Fast Lexicon-Based Word Recognition in Noisy Index Card Images

Simon. M. Lucas*, Gregory Patoulas** and Andy C. Downton**
* Department of Computer Science
** Department of Electronic Systems Engineering
University of Essex, Colchester CO4 3SQ, UK

Abstract

This paper describes a complete system for reading typewritten lexicon words in noisy images - in this case museum index cards. The system is conceptually simple, and straightforward to implement. It involves three stages of processing. The first stage extracts row-regions from the image, where each row is a hypothesized line of text. The next stage scans an OCR classifier over each row image, creating a character hypothesis graph in the process. This graph is then searched using a priority-queue based algorithm for the best matches with a set of words (lexicon). Performance evaluation on a set of museum archive cards indicates competitive accuracy and also reasonable throughput. The priority queue algorithm is over two hundred times faster than using flat dynamic programming on these graphs.

1 Introduction

Given the accuracy of current commercial off-the-shelf OCR packages, one might assume that reading the machine print in poor quality images, such as those scanned from museum archive cards, would be a solved problem. This, however, is not the case. Most commercial systems still seem to be too brittle to handle difficult cases, though the state of the art continues to improve.

On the other hand, there are hand-writing recognition systems[1, 3, 9, 2] that are designed from the outset to cope with hard-to-segment images, and like the method we propose here are based heavily on graph-search algorithms. The method we propose here is an extreme version of a graph-based search method, and improves on the one we reported recently [8] in three ways: we now use an optimal binarization method for our particular classifier, we don't do any prior word extraction (which means searching a much larger graph instead), and we use a priority-queue based search which significantly improves performance. Our system performs extreme over-segmentation in the sense that it applies the classifier at all possible positions in the image (in this case, an extracted row image). Most other systems consider only a limited number of segmentation points. The word-match computation is ultimately very similar to the top-down method of Ishidera *et al* [4], except that the approach described here is significantly more efficient. The Ishidera method has proven to be very accurate for the kinds of index card images considered in this paper, but for efficiency reasons is only sensibly applied to isolated word images. We expect the method described here to be similarly accurate, while possessing a big speed advantage.

In this paper we apply our system to reading the year field on a set of museum archive cards - the same archive as discussed in [8].

2 System Architecture

The system architecture is depicted in Figure 1. There are three stages. First, a standard histogram-based algorithm is used to extract row images. Each row image corresponds to a possible line of text. Next, the character classifier is applied to all possible positions in each row-image, to build a character hypothesis graph for each row. Finally, the graph for each row is searched to give a best-first list of all the lexicon words in that row.

Ideally, we would prefer to adopt an even simpler two stage approach, where the row extraction phase is eliminated (since it is a significant source of error and it also increases the complexity of the system). This would make the graph larger and more complex to search, however, since each hypothesis would be indexed by its x, y position, instead of its x position alone.

The character hypothesis graph is created by applying our character classifier at all possible positions within each row. The recognition confidence for a given character at a specified x-position (node) in the graph is taken to be the maximum for that character over all y positions in that column. The graph is not explicitly created, but built implicitly during the search procedure given below. Arcs are formed on demand between a node at position x in the graph to all nodes within a range of possible character offsets from x. An arc is added for each possible offset, and for each character hypothesis at position x that exceeds some confidence

Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR 2003)
0-7695-1960-1/03 $17.00 © 2003 **IEEE**

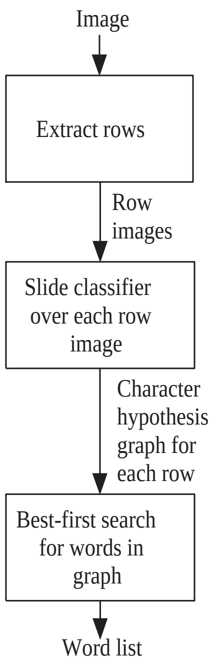


Image
Extract rows
Row images
Slide classifier over each row image
Character hypothesis graph for each row
Best-first search for words in graph
Word list

Figure 1. The system architecture.

threshold (or other criteria).

3 The OCR Algorithm: Optimal Threshold Template Matching

For our OCR method we currently use a nearest neighbour template matching algorithm. Before applying this kind of recognizer it is usual to binarize the colour or grey-level image in order to discard irrelevant variations in intensity. Image binarization is still an important issue. Fixed global binarization does not work well for these card images due to significant variations in background and foreground intensity across the cards. Niblack's method works well [10] - and the method of Seeger and Dance [11] would also be interesting to try, since it has been developed especially for document images, but is more complex than the method we propose here. Our method finds the optimal threshold for matching each template, can be computed efficiently (perhaps not *quite* as efficiently as the other methods), and has no parameters to set (unlike Niblack's method, for example, which has a threshold factor to adjust).

We define a template T to possess a set of foreground pixel locations T^f, and a set of background pixels locations T^b. The grey level image to match we denote as I, with n possible grey levels.

We define the grey-level histograms for the foreground and background sets:

$$h^f = \text{hist}(I, T^f)$$

and

$$h^b = \text{hist}(I, T^b)$$

The *hist* function creates a histogram as follows. It first creates the histogram array of size *n*, with all entries initialized to zero. It then loops over all pixel locations specified in its second argument, and increments the count for the pixel intensity found at that location in image I. The cost of producing these two histograms is proportional to the number of template pixels: $|T^f \cup T^b|$. We denote the ith entry of a histogram h as h_i i.e. the number of pixels found in the corresponding set with a grey-level of i.

The optimal match (and the threshold at which that match occurs) can now be found by iterating over the number of entries in the histogram (which is equal to the number of grey levels in the image).

The match m_i at each threshold point i may be calculated recursively as follows:

$$m_0 = h_0^f - h_0^b$$

$$m_i = m_{i-1} + (h_i^f - h_i^b) \text{ where } 0 < i < n$$

We define the un-normalized match m' as the maximum of m_i over all possible i:

$$m' = \max(m_i) \text{ where } 0 < i < n$$

This is un-normalized since we've only been subtracting the background pixels. To normalize it we add the number of background pixels, and divide by the total number of template pixels. Hence, the normalized match m is defined as:

$$m = \frac{m' + |T^b|}{|T^f| + |T^b|}$$

This then gives us a match quality in the range zero to one. For practical purposes, the worst possible match value is 0.5, which indicates a fifty per cent overlap between template foreground and image threshold foreground. A match of one indicates a perfect match between the template and the thresholded image. A match of zero indicates a perfect match between the template and the thresholded image negative. In this application, however, we just used the match value m in the range zero to one as given by the above equation, since all our type-written images are dark on a light background.

4 Efficient lexically-constrained graph search

To test the viability of the approach, we initially implemented a dynamic programming algorithm to find the optimal alignment between each lexicon word and the recognition hypothesis graph. This is often referred to as *flat* DP since no hierarchical data structures are involved. We emphasize that in this method, the computation is performed separately for each word in the lexicon. This is clearly wasteful, since it ignores the fact that many words share common substrings. This is the method described in Lucas *et al* [8].

The flat DP method described above gave very accurate results, but took about 5 seconds to search each row-graph for the possible range of 300 dates. This is already slow, but for large dictionaries would be worse - since the cost of flat DP is linear with respect to dictionary size.

To alleviate this we designed a more efficient algorithm which compiles the dictionary into a prefix tree and uses a priority queue to perform a best-first search of the recognition graph. This resulted in a huge speed-up - with most searches now taking around 20ms. Hence, the approach now becomes more practical.

The algorithm is a type of Stack Decoding algorithm [5, 6] that has been developed to efficiently search a lexicon of possible words given the character hypotheses in the recognition graph. Before listing the algorithm in pseudo-code, we describe the other data structures and object classes used.

A *CharHypothesis* is an element in the recognition graph that gives its x-position, its character class and its confidence. We treat these confidences here as independent probabilities, and compute the likelihood of a path as the product of its individual character confidence values.

The *PriorityQueue* is a standard data structure that offers ordered access to a set of objects, and is based on a binary heap. It provides three methods: hasMore() returns true if the queue is not empty, false if it is; add() adds an object to the queue; pop() returns and removes the first item from the queue. Note that all items in the queue must be comparable with each other. The queue is maintained in order of highest confidence objects (i.e. paths) first.

The *trie* is a data structure that stores all lexicon words in a prefix tree [7]. Ours is implemented using a Hash table indexed on the next character to store the references to the next nodes in the trie. Each node has a boolean flag to indicate whether this node represents a complete word. Each node also holds a reference to the previous node in order to retrieve the word represented by that node by performing a back-trace to the root of the trie. Our trie has two methods of relevance to the *wordSearch* function: follow(char c) returns the node reached from the current node by following character *c*. The function next() returns the set of all characters that can follow the current node in the trie.

The *Path* class stores the node in the trie reached by this path, the column in the graph (x-position) that this path ends at, and the confidence of this path. *Path* implements the *Comparable* interface such that more likely paths are pushed to the front of the priority queue.

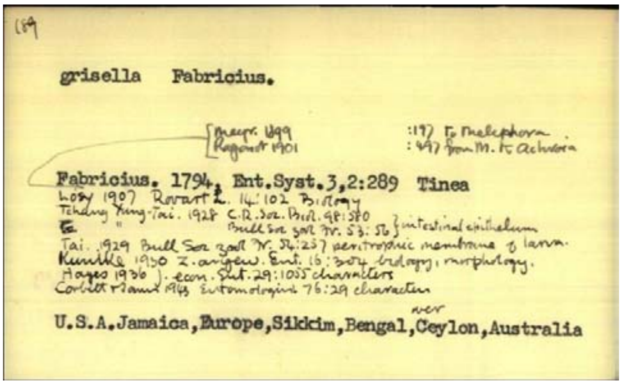
The PathLut class offers efficient access to the best paths to each node (column) in the graph. This is implemented as an array of hashtables. The hashtable for each column contains a set of paths, indexed on the trie node reached by the end of the path. This allows us to determine very efficiently whether a newly extended path should be extended any further. This brings us to the *extensionsOf* function. This takes a path, the recognition graph and the pathLut. This function gets the set of possible extensions of the trie node of the path, and considers extending each one. First the confidence of that character in that position is looked up in the recognition graph. We allow some spatial variation at this stage, by looking for characters in the recognition graph that lie in the inclusive range of gap = minGap to maxGap from the current end point of the path. Based on manual observation of the cards, and some tuning of the algorithm, we use a minGap of 10 pixels and a maxGap of fifteen pixels. The decision is made as follows. If pathLut[path.x + gap] already holds a better (more likely) way of reaching that trie node at that column in the graph, then there is no need to extend it further - otherwise it should be. It would also be possible to prune paths that fall below some likelihood threshold, but we don't do this yet.

```
PriorityQueue wordSearch(
  RecGraph recGraph ,
  Trie words,
  int n )
{
  // Declare variables
  PriorityQueue activeQueue;
  PriorityQueue completed;
  PathLut pathLut;

  // initialise
  activeQueue = new PriorityQueue();
  completed = new PriorityQueue();

  pathLut = new PathLut( recGraph.width );

  for each CharHypothesis ch
      in recGraph {
    Path path = new Path(
      words.follow( ch.char ),
      ch.x , ch.p);
    activeQueue.add( path );
  }

  // do the search
  while (activeQueue.hasMore() AND
         completed.size() < n  )
  {
    Path curPath = activeQueue.pop();
    Set extensions =
      extensionsOf(
        curPath , recGraph, pathLut );
    for each Path extended in extensions {
      if (extended is complete) {
        completed.add( extended );
      }
      activeQueue.add( extended );
    }
  }
  return completed;
}
```

5 Results

We evaluated the system on a batch of museum index cards. One of the main concerns with our previous method of reading the words on these cards was in extracting the year field of the first reference on these cards, so we chose this particular task for the initial evaluation.

An example card is shown in Figure 2.



189

grisella Fabricius.

[Meyr. 1899 :197 to Melophora
[Rogonot 1901 :497 from M. to Achroia

Fabricius. 1794, Ent.Syst.3,2:289 Tinea
Losy 1907 Rovart L. 14:102 Biology
Tchang Yung-Tai. 1928 C.R.Soc.Biol. 98:580
" Bull Soc zool Fr. 53:56 } intestinal epithelium
Tai. 1929 Bull Soc zool Fr. 54:257 peritrophic membrane of larva.
Kuntke 1930 Z. angew. Ent. 16:304 biology, morphology.
Hayes 1936 J. econ. Ent. 29:1055 characters
Corbett & Pound 1943 Entomologist 76:29 characters

ver
U.S.A.Jamaica,Europe,Sikkim,Bengal,Ceylon,Australia

Figure 2. Example index card image.

The year we wish to extract from this image is “1794” in the row beginning “Fabricius.”. The row extraction algorithm extracts fourteen rows for this image - many of them giving no match for any year. The image extracted for this row together with the first few retrievals is shown in Figure 3. The system returns XML data for the results, organized into rows. An abridged version of the XML output for this image in Figure 2 is shown below:

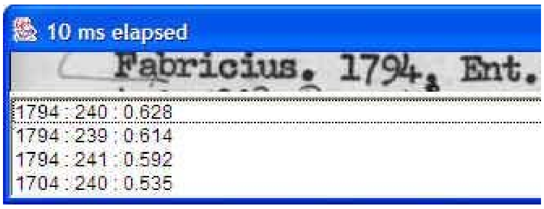
```
<Result path="images/Fabricius.jpg" >
    <Row y="11" />
    <Row y="73">
        <Word w="1801" x="129" p="0.436" />
        <Word w="1881" x="129" p="0.432" />
        <Word w="1891" x="129" p="0.432" />
        ...
    </Row>
    <Row y="150" />
    <Row y="201">
        <Word w="1794" x="240" p="0.628" />
        <Word w="1794" x="239" p="0.614" />
        <Word w="1794" x="241" p="0.592" />
        ...
    </Row>
    <Row y="229" />
    ...
```

For comparison, a leading commercial system returned the following for that row:

```
Pabrioiua., 17%, Ent.Syst.3,2:289 Tinea
```

- though we were unable to configure the system to use our highly restricted lexicon of just 300 possible years, and while we set the commercial system to ‘TypeWriter’, our system had the benefit of templates that were based on samples from museum cards similar to these ones.

We used this set of 815 images in another experiment where an alternative method of ours was under test, based on using various layout rules to extract the year field. This layout-based system only successfully read the year field in about 20% of cases in this batch of cards, owing to the difficulty in extracting the year part of the row. The system reported here improves significantly on that. The results are shown in Table 1. In the table *Correct(1)* shows the number of words correct in the first position, *Correct(2-11)* in positions 2 through 11 in the n-best list, and so on. Most of the errors (36) arise where the dates have been written in by hand instead of type-written. The next most frequent source of error (11) is in the row extraction, where the algorithm has cut a row in half, or missed the year row entirely. Some of the cards appear to have been scanned poorly, such that there is a significant slant on the row - enough to upset our simple row extraction algorithm. In other cases, the handwriting above and below the relevant row(s) is dense enough to confuse the extraction algorithm. While it is possible to apply various measures to overcome these problems, we believe a more robust and simpler approach would be to dispense with the row-extraction phase entirely, and just have a single graph to search for the entire image. In six cases the OCR failed due to a major change in the type-writer font. This could be counteracted with a larger selection of character templates. Finally, the crossed-out images (3) are a small but interesting source of error. One of these is shown in Figure 4. Ideally, we’d like to recognize both the crossed out year, and the hand-written correction above it, since both are interesting sources of information. We are exploring the idea of extending the optimal-threshold template matching algorithm to cope with these crossed-out images. This should be possible because where the pen-lines cross the type-written ink the image is slightly darker than either ink alone. Therefore, a multiple non-contiguous threshold optimizing algorithm could exploit this, and still obtain a very good match for the underlying type-written character.

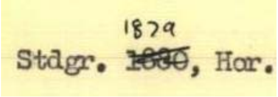


10 ms elapsed

Fabricius. 1794, Ent.

1794 : 240 : 0.628
1794 : 239 : 0.614
1794 : 241 : 0.592
1704 : 240 : 0.535

Figure 3. An example extracted row together with its search results where (1794, 240, 0.6628) is the best path.



1879
Stdgr. ~~1880~~, Hor.

Figure 4. An example crossed out image.

Note that we get a significant speed-up with the priority search over flat DP. These are fairly large recognition graphs, where each graph may contain as many nodes as the width of the row image (w), and up to ($w \times c \times g$) arcs, where c is the number of character classes and g is the num-

Result	n	%
Correct(1)	675	82.8
Correct(2-10)	69	8.5
Correct(11-21)	15	1.8
Error(hand-written)	36	4.4
Error (row-extract)	11	1.4
Error (diff-font)	6	0.7
Error (crossed-out)	3	0.4
Total	815	-

Table 1. Results at zero reject rate.

ber of possible inter-character gaps in pixel offsets. For our experiments $w \approx 700$, $c = 10$ and $g = 6$ - so there could be up to about 42,000 arcs in the graph - though most graphs would have far fewer than this.

Using flat DP to find the best paths for each of the 300 possible lexicon words (the years 1700 to 1999 in this case) individually takes about five seconds for each row graph. Using the priority search we cut this to an average of about twenty milliseconds. Note that with the implementation reported here, the priority queue algorithm does not return exactly the same results as flat DP, though this did not appear to affect the measured word recognition accuracy. The difference lies in a small oversight in our implementation. We returned the top n complete words found by the search, but the inaccuracy lies in the fact that these newly extended words may be less likely than ones which were below their immediate prefixes on the queue. The solution is to re-insert the complete words on the queue, and keep popping the queue until the first found n words are removed.

The system is implemented in Java and processes about 10 card images per minute on a 1ghz Athlon PC. The vast majority of this time is spent performing the sliding window OCR.

On the same set of images a leading commercial OCR system achieved around 70% accuracy, but was slightly faster than our system. Note that this comparison could be considered unfair to the commercial system, since our method has been tuned to the specific task at hand. However, it is a comparison that must be made, since if the commercial system was more accurate than our task-specific method, there would be no reason to use our method.

Currently, the system works with raw distances instead of probabilities. It is straightforward to map the distances into probabilities (e.g. by using distance histograms), but we've not yet done this. Doing so should improve the accuracy and also provide a sharper distinction between correct and incorrect words.

6 Conclusions

We have described an approach for searching for lexicon words in noisy images. The system works by sliding an OCR classifier over large portions of the image, and involves no prior segmentation into individual characters. The current system has been applied to reading year fields on museum index cards, and results for this are encouraging. The method is directly applicable to reading all the type-written text on these cards, and should achieve higher accuracy on other fields where the set of legal words is a much smaller fraction of the set of possible words, compared to the year field. The approach also has application to reading text in natural scenes, and this is the subject of ongoing investigations.

Acknowledgements This work was funded under research grant BBSRC/EPSRC 84/BIO11933.

References

[1] T. M. Breuel. A system for off-line recognition of hand-written text. In *Proceedings of 12th International Conference on Pattern Recognition (ICPR)*, pages 129 – 134 vol 2, 1994.

[2] D. Chen, J. Mao, and K. Mohiuddin. An efficient algorithm for matching a lexicon with a segmentation graph. In W. Lea, editor, *Proceedings of the Fifth International Conference on Document Analysis and Recognition*, pages 543 – 546. IEEE, 1999.

[3] G. Dzuba, A. Filatov, and A. Volgunin. Handwritten zip code recognition. In *Proceedings of the 4th IEEE Int. Conf. on Document Analysis and Recognition*, pages 766–770, Ulm, Germany, 1997.

[4] E. Ishidera, S. Lucas, and A. Downton. Top-down likelihood word image generation model for holistic word recognition. *Fifth IAPR International Workshop on Document Analysis Systems*, pages 82 – 94, 2002.

[5] F. Jelinek. Fast sequential decoding algorithm using a stack. *IBM Journal of Research and Development*, pages 675 – 685, (1969).

[6] F. Jelinek. Continuous speech recognition by statistical methods. *Proceedings of IEEE*, pages 532 – 556, (1975).

[7] D. Knuth. *The Art of Computer Programming – vol. 3, Sorting and Searching*. Addison Wesley, Reading, MA, (1973).

[8] S. M. Lucas, A. Tams, S. Cho, S. Ryu, and A. Downton. Robust word recognition for museum archive card indexing. In *Proceedings of the Sixth International Conference on Document Analysis and Recognition*, pages 144 – 148. IEEE, Seattle, WA, (2001).

[9] M. Mohamed and P. Gader. Handwritten word recognition using segmentation-free hidden markov modeling and segmentation-based dynamic programming. *IEEE Transations on Pattern Analysis and Machine Intelligence*, 5:548 – 554, (1996).

[10] W. Niblack. *An Introduction to Digital Image Processing*. Prentice Hall, 1986.

[11] M. Seeger and C. Dance. Binarising camera images for ocr. In *Proceedings of Sixth International Conference on Document Analysis and Recognition*, pages 54 – 58, Seattle, WA, 2001.