Very Large Vocabulary Voice Dictation for Mobile Devices

Jan Nouza, Petr Cerva, Jindrich Zdansky

SpeechLab, Institute of Information Technology and Electronics
Technical University of Liberec, 461 17 Liberec, Czech Republic
{jan.nouza, petr.cerva, jindrich.zdansky,}@tul.cz

10.21437/Interspeech.2009-295

Abstract

This paper deals with optimization techniques that can make very large vocabulary voice dictation applications deployable on recent mobile devices. We focus namely on optimization of signal parameterization (frame rate, FFT calculation, fixed-point representation) and on efficient pruning techniques employed on the state and Gaussian mixture level. We demonstrate the applicability of the proposed techniques on the practical design of an embedded 255K-word discrete dictation program developed for Czech. Its real performance is comparable to a client-server version of the fluent dictation program implemented on the same mobile device.

Index Terms: large vocabulary dictation, mobile device

1. Introduction

Voice-controlled applications, particularly dictation programs, have already proven their viability on classic PCs, where the efficiency of voice input starts to compete with traditional keyboard typing. The main advantage of a modern PC is that it offers a combination of high computation power and large memory, i.e. the two key parameters required for a practical implementation of complex speech recognition procedures.

To seek for new challenges, many speech researchers are shifting their attention from desktop PCs to mobile devices, such as PDAs or smart phones. The motivation is obvious. These miniature devices offer very poor (and uncomfortable) possibilities for entering text data. Their keyboards are small, usually display only a subset of all available characters, often require a special stylus, and last but not least, stylus typing is rather slow and suffers from frequent errors. Therefore, voice input implemented into these devices would be appreciated even more than in case of standard PCs.

Unfortunately, it is not easy to meet all the requirements that most mobile device users and developers expect from speech technology [1, 2]. The reason is that the desktop and handheld devices differ significantly in their technical parameters, as it is summarized in Table 1. In general, a recent mobile device (MD) operates with a CPU that is at least 5 times slower than that in a PC. Also the operational memory is 5 to 10 times smaller. What most MDs lack, is a Floating Point Unit (FPU), a component that is essential for complex speech processing algorithms. Moreover, there is a significant difference in voice signal quality. While on a PC we can use a high quality close-talk microphone supported by a standard sound card, in case of a mobile device we must use either the embedded miniature microphone or an external hands-free set. Unfortunately, none of these two options provide signal quality that is as good as that on a PC. (For comparison, see section 4.) Last but not least, we should mention also the problem of power supply. Batteries used in MDs have very limited capacity and this fact must be taken into account when developing programs that require CPU intensive operation.

Table 1. *Parameters of desktop and mobile devices.*

Parameters	Desktop PC	Mobile device
CPU speed	2 – 3 GHz	0.4 – 0.6 GHz
Memory	1 – 2 GB	0.1 – 0.25 GB
FPU	included	missing
Voice channel quality	high	lower
Power supply capacity	not so critical	very limited

Technical limitations of MDs can be overcome in several ways. An often used approach is to utilize distributed computing and let the CPU intensive procedures run on a remote server as shown, for example, in [3]. The main drawback of this solution is that during dictation, the MD must be connected to the server via one of the available wireless networks (e.g. WiFi, GPRS, etc). Another approach consists in the development of a special recognizer that can operate within the limits of MDs. This can be done either by porting and optimizing existing engines (e.g. pocket versions of Janus [4] or Sphinx [5]) or by designing mobile devices and platforms that already support speech recognition, like it has been done by Nokia [6, 7]. Unfortunately, even the latest products of this type are capable of managing vocabularies whose size is restricted to several thousands (at maximum tens of thousands) of words. Hence, they can be deployed in specific tasks only. Yet, this approach becomes totally unusable in inflected languages, where even domain restricted tasks require tens to hundreds of thousands words.

2. Motivation and goals

In this paper, we describe our solution to the very-large-vocabulary voice dictation problem. The task we solved was challenging: build a standalone speech recognizer for Czech that would be practically deployable on recent PDAs and smart mobile phones. We had to find a way to manage vocabularies with 250K+ words and make voice input faster than the typing with a stylus (supported by the T9). Our approach employs a discrete speech recognition engine optimized for speed, memory usage and power consumption. Using the touch screen for disambiguation and correction of voice input, we almost eliminate the use of the stylus [8].

The engine has been designed as language independent, though we had in mind Czech users as the first target group. Czech is an inflected language with more than one million distinct word-forms. If we want to ensure practically applicable dictation of common texts, the OOV rate must not be higher than 1 %. Our previous study [9] showed that in that case the lexicon had to contain at least 250K words.

To make the development fast and efficient, we wanted to re-use our previously created codes and modules, all written for the Microsoft Windows platform. Its 'pocket' version, the Windows Mobile (WM), has become quite popular among the producers of PDAs and smart phones (e.g. HP, Samsung, HTC, and others) as well as among MD users, recently. Therefore, we decided to port our engine to the WM OS.

The other practical requirements on the dictation program can be briefly summarized as follows: latency shorter than 0.5 seconds, speaker independent (but gender specific) operation, optional speaker adaptation, and on-line lexicon modification (a possibility to add new words during dictation).

For evaluation and comparison purposes we have developed also a client-server based fluent speech recognizer running on the same type of the MDs. We briefly describe its features in section 4.

3. Standalone dictation program

For the development of the dictation program we could use modules that had been already designed and deployed in the software called MyDictate [10]. The software is aimed at motor-handicapped users who cannot use keyboard and mouse. All actions, i.e. dictated words as well as correcting, editing and formatting commands are controlled by voice, using a vocabulary containing about 550K words. The words and commands have to be pronounced in isolated way with a short pause (> 0.5 s) between them. This solution proved to be both acceptable as well as robust for the target user group, which is documented by almost 100 installations distributed so far. A typical WER value is about 9 % in the speaker independent mode, or 6 % in the speaker adapted mode.

The first idea was to port this software to MDs equipped with StrongARM family of processors and Windows Mobile 6.1 OS. This initial attempt failed because: a) the 550K-word vocabulary and its language model were too large for MD's operation memory, and b) even if the vocabulary size was reduced to a half, the recognition time was extremely long (almost 30 seconds per word). We realized that for successful porting we had to *analyze* and consequently *optimize* namely the following settings, parameters, and procedures:

- *signal sampling rate and frame rate,*
- *FFT and MFCC calculation,*
- *acoustic model representation,*
- *likelihood computation and efficient decoding scheme.*

3.1. Baseline recognition engine

To allow for later performance comparison with the optimized versions, we briefly summarize the settings of the original engine: signal sampling rate 16 kHz, frame window length 25 ms (400 samples), window shift 10 ms (160 samples), 512-point FFT, 39 MFCC parameters, 41 phones + 7 noises, 3-state context independent HMMs, 96 gaussians per each state, tree-structure lexicon of 255K words, unigram LM. It should be noted that the mentioned lexicon size and the LM in form of unigrams were chosen here to make the baseline system comparable to what can be fitted to the latest MD processors. The baseline engine was implemented in C language using 32-bit float types for the variables, like features, likelihoods, etc.

To analyze the impact of various settings and alternative implementations on speed and recognition score, we created a development set of 2400 words recorded by 4 persons on 2 MD types. The WER for the baseline system was 14.13 %.

3.2. Optimization of parameterization process

Among the first investigated parameters there were sampling and frame rates. While an attempt to reduce the sampling frequency from 16 kHz to other popular values 11.025 kHz and 8 kHz did not yield any practical benefit, the change of the frame window settings – the window length from 400 to 256 samples and the window shift from 160 to 224 samples – speeded up parameterization without any negative impact on the baseline WER. The shorter window allowed us to apply the 256-point FFT instead of the 512-point one and the longer shift slightly reduced the number of frames to be decoded.

The calculation of the FFT is one the bottlenecks on devices that lack a FPU. Many researchers, therefore, try to implement it in fixed-point arithmetic. The resulting code is about one order faster but because of repeated rounding, a WER increase in range 2 to 5 % is often reported [4, 5].

In our system we could use another approach. We may postpone the start of the decoding procedure to the moment when the utterance to be recognized has finished. Hence, during speech recording, the CPU has enough time to compute the complete sequence of feature vectors. We have optimized the FFT procedure for the given (256-point) size at the algorithmic level and have done the same also for the conversion to the MFCC vector, but we employed the floating point routines provided by the compiler. In this way we obtain the feature vectors in their original precision in time that is a fraction (about 1/2) of the window shift period.

This signal preprocessing component is *the only module* in the entire recognizer *where floating point numbers and operations are used.* At the output from this component, the MFCCs are converted into 32-bit integers. The conversion is done via multiplying the features by factors that are powers of two. (This allows us to use fast bit-shifting scaling operations later.) Each feature type (static, dynamic and acceleration), has its own factor determined by feature histogram analysis.

3.3. Optimization of acoustic model

The use of context-independent phone models (monophones) significantly contributes to the speed/memory optimization of the decoding process. A small number of distinct states (48 HMMs x 3) allows for minimizing the need for likelihood computation as the already computed values can be retrieved from cache memory. The lower modeling accuracy of monophones (vs. triphones) can be partly eliminated by larger numbers of mixtures in each state. In the baseline system we used 96 mixtures, in the MD version we can go up to 32 or 64 mixtures due to the fast likelihood computation routine.

Two gender-dependent (male and female) models have been trained on approx. 60 hours of acoustic data. It was mainly fluent speech recordings (primarily used for training of continuous ASR models). To make the AM at least partly matched to the target conditions, the training set was enhanced by adding two hours of discrete speech recorded on two different MD types.

After training, the AM was transformed to the form where each gaussian is represented by its mean vector $\boldsymbol{\mu}_m$, inverse covariance vector $\mathbf{s}_m$ and constant C_m, i.e. the parameters that fit to eq. (2). All values were converted into integers - again by using parameter-specific power-of-two factors.

3.4. Optimization of likelihood computation

It is known that the likelihood computation is one of the most frequently used and the most CPU power demanding parts of the decoder. Like many other authors, we solve the problem by replacing the original equation of log pdf

$$\log(p(\mathbf{x})) = \log[\textstyle\sum_{m=1}^{M} c_m \cdot N(\mathbf{x}, \boldsymbol{\mu}_m, \boldsymbol{\sigma}_m^2)] \qquad (1)$$

by its max approximation

$$\log(p(\mathbf{x})) \approx \max_m(L_m(\mathbf{x})) = \max_m[C_m - (\mathbf{x} - \boldsymbol{\mu}_m)^2 \cdot \mathbf{s}_m] = L_{mbest} \quad (2)$$

Term L_m represents the pdf of mixture *m*. It is computed as the sum of contributions of individual features:

$$L_m = C_m - \sum_{p=1}^{P} [(x_{mp} - \mu_{mp})^2 . s_{mp}] \qquad (3)$$

Smart implementation of eq. (2) and (3) can save much CPU time. The largest savings are already achieved by adopting integer arithmetic. The code for eq. (3) is composed of addition/subtraction and multiplication, properly combined with the scaling of partial results. The scaling is done by applying left and right bit-shifting operations.

The execution time needed for eq. (3) can be significantly reduced by pruning at the mixture evaluation level. The L_m value calculated according to eq. (3) monotonically decreases with increasing *p*. Hence, the summing loop in eq. (2) can be stopped if $L_m(p) < L_{mbest*}$, where L_{mbest*} is the temporary max value achieved within the evaluation of eq. (2).

The efficiency of this pruning scheme can be further improved if mixture *mbest* is evaluated among the first ones. In this case, most evaluations of L_m will stop early. A detailed analysis showed that *mbest* found in frame *f* could serve as a good prediction of *mbest** in frame *f+1* (in the same state). Our implementation uses this enhanced pruning scheme, which reduces the average number of evaluated features (see Table 2) and saves about 40 % of computation time compared to direct evaluation of eq. (2-3). What is important is that this scheme does not introduce any additional loss of accuracy.

Table 2. *Average number of features in evaluation of eq. (3) for different techniques and mixtures numbers*

Technique	**Average number of features for**		
	16 mix	**32 mix**	**64 mix**
Direct evaluation	39	39	39
Evaluation with pruning	24.7	22.0	19.6
Pruning with prediction	21.2	18.8	16.7

3.5. Optimization of decoding procedure

The decoder is optimized for a lexicon that is represented as a tree with shared prefixes. It builds its branched network of nodes (model states) dynamically with respect to the processed speech signal. It operates with integer-coded likelihoods, performing actually only 2 types of operations: summing (accumulation of likelihoods) and comparison (recombination of accumulated likelihoods, pruning, candidate ordering). As most of the code could be re-used from the baseline engine, we just focused on the optimization of the pruning scheme.

The engine uses 2 types of pruning. Both are based on the best overall accumulated score achieved in the *previous frame*. Its value d_{best}, and state s_{best} where it was achieved, are used to determine 2 pruning thresholds for the *current frame f*.

$$pt_1(f) = d_{best}(f-1) - beamwidth \qquad (4)$$

$$pt_2(f) = d_{best}(f-1) + L(s_{best}, f) - beamwidth \qquad (5)$$

The first threshold serves for eliminating calculation in the nodes whose predecessors have scores lower than pt_1. The second threshold is based on an estimate of d_{best} in frame *f*. Its value is predicted from d_{best} (the score in state s_{best}), to which likelihood $L(s_{best}, f)$ in the current frame is added. If a node gets score that is below pt_2, it is pruned off. The value of *beamwidth* controls the decoder's speed, determining how many nodes will survive the pruning. In our system, we set it so that the increase of WER must not be worse than 0.2 % compared to the case when double *beamwidth* is used.

3.6. Optimization of power consumption

When developing programs for MDs, one must consider also the power consumption aspect. In our implementation it is solved by splitting the whole process into multiple threads, each having its priority level and CPU time requirements. The lowest-level thread continuously acquires a signal from the microphone and stores it in a 10-second-long circular buffer. Its CPU load is almost negligible. The second thread performs the FFT and MFCC computation using the emulated floating-point routines. It runs only on demand, being triggered by a voice activity detector. It requires about 40 % of CPU power. After the end-point of speech is reached and confirmed, the third thread gets the sequence of feature vectors, applies the CMS normalization and performs the decoding procedure. This thread utilizes the full power of the CPU.

3.7. Final implementation and user interface

The recognizer is written in C language and has form of a DLL. It communicates with the user through interface whose simple and intuitive design is shown in Fig. 1. Its largest area is occupied by a text box. When a new word is recognized, the candidate with the best score is added to the current text position. Up to 5 next candidates are displayed below. The user can select from them (using the touch screen) when he/she wants to correct an error or disambiguate a homophone. The two other buttons allow for deleting a word and for toggling upper/lower case of the word's first letter. The text can be stored to the MD's disk memory or sent as a short message via the mobile phone. The application also allows the user to add new words into the lexicon and to run a brief speaker adaptation session. In the latter case, the adaptation routine is performed on an attached PC. The program has been successfully tested on several devices (e.g. HP iPAQ214, Samsung Omnia i900, or HTC Touch HD) - see video [8].

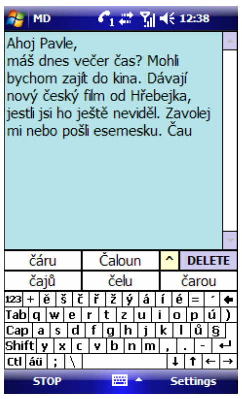
Ahoj Pavle,
máš dnes večer čas? Mohli bychom zajít do kina. Dávají nový český film od Hřebejka, jestli jsi ho ještě neviděl. Zavolej mi nebo pošli esemesku. Čau
MD
12:38
čáru
Čaloun
^
DELETE
čajů
čelu
čarou
123 + ě š č ř ž ý á í é = ´
Tab q w e r t z u i o p ú)
Cap a s d f g h j k l ů §
Shift y x c v b n m , . -
Ctl áü ; \
STOP
Settings

Fig. 1. *User interface of the dictation program*

3.8. Experimental evaluation and comparison

For performance evaluation we recorded a new test set. It contains newspaper text dictated by 24 speakers (13 male and 11 female), 5660 words in total, recorded on a PDA (HP iPAQ214) and a mobile phone (Samsung Omnia i900). On this data set, several types of experiments were conducted. In the first series we measured the WER of the baseline engine running on a PC (with full floating point implementation). After that we evaluated the performance of the MD engine optimized by employing the techniques described in sections 3.2 to 3.7. Again, we quantified WER (for the best candidate and also for the first 6 ones, as these are available to the user). Moreover, we measured the average time consumed by the decoder and also the total delay elapsed from the utterance endpoint to the moment when recognition is completed. In the latter case, a 155-ms latency caused by the endpoint detector must be added. (All the times were measured on the Omnia device, 624 MHz.) The results are summarized in Table 3.

The constraint that the recognition delay should not be longer than 0.5 s is fulfilled in case when the MD uses 32 mixtures. Then, the corresponding WER is 17.5 %, but at the same time, there is only 3.2 % chance that the correct word is not among the candidates the user can select from. If we compare 32-mixture results achieved on the PC (with floating point support) and on the MD, we see only 0.3 % difference in WER, caused by all the optimization techniques. On the last line we show also the results achieved after all the test persons passed a short speaker adaptation session (270 words).

Table 3. *Performance of several versions of dictation program (PC with FPU vs. MD, 255K lexicon)*

	WER [%]		**Average delay [ms]**	
Engine version	**1 best**	**6 best**	**decoder**	**total**
PC – FPU, 96 mix	14.3	2.0	NA	NA
PC – FPU, 32 mix	17.2	2.7	NA	NA
MD – 16 mixtures	19.9	4.1	211	365
MD – 32 mixtures	**17.5**	**3.2**	**341**	**495**
MD – 64 mixtures	16.7	2.8	604	759
MD – 32 mix, adapt.	11.5	1.8	306	461

4. Server based dictation program

As an alternative to the above described fully embedded application we developed also a prototype of a fluent speech dictation program with distributed speech recognition. The client is a rather small program that acquires a signal from the microphone and compresses it from its original 256 kbit/s rate to 16 kbits/s (using the Speex codec [11]). The compressed signal is sent via WiFi or GPRS to the server. On the server side, we run a slightly modified version of our NewtonDictate program [9] equipped with a 370K-word lexicon. The reason why we transmit the speech signal (not its preprocessed features) is that we want to utilize all the benefits that the dictation program offers, namely the option to replay and check the recorded and transcribed data (stored on the server). The recognized text is sent back to the MD and displayed on the interface in the way similar to the previous program. Its function can be seen in videos [12] or [13]. The latter demonstrates that by employing the Google translate facility [14] we can get a simple speech-to-L2text translation.

We have also conducted a series of experiments. The most interesting one was to compare the direct dictation into a PC with the dictation via a MD. Two persons were asked to dictate newspaper texts (7789 words in total) into a close-talk microphone attached to the PC, and at the same time, also into a MD held in a hand and connected by WiFi to the recognition server. The averaged WER value for the former situation was 8.2 % while for the latter it was 17.9 %, i.e. significantly worse. We found out that only 1 % of this accuracy loss was due to the applied signal compression. The remaining and larger portion was due to the lower quality of a signal provided by mobile devices. This demonstrates another limitation that must be taken into account when developing voice applications for recently available mobile devices.

5. Discussion and conclusions

In this paper we propose two solutions to the problem of voice dictation into modern mobile devices. Both are focused on the tasks and the languages where large vocabularies are required.

The first (embedded) program has the advantage that it can be used anywhere without the need to be connected to (and pay for) the wireless network. Its drawback consists in the constraint that the user must dictate the text with short pauses between words. The relatively high WER (about 17 % in the speaker independent mode) is caused partly by the lower speech signal quality (compared to a PC with a close-talk microphone).

The second, client-server based solution allows the user to dictate in a natural fluent way, but the overall results are not significantly better compared to the standalone version – the main reason being the lower signal quality.

From the user's point of view, the discrete dictation system offers an immediate and more comfortable method for error correction (or disambiguation). When compared to the classic T9-supported stylus typing, the program saves ~ 30 % time. The recognition accuracy, and hence also the dictation speed and comfort, can be further improved by speaker adaptation, which takes about 10 minutes and reduces the WER to some 10 %.

6. Acknowledgements

This work was supported by the Grant Agency of the Czech Republic (grants no. 102/08/0707 and 102/07/P430) and by the Grant Agency of the Czech Academy of Sciences (grant no. 1QS108040569).

7. References

[1] Basapur, S., Xu, S., Ahlenius, M., and Lee, Y. S., "User Expectations from Dictation on Mobile Devices", J. Jacko (Ed.): Human-Computer Interaction, part II, LNCS 4551. Springer-Verlag Berlin Heidelberg, pp. 217-225.

[2] Cohen, J., "Embedded speech recognition applications in mobile phones: status, trends and challenges", Proc. of ICASSP 2008, Las Vegas, pp. 5352-5355.

[3] Rose, R. C., Arizmedi, I.: Efficient client-server based implementations of mobile speech recognition services. Speech communication, vol. 48 (2006), no. 11, pp. 1573-1589.

[4] Kohler, T.W., Fugen, C, Stuker, S and A. Waibel, Rapid porting of ASR-systems to mobile devices," Proc. of Interspeech 2005, Lisbon, pp. 233–236.

[5] Huggins-Daines, D. et al, "PocketSphinx: A Free, Real-Time Continuous Speech Recognition System for Hand-Held Devices", Proc. of ICASSP 2006, pp. 185-188.

[6] Olsen J., Cao Y., Ding G., Yang X., "A Decoder for Large Vocabulary Continuous Short Message Dictation On Embedded Devices", Proc. of ICASSP 2008, 4337-4340, 2008.

[7] Alhonen J., Cao Y., Ding G., Liu Y., Olsen J., Wang X., Yang X., "Mandarin Short Message Dictation on Symbian Series 60 Mobile Phones," The Int. Conf. on Mobile Technology, Applications and Systems, Singapore, 2007.

[8] http://www.youtube.com/watch?v=T0oFoPfIpgI

[9] Nouza J., Zdansky J, Cerva P, Silovsky J: Challenges in Speech Processing of Slavic Languages (Case Studies in Speech Recognition of Czech and Slovak). To appear in Springer LNAI Series, 2009.

[10] Cerva, P., Nouza J.: Design and Development of Voice Controlled Aids for Motor-Handicapped Persons. Proc. of Interspeech, Antwerp, 2007, pp. 2521-2524

[11] http://speex.org/

[12] http://www.youtube.com/watch?v=6-7R7Vsdewc

[13] http://www.youtube.com/watch?v=jO-4tySAUP4

[14] http://translate.google.com/