. 2004.
- [4] P. Lamere, P. Kwok, E. Gouvea, B. Raj, R. Singh, W. Walker, M. Warmuth, and P. Wolf, "The CMU sphinx-4 speech recognition system," in *ICASSP*, Hong Kong, 2003.
- [5] P. Heeman, A. McMillin, and J. S. Yaruss, "An annotation scheme for complex disfluencies," in *ICSLP*, Pittsburgh PA, Sep. 2006.
- [6] P. Heeman and J. Allen, "Speech repairs, intonational phrases and discourse markers: Modeling speakers' utterances in spoken dialog," *Computational Linguistics*, vol. 25, pp. 527–572, 1999.
- [7] J. S. Yaruss, "Real-time analysis of speech fluency: Procedures and reliability training," *American Journal for Speech-Language Pathology*, vol. 7, no. 2, pp. 25–37, 1998.
- [8] J. S. Yaruss, M. Max, R. Newman, and J. Campbell, "Comparing real-time and transcript-based techniques for measuring stuttering," *Journal of Fluency Disorders*, vol. 23, pp. 137–151, 1998.