Fast Script Word Recognition with Very Large Vocabulary

Marc-Peter Schambach
Siemens AG, Logistics and Assembly Systems, Postal Automation
78459 Konstanz, Germany
Marc-Peter.Schambach@siemens.com

Abstract

For an HMM-based script word recognition system an algorithm for fast processing of large lexica is presented. It consists of two steps: First, a lexicon-free recognition is performed, followed by a tree search on the intermediate results of the first step, the trellis of probabilities. Thus, the computational effort for recognition itself can be reduced in the first step, while preserving recognition accuracy by the use of detailed information in the second step. A speedup factor of up to $15\times$ *could be obtained compared to traditional tree recognition, making script word recognition with large lexica available to time-critical tasks like in postal automation. There, lexica with e.g. all city or street names (20-500k) have to be processed within a few milliseconds.*

1 Introduction

Recognition of cursive script has made considerable progress over the last years. State-of-the-art word recognition engines yield recognition rates that make it usable in industrial applications like postal automation. But, in that application area, another crucial requirement is recognition speed. For postal applications, recognition tasks with large lexica of sizes up to 500k have to be recognized in 50 to 100 milliseconds. Those tasks arise, for example, in the recognition of addresses written in cursive script, when only view context is available because e.g. the postal code is missing and all city hypotheses have to be checked.

The word recognition system discussed here is based on character level left-to-right HMMs. It has been presented first in [2, 3] and is described in full detail in [5]. Operating on lexica of size 100, it achieves recognition rates around 94% (see section 4). By default, recognition is performed on a character trie, merging word prefixes, and exact likelihoods are computed for each lexicon word by the Viterbi algorithm. While reaching maximum recognition performance, recognition times on bigger lexica (size 22000) goes up to 180 ms (standard PC with 2.8 GHz).

One popular solution to the challenge of large vocabulary is the usage of character N-grams. In [1], a system has been presented that yields good results, especially in out-of-vocabulary situations. But, for the given application, checks against given lexica have to be performed finally, limiting the utility of the N-gram method. Furthermore, implementations of N-gram recognition as presented in [7] tend to be time-consuming. Own experiments achieved runtimes exceeding the constraints for practical applications.

The solution presented here is a two-step Viterbi decoding, performing context-free recognition in the first step, building the base for the probability estimation of the lexicon words in the second step. This results in a significant speed-up, by the cost of some loss in recognition accuracy.

The paper is organized as follows. In section 2, the proposed algorithm is presented in detail. Section 3 discusses the approximations made and sketches a solution for more accuracy. Section 4 describes some experiments and results, comparing standard trie recognition with the proposed method. Consequences of these results are discussed in section 5, before a summary and outlook is given in section 6.

2 The algorithm

The proposed algorithm performs two recognition steps (see Fig. 1):

- In the first (forward) step, a lexicon-free recognition is performed. This is done by recognition on an automaton that generates any possible character sequence. This step can be performed fast, because the automaton is small compared to a lexicon trie.
- In the second (backward) step, a breadth-first search on a trie representing the lexicon is performed. It uses the results from the first step to estimate the likelihoods for each path.

The advantage of this method is to keep the complexity of HMM calculation low in the first step, with only a small automaton; later on, in the second step with a huge lexicon

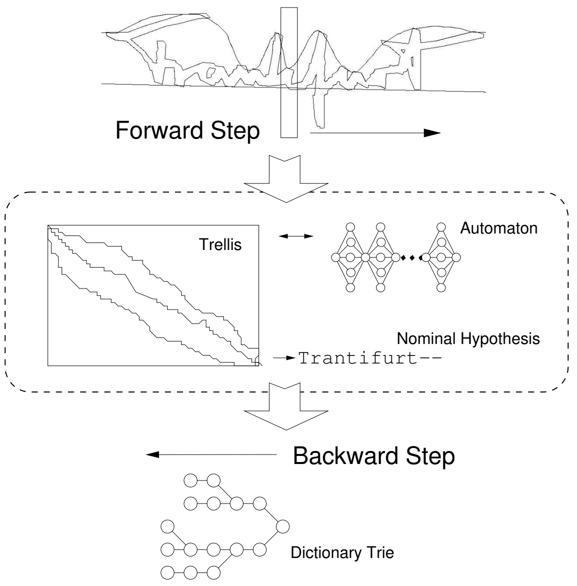

Figure 1. Algorithm: The first step creates a likelihood trellis, used by the second step

trie, one is able to use the precomputed likelihood values of the first step.

2.1 Forward step

In the forward step, HMM calculation is performed without detailed context knowledge. Context is only provided in terms of alphabet and of the length of the words to be recognized. It is defined as an automaton, which generates a superset of all possible words with variable but limited number of characters. If specified in terms of regular expressions, German postal codes (5 digits) are defined by `[0-9]`{`5`} (see Fig. 2), while German city names (up to 20 characters) can be defined by `[a-Z]`{`1-20`}. In the case of fixed length, only the last node is an accepting state of the automaton, while with variable length all intermediate nodes are accepting states as well.

Compared to a trie representation, there are much fewer nodes, keeping the number of calculations for emission and transition probabilities small. Each node has multiple predecessors. To reduce the number of transitions between nodes, intermediate nodes are introduced; this corresponds to the transformation into an edge-based graph representation. Time-consuming maximizations of likelihood during Viterbi calculation can be reduced this way. A minor drawback of the intermediate nodes is given by the fact that the complexity of the automaton can not be further reduced by the use of positional character N-grams of lexicon words.

The primary result of Viterbi forward calculation is the likelihood of the best-fitting sequence of characters (HMM question 1 in the Rabiner paper [4]). But to extract this sequence itself ([4], question 2), the intermediate likelihood values during Viterbi forward calculations have to be stored and used in a backtracking step. These values build the *trellis* and contain all information about the recognition process. For each HMM state at each frame the likelihood and thus implicitly the best predecessor states are stored. They are of interest especially at the boundaries of the character nodes.

2.2 Backward step

The single best result of context-free recognition, the *nominal hypothesis*, can be found in a backward step, traversing back the trellis to each best predecessor state. This hypothesis represents in the case of readable words mostly a plausible result, but in the majority of cases it is not a word from the lexicon. Therefore the backtracking step is adapted to extract the n best likelihoods of words which are contained in the lexicon.

To facilitate this, a trie is built from the lexicon which merges the word endings (see Fig. 3). Separate branches are created for each word length, because different word lengths correspond to different accepting states in the automaton the trellis has been build for. In the case of constant lexica the construction of this trie can be done offline, saving online computation time.

A breadth-first search is performed on this trie to determine the probabilities of all lexicon words. Trie and trellis are processed from right to left. For each node, two steps are conducted:

- Compute the loss in log-likelihood Δp which results in choosing this node compared to the best node. The current frame position is given by the right side's node and has been already calculated. Best and actual likelihoods are given by the trellis: $\Delta p = p_{\text{best}} - p_{\text{node}}$. The total loss in log-likelihood for the current branch

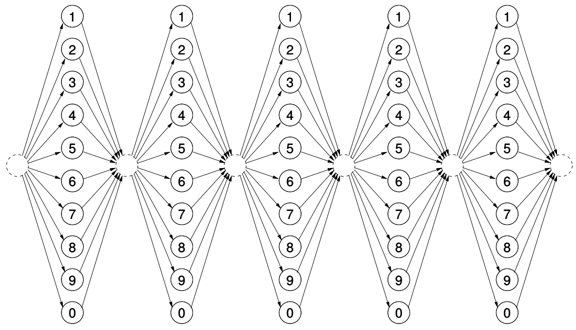

Figure 2. Automaton for forward recognition of German postal codes

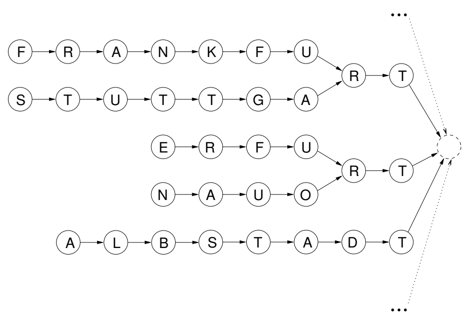

Figure 3. Trie for backward result extraction.

is summed up and stored for the active node.

- Determine the best entry frame from the trellis and store it for use by the next preceding nodes.

Thus for each end node of the trie a total loss of likelihood is summed up, resulting in an individual likelihood for each lexicon word. The lexicon words are sorted by this value. Thus n best results are computed. The computation is performed very efficiently, because only trellis values have to be indexed and summed up.

2.3 Pruning

Pruning is performed on both the forward and backward steps. In both cases the aim is to keep the number of nodes for calculation small. The pruning factors strongly influence recognition rate and speed and have to be chosen judiciously.

In case of the forward step, the maximum likelihood p_{max} is determined at each frame. Given a pruning factor f_{forw}, a minimum likelihood for all states is defined by $p_{\text{min}} = p_{\text{max}}/f_{\text{forw}}$. Only those character nodes containing at least one state more likely than p_{min} will be included in the calculations for the next frames; all other branches are *pruned*.

At the backward step, the number of active nodes is kept small in a similar way. Pruning action is taken now during breadth-first search at each step down the tree depth. Again, the maximum likelihood is determined, and all trie-branches below p_{min} (defined by a factor f_{back}) are pruned. Useful values for the pruning factors hold $f_{\text{back}} <= f_{\text{forw}}$, in order to prevent the calculation of branches already pruned in the first step.

Another speedup is achieved by the estimation of word length before recognition in order to keep the number of nodes at a minimum. The estimation is simply based on the number of frames and the width-to-height-ratio of the input image.

3 Accuracy of the algorithm

3.1 Approximations

A drawback of the proposed algorithm is that not for all lexical words the single best segmentation is calculated. This results from the way the segmentation points are determined:

- In traditional, exact recognition, all segmentation points of a given word are chosen to maximize the likelihood of this word.
- In the approach presented here, each segmentation point of a given word maximizes the likelihood of a word model, consisting of arbitrary characters on its left side and the word's character sequence on its right side.

For example, the segmentation points for the word "Berlin", from left to right, are optimal for the word models "*erlin", "**rlin", "***lin", "****in" and "*****n" (stars "*" indicating sequences of arbitrary characters). This could result in a segmentation that is not optimal for the word "Berlin" itself. Another, artificial example may illustrate this. The locally best segmentation of two words "Iiiitown" and "Wwwwtown" may be different (in case "town" is not easy readable), but the algorithm would enforce a common segmentation for the letters "town".

An example is shown in Fig. 4. The exact calculation gives a plausible segmentation, while the approximation calculates some segments differently (and wrong). However, in the example shown, this segmentation is still good enough for correct recognition.

3.2 Exact estimation

To overcome the mentioned approximation problem, a 2-step algorithm has been implemented. After recognition with the proposed algorithm, a traditional, trie based recognition is performed subsequently on the n-best results of

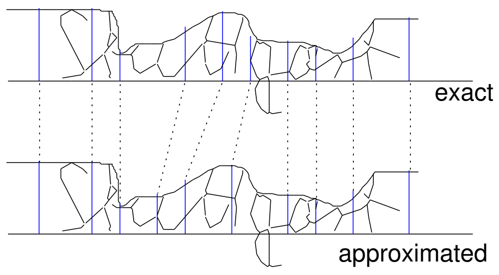

Figure 4. Segmentation differences

the first step. This way, exact likelihood estimations can be given for all results.

This procedure results in a re-ordering of the results, thus improving recognition accuracy. The time penalty of the second step is low, because tries are small and preprocessing results can be cached.

4 Experiments and results

Experiments have been conducted to evaluate recognition time and accuracy. Recognition performance achieved by the presented algorithm are compared to those obtained by a system performing exact recognition on a trie. This system has been in practical use for years, operating typically on lexica of sizes 10 to 1000. Additionally, results of nominal (lexicon-free) recognition are presented.

4.1 Interface

The input to word recognition consists typically of a word image and a *lexicon*. For the proposed algorithm, a lexical *pattern* for the first recognition step is used additionally. The following recognition modes can be chosen by providing different types of input:

- Lexicon only: Exact recognition on a trie.
- Pattern and lexicon: The proposed 2-step algorithm; subsequent exact recognition is optional and triggered by a corresponding switch.
- Pattern only: Nominal, lexicon-free recognition, providing one best result alternative.

For each recognition mode, results are presented.

4.2 Test data

Three data sets from the postal automation domain have been tested. The word images are taken from scanned German addresses (cursive script style), extracted semi-automatically.

- 580 postal code images (5 digits; lexicon contains 27540 valid codes)
- 1047 city names (lexicon with 21757 valid names, including common abbreviations)
- 1007 street names (lexicon with 478123 valid names, including common abbreviations)

Examples of test images are shown in Fig. 5. Lexica are collected from standard databases. All image labels are contained in the lexica, i.e. no out-of-vocabulary situations have been tested.

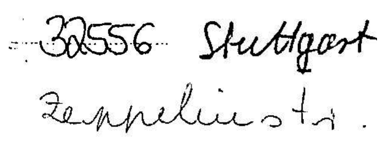

Figure 5. Examples of test images

4.3 Results

Results are shown in Table 1. All experiments have been conducted with identical configurations; recognition rates for smaller lexica of size 100 and 1000 are shown to characterize the recognition engine in standard conditions. Computations have been performed on a standard PC with 2.8 GHz Pentium 4 processor and 1 GByte memory. For each algorithm, recognition rate and computation time are shown.

Each result is strongly dependent on pruning factors. Pruning factors have been adjusted semi-automatically, aimed at maximum recognition performance. A decrease of 0.2% in recognition rate has been accepted if processing time has dropped significantly.

The first row shows the traditional system. Its recognition rates build the reference on which the new algorithm has been evaluated. The low recognition rates (around 60%) are accounted by the large lexica, providing many similar words, which are counted as errors even in case of only small misspellings. While computation time may be acceptable for postal codes and cities, it forbids real applications in the case of street names.

The second row shows recognition results on the pattern only. High recognition rates have not been expected, but the obtained results give a good impression of the difficulty of the task and the potential of the word recognition engine itself. The good recognition results for postal codes are due to the relative simplicity of the segmentation task and the fact that the pattern corresponds to 10^5 valid alternatives only. The computation time corresponds to the first step of the combined algorithm.

Results of the proposed algorithm are shown in the third row. Computational times are reduced drastically compared to the exact approach, but recognition rates are also lower. Reasons are partly given by the approximations that have been discussed in section 3.1. These problems are handled by the exact preprocessing which is evaluated in the 4th row. This approach improves recognition performance by up to 5%, adding only about 10% in computation time. It always performs better than without post-processing; for postal codes it even improves the one-step exact calculation. The overall speed-up factor compared to exact calculation is up to $15.8\times$, by an average of $8.6\times$.

Table 1. Recognition results and computation times of the various algorithms

	Postal codes	City names	Street names
Images	580	1047	1007
Lexicon = 100	93.6 %	93.6 %	94.9 %
Lexicon = 1000	88.1 %	87.9 %	92.2 %
Lexicon size	27540	21757	478123
Pattern	`[1-9]{5}`	`[a-Z]{1,15}`	`[a-Z]{1,25}`
1 Exact recognition (trie)			
Recognition	54.1 %	63.1 %	59.1 %
Time	37.8 ms	178.3 ms	2288 ms
2 Nominal recognition (pattern)			
Recognition	44.1 %	4.3 %	2.5 %
Time	3.5 ms	18.7 ms	26.4 ms
3 Approximation (pattern & trie)			
Recognition	53.8 %	55.1 %	48.1 %
Time	4.3 ms	44.2 ms	135.7 ms
4 Approximation & exact post-processing			
Recognition	54.2 %	60.0 %	51.4 %
Time	5.7 ms	52.0 ms	145.2 ms
Accuracy	100 %	95.1 %	87.0 %
Speed-up	6.6×	3.4×	15.8×

4.4 Discussion

For a reasonable evaluation of the presented results it has to be noted that with lexica as big as used here, the definition of recognition and error rates become questionable. This is mainly due to the increased similarities between alternatives. See [6] for a detailed discussion of this fact. There, definitions of word similarities and overall recognition quality have been given. These measures can be estimated with high accuracy by the results of the forward and backward recognition step, respectively.

Furthermore, the correct word has always been included to the test lexicon, thus no reject criteria have been tested systematically yet. Nevertheless, the experiments give a good overall impression of the performance and usability of the proposed algorithm. Additional experiments show that the calculated measures can be used as reject criteria superior to the results of exact trie recognition.

5 Conclusions

The main conclusion of the experiments is that cursive script word recognition is applicable even when no further context knowledge is given and huge lexica (e.g. all street names) have to be used. In these cases, high speed is more important for practical applications than high recognition performance. Various n-best results on a document (e.g. address) may later be correlated on higher levels (e.g. address level) to yield good overall recognition rates.

The experiments show that the usability of the proposed algorithm is reduced in cases when the word images get too wide, as it is the case with street name. These consist quite often of multiple words. In the case of large images, segmentation errors (as discussed in section 3.1) tend to limit the possible recognition performance. A solution to this problem would consist of a multi-step approach with word level segmentation, followed by multiple word recognition calls.

6 Summary and outlook

In this paper, an algorithm has been proposed which allows a fast recognition of very large lexica in an HMM based word recognition system. Besides approximations that are made, recognition rates are comparable to those achieved by a system performing exact recognition. A significant speed-up of factor $8.6\times$ in average could be obtained. This makes the algorithm useful in applications with hard time constraints. Next, the algorithm will be included in an address recognition system to operate in situations where no initial context knowledge is available to restrict lexica. Significant improvements for addresses written in cursive script are expected.

References

[1] A. Brakensiek and G. Rigoll. A comparison of character N-grams and dictionaries used for script recognition. In *Proc. of the 6th ICDAR*, pages 241–245, Seattle, WA, Sept. 2001.

[2] T. Caesar, J. Gloger, and E. Mandler. Preprocessing and feature extraction for a handwriting recognition system. In *Proc. of the 2nd ICDAR*, pages 408–411, Tsukuba Science City, Japan, Oct. 1993. IEEE Computer Society Press.

[3] A. Kaltenmeier, T. Caesar, J. Gloger, and E. Mandler. Sophisticated topology of hidden Markov models for cursive script recognition. In *Proc. of the 2nd ICDAR*, pages 139–142, Tsukuba Science City, Japan, Oct. 1993.

[4] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, Feb. 1989.

[5] M.-P. Schambach. *Automatische Modellierung gebundener Handschrift in einem HMM-basierten Erkennungssystem.* Dissertation, Universität Ulm, 2004.

[6] M.-P. Schambach. A new view of the output from word recognition. In *9th IWFHR*, Tokyo, Japan, Oct. 2004.

[7] D. Willett, C. Neukirchen, and G. Rigoll. DUcoder - The Duisburg University LVCSR stackdecoder. In *Proc. of the ICASSP*, Istanbul, 2000.