Probe Trees for Touching Character Recognition

George N. Sazaklis* Esther M. Arkin† Joseph S. B. Mitchell‡
Steven S. Skiena§

Dept. of Computer Science and Dept. Applied Mathematics and Statistics
State University of New York, Stony Brook, NY 11794

Abstract

The problem of touching characters is very important for the recognition of low quality text. A solution is presented here for the problem of touching character recognition for fixed font, using a decision tree classifier paradigm. The method is based on the concept of probe trees, a fast recognition method that uses character probes to acquire knowledge about the input samples. Touching characters are recognized without segmentation, so errors common in segmentation-based methods are avoided. Speed is achieved by constructing a decision tree for a specific font off-line, before any samples are seen. A deformation model is used to generate probes that withstand certain image distortions. Experimental results are presented in support of the method.

Keywords: Touching Characters, Probe Trees, Optical Character Recognition, Document Image Understanding.

*sazaklis@cs.sunysb.edu. Supported in part by a grant from Syngen Corp. and by the Strategic Partnership for Industrial Resurgence, College of Engineering and Applied Science, SUNY Stony Brook.

†estie@ams.sunysb.edu. Supported in part by NSF grant CCR-9504192.

‡jsbm@ams.sunysb.edu. Supported in part by NSF grant CCR-9504192, a Fulbright research award, and by grants from Boeing Computer Services, Bridgeport Machines, Hughes Aircraft, and Sun Microsystems.

§skiena@cs.sunysb.edu. Supported in part by ONR Award 431-0857A and NSF Grant CCR-9625669.

1 Introduction

A major source of distortions that must be addressed in a practical OCR system is touching characters. Touching characters can occur because of too little spacing between neighboring characters, excessive ink dispersion, or other causes (such as streaks introduced during photocopying that cross over many characters and prevent segmentation).

The shapes of the English letters do not guarantee an unambiguous distinction between single letters and touching character pairs. Notorious is the case of "m" versus a touching "rn" pair, where the distinction may be only in a tiny fraction of the image (or missing altogether). For a review of the field of OCR see [6].

A significant amount of research is devoted to finding a robust solution to the problem of touching character recognition in a variety of contexts; see [7, 9, 5]. Many of these methods, however, are based on a tentative character segmentation that is carried out before recognition. For example, [3] splits the characters in plausible points, using a neural network. Such an approach has the disadvantage that some characters may be separated into many pieces, each of which has to be assembled in a subsequent stage, creating additional broken characters. Along similar lines, [4] proposes segmenting the characters and later using a spell-checker to correct possible segmentation errors.

The *probe tree method* for OCR takes a given

set of models for the characters and constructs a hierarchical decision tree classifier that enables rapid discrimination among images of character instances. Specifically, at each internal node of the decision tree, there is an associated set of candidate characters as well as a *probe*; the "outcome" of the probe (typically mapped to a real number) determines which child is followed by the recognition algorithm during a processing of an input character. The leaves of the tree correspond to complete knowledge of the unknown character (the candidate set consists of a single element). A major advantage of the probe tree method (and decision tree classifiers, in general) is recognition speed, since the recognition algorithm performs simple local probes at each of a small number of nodes along a path from the root to a leaf of the tree. The algorithmic theory of the probe tree method was explored from a computational geometry perspective in [1, 2]. Experimental results on an implementation of these methods was given in our earlier work [8].

This paper extends our earlier work to address the problem of touching character recognition. We propose a novel kind of probe tree: The *prefix* and *suffix* probe trees. In contrast with traditional probe trees, whose purpose is to determine the identity of a single segmented character, prefix, or suffix probe trees can be used to determine the identity of a prefix or suffix of a sequence of touching characters respectively.

We should also note that our method achieves recognition without segmenting the sequence, so it avoids errors common in segmentation-based methods (e.g. [3, 4]).

The next section presents a summary of the concept of the probe tree and how it is used for OCR. Section 3 presents the main method for the construction of probe trees for touching characters. Finally, experiments, results and future work are presented in Section 4.

2 Probe Trees

In our discussion of the probe tree method, we follow much of the terminology used in [1, 2]. A *probe tree* is a decision tree that is used to classify segmented characters of a fixed font. Each node in the decision tree is associated with a set of candidate models; the root corresponds to the set of the whole alphabet S, while the leaves of the tree correspond to individual models. Each internal node is associated with a probe query that is applied to the image in order to decide which child to descend to next. A path from the root to a leaf in the decision tree represents a possible outcome for a particular image. Each probe query consists of a simple local operator. For example, we can average the intensity values on the image that lie within a selected small box, and compare the resulting value with threshold values that determine which branch of the decision tree is to be taken next. An example is illustrated in Figure 1 (from [2]).

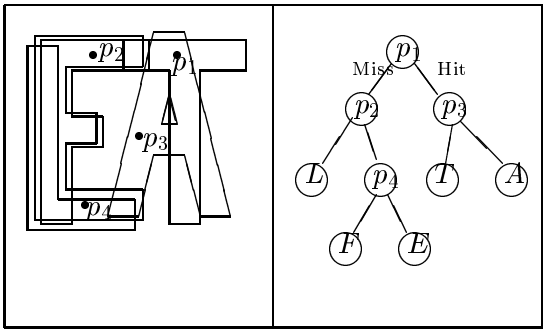

Figure 1: A set S of 5 character models (left) and a decision tree (right) that can determine, using "point probes", which model is actually present.

In this example, each probe is a single point of the input character with known coordinates. There are many variations on this scheme, for example we can query the bounding box of the character, or its position relative to the baseline, or sum the intensity values within a small box and compare the outcome with preset thresholds associated with the current

node. In order to apply the probes successfully, we have to place the input character in a known reference frame. For segmented characters, this process (the *registration*) is achieved by placing the character bounding box at prespecified locations in the pixel grid.

A major hurdle for OCR systems is that character images are distorted during printing, handling, positioning and scanning of the page. As a consequence of these sources of errors, not all pixels in the input sample are equally reliable. We employ two methods to increase probe reliability:

- We have developed a simple *error model* that considers certain effects as physical processes that affect the character image. Our current model takes into account the random positions that characters will take with respect to the pixel grid and the blurring effects of the point spread function (which is modeled as a two-dimensional Gaussian bell). The pixel intensities are considered random variables that have a certain distribution, while different probes are judged according to their robustness (that is, the probability that a probe will produce a value which will send the recognition process to the correct child of the current node). Whenever the selected probe is not robust enough for a certain model, we place the model in each of the children of the node; thus, no decision is made for the model at this stage. We pay for this increased reliability by having a larger tree. Our current and future work is directed at attacking other common defects, particularly document skew and noise.

- Since the probes utilize only a small area of the sample during recognition, we can use the rest of the area to *verify* the tree's suggestion. The verifier consists of additional probes that depend on the tree outcome and can discriminate between the suggested model and any other model. If the candidate passes all of the tests successfully, the tree suggestion is output as the solution. If not, there has been a potential error either during the recognition, or verification; in this case, we can reject the character, or discard the solution and cascade the sample to a different tree/verifier combination.

3 Probe Trees for Touching Characters

The main idea for a probe tree suitable for touching characters is to use the structure that exists inside a touching character pair: We can take advantage of the fact that pairs are not really new shapes, but are formed by concatenation of other letters. The objective is to recognize each constituent character individually.

As was pointed out earlier, registration has to precede recognition. Although the bounding box of individual characters was a convenient way to do registration of isolated characters, it is not available when characters are touching. In the latter case, we can only use the left side of the leftmost character, or the right side of the rightmost character along a sequence of touching characters, to position the characters along the horizontal direction. A tree that probes on the leftmost (rightmost) character of a sequence using the left (right) side of the character for horizontal registration is called an *L-tree (R-tree)*. We use the baseline as a reference line along the vertical direction.

The use of the baseline for vertical registration means that we have to detect the baseline before we attempt to recognize any touching characters. This is done in two phases:

(i) We use a tree for single characters (that does not need the baseline position) to recognize some single characters on the line. Of course, some touching characters may be misclassified during this phase, since the tree has no knowledge of the shapes of touching character pairs. However, the results are used only to detect the baseline; misclassifications have no effect.

(ii) We apply a least squares fit procedure on

the bottom points of recognized characters that are not descenders (do not recede below the baseline). The least squares fit will find the straight line that minimizes the distance error from the bottoms of the characters, that is, an estimate of the baseline. This technique can tolerate document skew, in a local fashion, but it makes the reasonable assumption that the baseline is a straight line (i.e. it will fail in the rare case that text is printed along a curve).

Also note that a technique that simply tries to fit a straight line to the bottom of the character bounding boxes without first identifying them will fail since it does not know which characters are descenders. Techniques for line fitting in the presence of outliers exist, but they assume an upper limit on the number of outliers. In our case, there is no upper limit. Consider for example, the word "puppy" alone on a line: Having just the bounding box information, any least squares method will fit the bottoms of all the descenders, and consequently will fail to detect the real baseline, hindering the task of subsequent recognition stages.

Once the touching characters have been registered in place, probing may begin. Our technique is called *prefix probing*: We try to gain knowledge about the unknown shape at hand, by looking at a prefix of the shape.

Definition 1 *A model m_1 is a* prefix *of model m_2, if m_1 is indistinguishable from a* left *part of m_2. Similarly, a model m_1 is a* suffix *of model m_2, if m_1 is indistinguishable from a* right *part of m_2.*

For example, "l" can be considered a prefix of "h", since the "l" shape is embedded in the left part of "h". Similarly, "l" can be a suffix of "d".

The probes satisfy the following property (1):

> *When a prefix probe excludes a model from further consideration, it has probed (and rejected) within the horizontal extent of that model.*

Figure 2: This probe could be used to discriminate between "l" and "u", only when we have single characters: If the probe hits black (white), we have a "u" ("l"). In the presence of touching characters, though, it is erroneous to exclude "l" whenever the probe hits black, since the probe outcome cannot be trusted. Only if "l" is excluded by a probe hitting within its horizontal range, is it legitimate to eliminate it from further consideration.

That is, in order to exclude a certain letter, the probe has hit in the area of the sample that letter would occupy, if it were present. This property ensures that the results can be trusted and the excluded letter is definitely not present; i.e. we have not erroneously excluded it, because we have accidentally probed an the neighboring character (see Figure 2).

Touching characters tree construction enforces the above property. The important idea here is that we do not need any knowledge from the character we are probing (the sample) in order to enforce it. All relevant parameters are known at tree construction time and therefore the property can be enforced at that time (off-line, rather than at recognition time).

We can partition the models at a particular node of the L-tree with two children, C_1 and C_2, into three subsets: The models that belong to both children are the set $I = C_1 \cap C_2$, the models only in C_1 are $L = C_1 - I$ and the models only in C_2 are $R = C_2 - I$. Recall that I might be non-empty, since models for which the probe is unreliable are placed in all children. Then $R \cup L$ consists of all the models that might be excluded at this node, if the recognition process goes into the other child. Therefore, to satisfy the above property, the probe is restricted to hit inside the intersection of their horizontal ranges. Since the

left sides of the models coincide because of the registration (L-tree uses the left side for horizontal registration), the intersection region is the model with the minimum width from $R \cup L$.

We can define the *probe horizontal range* as the maximum distance of any pixel that is used in the probe from the x-position of the left side of the models (when we have left registration). Therefore, during tree construction, a probe is allowed to make a decision for some model (i.e. place it in only one child but not in the other), only if the horizontal range of the model is larger than the horizontal range of the probe. Models that lie within the probe horizontal range are placed in all children and no decision is made for them at this stage.

According to the above, our top-down tree construction algorithm can be formulated as follows for a node with degree 2:

```
At current tree node, with model set M
For each probe p:
    Add to I_p all models lying within
      the horizontal range of p
    Determine L_p, R_p for rest of
      models M − I_p
    Compute the probe robustness.
Among all probes, select probe, q,
    of highest robustness;
Set C_1 = I_q ∪ L_q, C_2 = I_q ∪ R_q.
Recursively, build the two subtrees.
```

In summary, the probes may discriminate only between models that extend beyond the probe horizontal range; this will ensure that the aforementioned property is satisfied.

The natural question now is: How effective is prefix probing?

We can prove that prefixes and suffixes present hard cases for our algorithm: The produced tree cannot discriminate between shapes that are prefixes of one another.

Lemma 1 *Let m_1 and m_2 be two models. Without loss of generality, let $width(m_1) < width(m_2)$. Then, the L-tree (R-tree) constructed by the above algorithm cannot distinguish between models m_1 and m_2, if m_1 is a prefix (suffix) of m_2.*

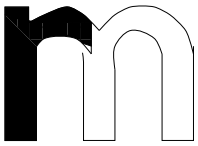

Figure 3: "r" is a prefix of "m", so any probe in the L-tree discriminating between the two has to hit beyond the right side of "r"

Proof. For the sake of contradiction, let m_1 be a prefix of m_2 and assume that there exists a node v in the L-tree with probe p that manages to discriminate between the two models (that is, there is no child of v that contains both m_1 and m_2). Since m_1 is a prefix of m_2, and they are both left-registered, it is clear that no probe hitting within the horizontal range of model m_1 is going to discriminate between the two models. Since probe p discriminates between them, it has to include some pixel beyond the right side of m_1 (see Figure 3).

Since probe p is hitting beyond the right side of model m_1, we can conclude that model m_1 lies within the horizontal range of probe p and has been placed in set I_p by the algorithm. Eventually, the algorithm places set I_p in all the children, and in particular, in the child that contains m_2. Therefore probe p cannot discriminate between m_1 and m_2, a contradiction. The proof for R-trees and suffixes is similar. □

In summary, this lemma stipulates that we cannot tell whether a certain character is present using prefix probing only. What we can say is that some character is *not* present by collecting negative information; i.e., if a prefix of m is not there, then m cannot be present. This is the reason that property (1) had to be formulated in terms of exclusion of models.

Thus the two trees that are to be constructed by the algorithm (the L-tree and the R-tree) do not discriminate between models associated with each other through the prefix relation. Instead, those models are placed to-

gether in tree leaves. If we end up at one of these leaves during recognition, we must do more work to identify the character unambiguously; specifically, we have to probe deeper into the character sequence for additional evidence that will narrow down our options for the leftmost character.

In the case of character pairs, this means probing in the entire sample area. To do that, we construct an additional decision tree that regards the entire input candidate as a single shape. The task of this tree will be to distinguish between pairs of letters. Note that this is not the same as constructing a complete tree for touching character recognition using character pairs as models, since we have some partial knowledge about the left and right character derived from the L- or R-tree. This partial knowledge can be used to concentrate on a small subset of character pairs, instead of all possible letter combinations. Typically this tree will be quite small, as it only has to distinguish between characters that are associated with the prefix relation.

In this small tree, we can use all kinds of probes or registration. For example, use of the bounding box is allowed, since it is going to be estimated correctly in each case.

We call a leaf of the L- or R-tree *incomplete,* if it contains more than one model. For every pair of incomplete leaves l and r inside the L-tree and R-tree, containing model sets M_l and M_r respectively, where $|M_l| > 1$ and $|M_r| > 1$, we construct a small tree on the set of pairs: $M_l \times M_r$ plus the common models in M_l and M_r (the common models are needed to take care of segmented characters). This is the final stage for touching character tree construction:

```
For each incomplete leaf l of L-tree
    with model set M_l
  For each incomplete leaf r
    of the R-tree with model set M_r
    Create a new model set
      M = (M_l × M_r) ∪ (M_l ∩ M_r)
    Build a tree on M using any probes
      and registration
```

This algorithm produces a forest of small trees that are used during the recognition process, when we end up at an incomplete leaf for both L-tree and R-tree.

Note that we might use the small tree not only for touching characters but also for single character recognition. When we are recognizing an "m", for example, we end up at a leaf "r m" at the L-tree (since "r" is a prefix of "m") and at a leaf "m n" at the R-tree (since "n" is a suffix of "m"). Then, the small tree corresponding to these two leaves is used to complete recognition.

This scheme can now take care of all touching character pairs. Moreover, it can be extended to recognize touching character sequences of arbitrary length. We are naturally interested in keeping small the overall number of trees, as this will play a role on the memory requirements as well as the time taken for tree construction. From the algorithm above, the number of small trees is proportional to the product of the incomplete leaves of the L-tree and R-tree. It can be easily shown that the prefix relation is an order relation and that the number of incomplete leaves on the L-tree is equal to the number of non-trivial chains in this relation (chains with length more than one). This, in turn, strongly depends on the letter shapes present in the alphabet.

Of interest also is the construction of the model shape for character pairs, something we have to do when we regard the whole pair as a single model. We create the model for a touching character pair by concatenating the models of the constituent characters so that their sides are consecutive columns in the character frame. The reasoning behind this choice is as follows: If the characters are separated by one column or more, they are going to be segmented and recognized individually, so we need not be concerned with that case. On the other hand, under regular spacing conditions, the characters are printed apart by the system (i.e. we assume that the printing device will not deliberately backspace the head to create touching characters). So, under regular spacing conditions, neighboring characters do not

	Originals				Photocopied			
	Single Char.		Touching Char.		Single Char.		Touching Char.	
	Amount	Rate	Amount	Rate	Amount	Rate	Amount	Rate
Segmentation	14508	98.04 %	14732	99.55 %	13936	94.17 %	14406	97.35 %
Recognition	14433	99.48 %	14690	99.71 %	13685	98.20 %	14152	98.24 %
Rejections	71	0.49 %	34	0.23 %	227	1.63 %	220	1.53 %
Errors	4	0.03 %	8	0.05 %	24	0.17 %	34	0.24 %
Overall Perf.	14433	**97.53 %**	14690	**99.27 %**	13685	**92.47 %**	14152	**95.63 %**

generally appear less than one pixel apart. We can conclude that our choice to create models for touching characters by concatenating the bounding boxes is reasonable.

The ligatures (e.g. ff, fl, fi, ffl), on the other hand, are considered separate shapes by the system and should be added to the set as separate models.

Once the touching characters forest is built, it is easy to see what is happening during recognition. On the first pass over the characters, a tree is used to detect the baseline. At the next stage, characters can be registered having the baseline as a reference line on the vertical direction and using the left or right side for registration along the horizontal direction.

Next, we descend down the L-tree. If we end up at a singleton leaf, we know the identity of the left character completely. After a width comparison, we can decide whether we have one or two characters present. If we have a pair of characters, we can cut the known character off and use a single-character tree to recognize the other character. On the other hand, if we end up in a leaf v with more than one model (an incomplete leaf) at the L-tree, we register the sample on the right side and use the R-tree to obtain some information about the right character. If we manage to determine the identity of the right character completely (i.e. we end up in a leaf w that includes a single model), we cut it off. Then, if we had a pair, we can easily recognize the left character, since it is alone now in the character frame. If leaf w is also incomplete, then by the tree construction algorithm, there exists a small tree $T_{(v,w)}$ corresponding to these two incomplete leaves of the L-tree and R-tree. We use it for final determination of the identity of the character pair.

4 Results

We evaluated the proposed method for touching character recognition by scanning running text of 14798 characters at Times Roman font at 10pt size. We used two single-character trees and observed their performance with and without an additional tree for touching character recognition cascaded to them. The table above compares the results for single vs. touching character recognition for original printouts and photocopies. Rates are computed as a percentage of the input to the relevant stage. E.g. the segmentation rate is computed on the total number of characters, while recognition is computed on the properly segmented characters. We see a large improvement on the segmentation rate when touching characters are recognized. The overall system rate (which is defined as the number of correct characters in the output over the total input to the system) was also improved. The higher number of errors on touching characters is due to the fact that the touching characters tree is cascaded to the single characters tree. So all errors from the first phase will go through to the final result.

In conclusion, we have described a novel extension on the concept of probe trees as they are used for fixed-font OCR. The main idea in the trees for touching characters is to probe only at a prefix (or suffix) of the shape and

deduce as much information as possible about the identity of the character, before we spend more resources to examine the whole shape. The method is easily extendible to recognition of longer sequences of touching characters; this is on our agenda for ongoing work in the near future.

References

[1] E. M. Arkin, M. T. Goodrich, J. S. B. Mitchell, D. Mount, C. D. Piatko, and S. S. Skiena. Point probe decision trees for geometric concept classes. In *Proc. 3rd Workshop Algorithms Data Struct.*, volume 709 of *Lecture Notes in Computer Science*, pages 95–106. Springer-Verlag, 1993.

[2] E. M. Arkin, H. Meijer, J. S. B. Mitchell, D. Rappaport, and S. S. Skiena. Decision trees for geometric models. *Proc. Ninth ACM Symposium on Computational Geometry*, pages 369–378, 1993.

[3] J.-H. Bae, M.-H. Park, and H.-J. Kim. Recognition-based extraction of characters in printed documents. In *Proc. of the 1996 International Conference on Neural Information Processing, ICONIP'96*, volume 2, pages 1102–7. Singapore Springer-Verlag, Sept. 1996.

[4] S. Liang, M. Ahmadi, and M. Shridhard. Segmentation of touching characters in printed document recognition. In *Proc. of 2nd International Conference on Document Analysis and Recognition (ICDAR '93)*, pages 569–72. IEEE Comput. Soc., Oct. 1993.

[5] Y. Lu. On the segmentation of touching characters. In *Proc. of the 2nd International Conference on Document Analysis and Recognition (ICDAR '93)*, pages 440–3. IEEE Comput. Soc., Oct. 1993.

[6] S. Mori, C. Suen, and K. Yamamoto. Historical review of OCR research and development. *Proc. IEEE*, 80:1029–1058, July 1992.

[7] G. Raza, N. Sherkat, and R. Whitrow. Recognition of poor quality words without segmentation. In *Proc. of the IEEE International Conference on Systems, Man and Cybernetics*, pages 64–9. IEEE New York, Oct. 1996.

[8] G. Sazaklis, E. M. Arkin, J. S. B. Mitchell, and S. S. Skiena. Geometric decision trees for optical character recognition. In *Proc. Thirteenth ACM Symposium on Computational Geometry*, pages 394–396, Nice, France, June 4-6, 1997.

[9] P. Stubberud, J. Kanai, and V. Kalluri. Adaptive image restoration of text images that contain touching or broken characters. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, volume 2, pages 778–81, Los Alamitos, CA, Aug. 1996. USA IEEE Comput. Soc.