

# Diagnostics for Debugging Speech Recognition Systems

Miloš Cernák

Institute of Informatics, Slovak Academy of Sciences  
Dúbravská cesta 9, 045 07 Bratislava  
`milos.cernak@savba.sk`

**Abstract.** Modern speech recognition applications are becoming very complex program packages. To understand the error behaviour of the ASR systems, a special diagnosis - a procedure or a tool - is needed. Many ASR users and developers have developed their own expert diagnostic rules that can be successfully applied to a system. There are also several explicit approaches in the literature for determining the problems related to application errors. The approaches are based on error and ablativ analyses of the ASR components, with a blame assignment to a problematic component. The disadvantage of those methods is that they are either quite time-consuming to acquire expert diagnostic knowledge, or that they offer very coarse-grained localization of a problematic ASR part. This paper proposes fine-grained diagnostics for debugging ASR by applying a program-spectra based failure localization, and it localizes directly a part of ASR implementation. We designed a toy experiment with diagnostic database OLLO to show that our method is very easy to use and that it provides a good localization accuracy. Because it is not able to localize all the errors, an issue that we discuss in the discussion, we recommend to use it with other coarse-grained localization methods for a complex ASR diagnosis.

**Key words:** automatic speech recognition, fault diagnosis

## 1 Introduction

With increasing complexity of the ASR systems (advanced algorithms, multi- and many-processors implementations and so on), effective diagnostics has to be proposed. It is obvious mainly in the commercial environment where the time for error analysis is often a critical parameter and a fast solution, an automatic or semi-automatic method, is required. Academic groups can profit from such diagnostics as well, as it could improve understanding of the error behaviour of the system, which could result in better tuning of the systems.

There are several approaches in the literature that propose the diagnostic methods. They try to answer the question: How much error is attributable to each of the components? **Error analysis** tries to explain the difference between current performance and perfect performance. As a ground-truth a forced alignment is used, and then an analysis is done on error regions with respect to

partial acoustic and language modeling scores. In [1] a diagram, a sequence of tests applied to the error region produced by an alignment of recognized and forced aligned phones, predicts a category to each error region. Front-end, language modeling, and the search belong to the main error categories, and this is directly linked to a concrete component of the ASR in the test. A similar approach was proposed in [2], where the error-causing components were exactly the three mentioned above. Binary decision trees are mostly employed as error predictors [3], this being a popular approach not only in ASR (an example is [4]). **Ablative analysis**, on the contrary, tries to explain the difference between a much poorer baseline performance and current performance. A good example can be found in [5]. The authors present a system with a poor baseline with an average WER of 51.1 %, and with application of more components in a pipeline or a component combinations, they achieved an average WER of 32.0 %, with a calculated contribution of each component to the error rate. Both error and ablative analyses offer coarse-grained localization of the error-causing components.

To achieve fine-grained localization we can either partition existing components to sub-components, which might be a laborious task without any effect on the system performance, or we can directly analyse the implementation of the components. In this paper we explore the second approach, focusing on program-spectra based error localization [6]. The key idea is linking dynamic program events (program spectra) with erroneous program behaviour at the level of code lines, and localize the lines that match error indicators most closely. Diagnostic accuracy is then computed to rate the finding for a human examiner.

The remainder of the paper is structured as follows. In the next Section 2 we introduce software fault diagnosis, which we apply to a special test scenario in the experimental part of the paper in Section 3. Finally in Section 4 we discuss pros and cons of the approach.

## 2 Software fault diagnosis

Model-based diagnosis is a central point of the fault diagnosis theory. The model of a system serves as a definition of its intended behavior, and may contain additional information about its composition and operation. While model-based diagnosis has successfully been applied for diagnosing complex mechanical systems, its application to software has proven to be difficult. Spectra-based fault localization has been proposed recently for complex software system [6].

Central to the discussion of diagnosis are the notions of failure and fault. A failure is a discrepancy between expected and observed behavior, and a fault is a property of the system that causes such a discrepancy. Fault localization is a task of finding a place in the program, which should be responsible for a failure. The fault in terms of software can be considered in the worst case as a bug in the program code.

A program spectrum is a collection of data that provides a specific view on the dynamic behavior of software. This data is collected at run-time using program profiling tools. In this paper we work with code coverage spectra that indicate

$$\begin{array}{c}
\begin{array}{c} P \text{ parts} \\ \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1P} \\ x_{21} & x_{22} & \dots & x_{2P} \\ \vdots & \vdots & \ddots & \vdots \\ x_{R1} & x_{R2} & \dots & x_{RP} \end{bmatrix} \end{array} \\
R \text{ spectra}
\end{array}
\begin{array}{c}
\text{errors} \\
\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_R \end{bmatrix}
\end{array}$$

$$s_1 \ s_2 \ \dots \ s_P$$

**Fig. 1.**  $R \times P$  binary program spectra matrix and  $R \times 1$  error vector. Similarity coefficients  $s_1, s_2, \dots, s_P$  are calculated for each column of the spectra matrix.

whether or not a block of code was executed in a particular run. By the block we mean a C/C++ language statement (but the method is also applicable for different languages). The tutorial [6] gives an excellent overview of the program analysis technique. Once the potential spectra-based fault has been localized, the assessment requires understanding of the correlation between localization and program behavior.

## 2.1 Fault localization

A test set for regression testing of a speech recognizer consists of  $R$  utterances. The hit spectra of  $R$  ASR runs constitute a binary matrix, whose columns correspond to  $P$  different blocks of the program code. The information about failure detection (misrecognition) constitutes another column vector, the error vector (see Fig. 1). This vector can be construed as representing error indicators. Fault localization essentially consists of identifying which column vectors resemble the error vector most [6].

Similarity coefficients are used as quantifiers of resemblances between the columns of program spectra matrices. The coefficients are used to express the similarity  $s_j$  of the column  $j$  and the error vector. As an example, below are three different similarity coefficients:

- The Jaccard similarity coefficient:

$$s_J(j) = \frac{a_{11}(j)}{a_{11}(j) + a_{01}(j) + a_{10}(j)} \quad (1)$$

- The Tarantula coefficient:

$$s_T(j) = \frac{\frac{a_{11}(j)}{a_{11}(j) + a_{01}(j)}}{\frac{a_{11}(j)}{a_{11}(j) + a_{01}(j)} + \frac{a_{10}(j)}{a_{10}(j) + a_{00}(j)}} \quad (2)$$

- The Ochiai coefficient:

$$s_O(j) = \frac{a_{11}(j)}{\sqrt{(a_{11}(j) + a_{01}(j)) * (a_{11}(j) + a_{10}(j))}} \quad (3)$$

where  $a_{pq}(j) = |\{i \mid x_{ij} = p \wedge e_i = q\}|$ , and  $p, q \in \{0, 1\}$ . The columns with the highest similarity coefficients are considered as the localization of  $j$ -th program code block.

## 2.2 Diagnostic accuracy

In general, the following evaluation metrics is used for diagnostic accuracy [7]. Let  $d \in \{1, \dots, P\}$  be the index of the block that we know to contain a fault. For all  $j \in \{1, \dots, P\}$ , let  $s_j$  denote the similarity coefficient calculated for block  $j$ . Then the ranking position is given by

$$\tau = \frac{|\{j \mid s_j > s_d\}| + |\{j \mid s_j \geq s_d\}| - 1}{2} \quad (4)$$

Accuracy, the percentage of blocks that need not to be considered when searching for the fault by traversing the ranking, is defined as

$$q_d = \left(1 - \frac{\tau}{P-1}\right) \cdot 100\% \quad (5)$$

## 3 Experiments

This section describes the toy experiment of ASR program spectra-based localization. The objective of the experiment was as follows: we injected an accuracy drop between several batch runs of the test under different conditions and the task was to accurately localize a part of the program responsible for the accuracy drop. A free speech recognizer<sup>1</sup> was used for logatome recognition with uniform distribution of grammar weights. Logatome recognition task was selected as a standard diagnostic task with the vocabulary consisting of CVC isolated words in comparable quantity as used by Steeneken in his diagnostic work [8], and we use it often for validation of our diagnostic methods.

### 3.1 ASR setup

The speech database used for ASR experiments is the Oldenburg Logatome Corpus (OLLO) [9]. It contains 150 different non-sense utterances (logatomes) spoken by 40 German and 10 French speakers. Each logatome consists of the combination of consonant-vowel-consonant (CVC) or vowel-consonant-vowel (VCV) with the outer phonemes being identical. To provide an insight into the influence of speech intrinsic variabilities on speech recognition, OLLO covers several variabilities such as speaking rate and effort, dialect, accent and speaking style (statement and question). The OLLO database is freely available at <http://sirius.physik.uni-oldenburg.de>.

Each of the 150 logatomes was recorded in six conditions (speaking rate: fast and slow; speaking effort: loud and soft; speaking style: spoken as question

<sup>1</sup> The source code is freely accessible at <http://www.torch.ch>

and normal) with three repetitions. This results in 2,700 logatomes per speaker. Influences of dialect may be investigated, as speakers without a clearly marked dialect and from four different dialect/accent regions were recorded. Utterances of ten speakers with no accented speech (five speakers for training and five speakers for testing) were used for ASR tests.

A Hidden Markov Models (HMM) and Gaussian Mixture Models (GMM) based speech recognition system was trained using public domain machine-learning library TORCH [10] on the training set that consists of 13446 logatome utterances. Three states left-right HMM models were trained for each of the 26 phonemes in the OLLO database including silence as well. Three acoustic models (AMs) were trained, with 8, 16 and 32 GMMs, respectively. Diagonal covariance matrices were used to model the emission probability densities of the 39 dimensional feature vectors - 13 cepstral coefficients and their derivatives ( $\Delta s$ ) and double derivatives ( $\Delta\Delta s$ ). The phoneme HMMs were connected with no skip. We trained and tested the ASR system with MFCC feature set, calculated using HTK `hcopy` tool. We calculated MFCC vectors every 10 msec using windows of size 25 msec.

### 3.2 Error injection in grammar constrains

The test set size was 13.5k logatomes. We injected accuracy drop with (de)activation of isolated word recognition within an experiment: running isolated logatome recognition the system's accuracy was higher than running the test with the word loop grammar. To test the robustness of the localization, 3 experiments were performed, each with different AM. Performances of the tests are shown in Table 1.

**Table 1.** *Accuracies of particular experiments.*

Experiments	AM	isolated	non-isolated
Experiment 1	8 GMM	71.83	69.40
Experiment 2	16 GMM	74.96	72.94
Experiment 3	32 GMM	75.87	73.39

The task was to construct program spectra for each experiment from  $R$  runs (13.5k) of isolated and  $R$  runs of non-isolated logatome recognitions (this constituted  $2R \times P$  program spectra matrices), and to localize a part of the ASR implementation responsible for the accuracy drop. Figure 2 lists a simplified grammar construction used in the experiments. Program lines 6-8 constitute the part of the program executed for non-isolated word recognition grammar.

A program spectrum was obtained using the standard GNU code coverage analysis tool `gcov`. This tool was used in conjunction with `gcc` to dump run-time code coverage of the ASR. The ASR was compiled with the `gcc -fprofile-arcs -ftest-coverage` options and no optimizations, and with each execution of the

```

        // from leading silence to a start state
1: gramm ar.transitions[1][0] = true;
2: for (int i=0;i<n_words-1;i++) {
        // from the word to the leading silence
3:   grammar.transitions[i+2][1] = true;
        // from the trailing silence to the word
4:   grammar.transitions[n_words+1][i+2] = true;
5:   if (!isolated) {
        /* A block of interest */
6:     for (int j=0;j<n_words-1;j++)
7:       if (!no_self_transitions || i!=j)
            // word to word transition
8:         grammar.transitions[j+2][i+2] = true;
9:   }
10: }
11: grammar.transitions[n_words+2][n_words+1] = true;

```

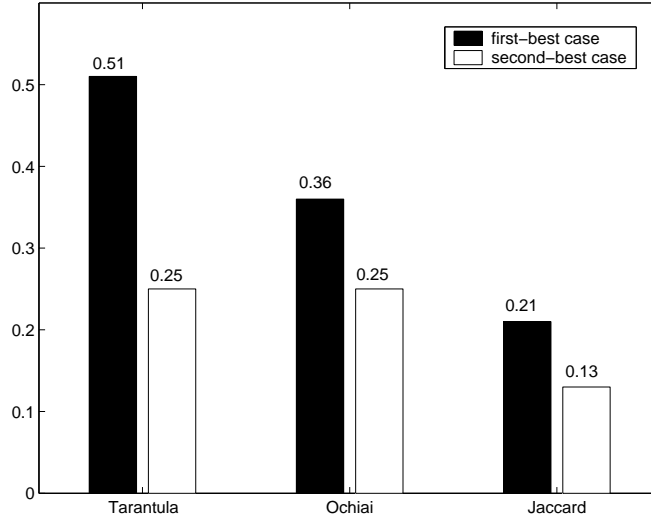
**Fig. 2.** Simple grammar construction code

program, a separate data file `.gcda` was created for each object file. Using data files and source files `gcov` tool then generated profiling `.gcov` files, which described which lines of code were actually executed and how often each line of code was executed. This information was used for program spectra matrix construction, where lines of code were transformed to columns with a binary value indicating the execution or non execution of the line in a particular program run. Program executions formed rows of the matrix. The last column of the matrix represented recognition failures (detected insertions, deletions or substitutions) with values of 1 for errors detected.

We generated three spectra, each for different experiment. The first part of the spectra was from isolated logatome recognition, and the second part of the spectra was from non-isolated recognition. We calculated  $s_J(j)$ ,  $s_T(j)$ , and  $s_O(j)$  similarity coefficients for the obtained spectra. We chose the parameter of different number of mixtures in GMM models just for getting more tests with different WER. Because we ran speech recognition program  $R$  times for each test, we also obtained static parts of the spectra ( $x_{ij} = 1$ , where  $\forall i$  is  $j$  constant  $j = k$ ,  $k \in \{1, \dots, P\}$ ). Such a static spectra must be penalized to achieve good diagnostic performance, and we estimated this penalty as  $s_j = \frac{1}{n} * s_j$ , where  $n$  is a number of ASR parameter sets in the experiment. In our case  $n = 2$ , as we first ran non-word isolated speech recognition, and then word isolated speech recognition.

We achieved good diagnostic accuracy  $q_{6-8}$  in our experiments (the index of  $q$  represents the code lines from Fig. 2). The accuracy was 99.995% for all the similarity measures and for the tests, except for  $s_J(j)$  and  $s_O(j)$  in 8GMM test case, where  $q_{6-8}$  was 75.51%.

In addition to  $q_{6-8}$ , Fig. 3 shows how the measures scored first-best and second-best cases for fault localization. Tarantula measure  $s_T(j)$ , the only measure with  $q_{6-8} = 99.995\%$  for all three test cases (AMs), has also the highest difference in scoring of first two localization candidates. This difference may also indicate the quality of fault localization.



**Fig. 3.** Differences in scoring first two localization candidates for 16GMM test case.

## 4 Discussion

The presented fine-grained localization has the following advantages: First, it can be applied without any code modification in a fully automatic manner, as it is a black-box diagnosis method applicable to any ASR system without providing access to individual components within the system. Second, the diagnosis is easily implemented using existing profiling tools (such as Linux `gprof` or Windows `VSPerfMon`). Third, if the program spectrum and its further processing is well designed, the method offers the finest localization revealing possible errors ranging from software bugs via badly tuned ASR parameters toward the ultimate goal of providing an insight into erroneous behavior of computer speech recognition.

However, there are still limitations to the application of the diagnosis as described in this paper. Faults, that does not change the program spectra with a failure, cannot be detected. As it sticks to the implementation, even the best code in the world will fail with poor acoustical and language models used; but this can be well detected by existing coarse-grained decision-tree-based diagnosis

methods. It seems that their combination for complex diagnosis is needed, and we plan to examine this in the future.

The diagnosis is open to using different runtime coverage of entities such as statements, branches and du-pair coverage. Experiments show that the choice of coverage type can significantly affect the effectiveness of fault localization [11]. In addition, different similarity measures can be further proposed. It would be worth to predict more localities that all simultaneously contribute to the error. For now only single locality was considered. All of these issues show the directions of our future work.

In the spirit of making research reproducible [12], the diagnostic tools used, including batch scripts, data descriptions, ASR and diagnosis setup, are available at [http://www.ui.savba.sk/speech/milos\\_web\\_data/asr\\_diagnosis.zip](http://www.ui.savba.sk/speech/milos_web_data/asr_diagnosis.zip).

**Acknowledgements** The work has been funded with support from the VEGA grant No. 2/0138/08 entitled “Robust speech technologies for information systems in Slovak and their diagnostics”, and by Slovak Research and Development Agency under research project APVV-0369-07.

## References

1. Chase L.L.: Error-Responsive Feedback Mechanisms for Speech Recognizers. Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, USA, April 1997. Also available as Robotics Institute Tech. Report # CMU-RI-TR-97-18.
2. Nanjo, H., Lee, A., Kawahara, T., Automatic diagnosis of recognition errors in LVCSR systems, In ICSLP-2000, vol.2, pp. 1027–1030 (2000)
3. Cernák, M.: A Comparison of Decision Tree Classifiers for Automatic Diagnosis of Speech Recognition Errors. In: Computing and Informatics, vol. 3, July 2010.
4. Alice X. Zheng, Jim Lloyd, Brewer, E.: Failure diagnosis using decision trees. In ICAC 04: Proceedings of the First International Conference on Autonomic Computing (ICAC04), Washington, DC, USA, 2004, pp. 36–43, IEEE Computer Society.
5. Picheny, M., Nahamoo, D.: Towards Superhuman Speech Recognition. In: Benesty, J., Sondhi M.M., Huang, Y. (eds.) Springer Handbook of Speech Processing, pp. 597–616. Springer, Heidelberg (2008)
6. Zoetewij, P., Abreu, R. and van Gemund, Arjan J.C.: Software Fault Diagnosis, A tutorial in TESTCOM/FATES, Tallinn, Estonia, June 26–29 (2007)
7. Abreu, R., Zoetewij, P., van Gemund, A.J.C.: On the Accuracy of Spectrum-based Fault Localization, In Proceedings of TAIC PART (2007)
8. Steeneken, Herman J.M., Varga, A.: Assessment for automatic speech recognition: I. Comparison of assessment methods, Speech Communication, 12(3):241–246, (1993)
9. Wesker, T., Meyer, B., Wagener, J., Anemuller, J., Mertins, A. and Kollmeier, B.: Oldenburg logatome speech corpus (OLLO) for speech recognition experiments with humans and machines, In Interspeech, pp. 1273–1276 (2005)
10. Collobert, R., Bengio, S., Marthoz, J.: Torch: a modular machine learning software library, Technical Report IDIAP-RR 02-46, IDIAP, (2002)
11. Santelices, R., Jones, J.A., Yu, Y., Harrold, M.J.: Lightweight Fault-Localization Using Multiple Coverage Types. In Proc. of the ICSE, pp. 56–66, 2009.
12. Vandewalle, P., Kovačević, J., Vetterli, M.: Reproducible Research in Signal Processing: What, Why, and How. IEEE Signal Processing Magazine (37), May 2009.