Adding Feedback to Improve Segmentation and Recognition of Handwritten Numerals

by

Susan A. Dey

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Computer Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 21, 1999

[June 1999]

Author__________ ____________________
Department of Electrical Engineering and Computer Science
May 21, 1999

Certified by____ ________________
Dr. Amar Gupta
Thesis Supervisor

Accepted by________ ________________
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Adding Feedback to Improve Segmentation and Recognition of Handwritten Numerals

by
Susan A. Dey

Submitted to the
Department of Electrical Engineering and Computer Science

May 21, 1999

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT:

WinBank is a system which performs automated reading of handwritten documents, particularly of strings of handwritten numerals on bank checks. A commonly-used strategy for reading handwritten strings is a combination of segmentation and recognition. In the WinBank program, segmentation involves both the separation of touching characters and the merging of character fragments to other pieces. An inherent problem with the segmentation/recognition strategy is the fact that if segmentation is done incorrectly, the erroneous characters cannot be recognized. This paper discusses a new method for feeding rejected characters back into the segmentation process to perform error recovery. Such feedback reduces the frequency with which segmentation errors occur, and increases the probability that each character will be recognized.

Thesis Supervisor: Dr. Amar Gupta
Title: Co-Director, Productivity From Information Technology (PROFIT) Initiative
Sloan School of Management

TABLE OF CONTENTS

1 Introduction

Banks and other financial institutions are currently facing high costs because they still rely heavily on paper documents. In an ever-increasingly digital world, paper documents are becoming outdated. For many institutions, the first step in processing paper checks is performed by human operators who manually enter the information written or typewritten on them. Errors are highly undesirable, so often multiple operators handle the checks for redundancy. This improves reliability but requires paying for a greater amount of labor time. Since even experienced human operators require several seconds to process a check, throughput can only be increased by increasing the number of human operators processing checks. If these people were replaced by specialized computer image processing, a great deal of money would be saved.

The WinBank system [1] takes scanned-in checks and determines the dollar amounts they are written for. By processing checks in a fraction of the time required by human operators, with at least as much reliability, this program could reduce bank costs as well as improve processing speed.

Checks contain redundant information by including a field for digits (the courtesy amount block, or CAB) and a field for written-out words (the legal amount block), both of which represent the amount of money the check is written for. Recognition of free handwriting is a particularly complicated problem, but a simplification in the case of recognizing check amounts is that it is possible to extract all necessary information from only the digits written in the courtesy amount block. The legal amount block is sometimes used to augment or verify the digit recognition [3], but since processing natural handwriting is very difficult, the increase in complexity and reduction of efficiency usually outweigh the benefits. In the case of the WinBank system, only the digits in the courtesy amount block are used.

A great variety of strategies have been applied to recognize handwritten numeral strings. Most of these can be described as *segmentation-based*, in which the string is first broken into units suspected of representing individual digits. These are passed to a recognition module which identifies the isolated numerals. [1, 2, 3, 4, 5, 6] Another approach, sometimes called *segmentation-free*, integrates the segmentation and recognition into a single process. [8, 9, 10] Blends of these strategies have also been applied to the problem, using combinations of segmentation-based preprocessing steps and segmentation-free completion. [11] The WinBank system uses a segmentation-based approach, dividing the procedure into separate steps for segmentation and recognition.

A critical problem with such a segmentation-based strategy is that improperly segmented numerals cannot be recognized. If digits are segmented wrongly, the recognizer can do nothing but reject the string, or even worse, it could improperly interpret the nonsense characters as erroneous numbers. In many check-processing systems, cases of this nature result in the checks being passed back to human operators for interpretation. The previous version of the WinBank system took this strategy [1].

Improper segmentation can often be caught, however. Whenever the recognizer is unable to identify a digit, or especially two digits in a row, it is likely that the digits were badly segmented. This paper describes an improvement which was added to the segmentation-based system to allow unrecognized digits to be fed back to the segmentor for re-evaluation. Such feedback allows cases of improper segmentation to be repaired. The improvement reduced the frequency with which strings were rejected and improved the percentage of strings which were correctly recognized. One goal of implementing this improvement correctly was to avoid increasing the frequency with which strings are improperly recognized, or even to reduce it.

2 Segmentation Overview

To understand the flow of segmentation-based feedback systems, an explanation of the steps taken during processing must be given. Figure 1 illustrates the program flow in the WinBank system. The program begins with a previously-scanned grayscale check image. The first step which must be done is *binarization*, in which the image is converted to a black-and-white representation. This process involves filtering the image to remove unwanted noise, locating the courtesy amount block region, and choosing a threshold color which marks the dividing point between black and white within the grayscale colors. The process of thresholding is described in greater depth in Section 6.1, and CAB location is explained in Section 6.2.

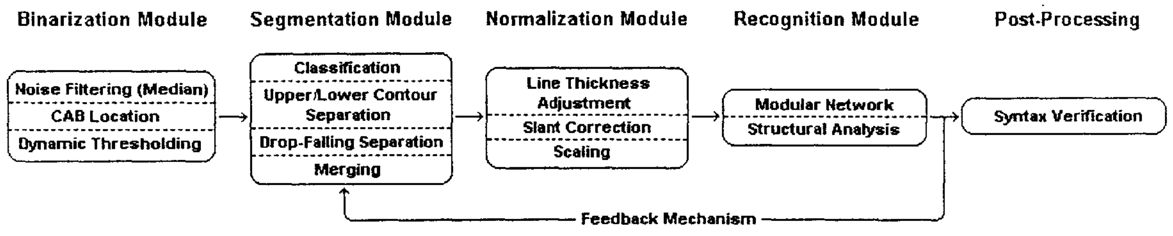


Figure 1: Program flow.

Once the image has been converted to a useful representation, the writing must be separated into individual digits, or segments. This process is known as *segmentation.* The process of segmentation is explained below. Conceptually it involves merging fragments of digits to create whole units and separating touching digits into their component pieces. Once a suitable breakdown of the image is obtained, the list of characters is allowed to continue through the process of normalization (which prepares them for recognition) and recognition.

After all of the digits have been recognized, a final post-processing step is used to verify that the recognized value makes sense in terms of money. For example, this uses a few syntax rules to ensure that the detected commas have a proper relationship with each other, so that strings like "1,0,0" are not accepted. The syntax verification is described in greater detail in Section 6.6.

To understand the feedback process, the segmentation process must be explained in more detail. The problem of segmentation has been studied in depth with a variety of different approaches. The problem is explained here, and an overview of the approaches which have been used is given in Section 3. Some research has been done into advanced mathematical separation techniques [6] or the application of neural networks [8, 9, 10], but more often images are divided into individual digits using structure-based techniques [1, 2, 3]. These analyze features of the image such as corners, straight lines, loop structures, and upper and lower contours.

The structural segmentation process is complex and riddled with special cases. A combination of carefully planned algorithms and simple heuristics must be used to reliably divide the region of interest into individual digits. First, continuous connected regions of pixels must be found and isolated. This provides a rough initial estimate of the characters. The connected regions are then classified as fragments of characters, whole characters or multiple touching characters that need to be separated. Then the regions that are fragments are corrected by merging them with other regions, and the regions that are multiple characters are separated into their component pieces. This process is continued until all regions are classified as single digits. The set of digits is then passed through the recognition module.

As stated above, one part of the segmentation process is the classification of regions as being composed of partial, single, or multiple characters. The WinBank classification module employs a few simple heuristics. The details of these heuristics are not important for the purpose of this discussion, but it must be noted that the classification can only be used as a rough guide. Regions are classified with reasonable enough accuracy for these predictions to be useful, but not well enough to be relied on as correct in all cases. Often peculiar combinations of fragments or touching characters can be classified as single characters. Sometimes large characters may be classified as composed of multiple digits. Therefore the results of the classification module can be used to guide the initial segmentation of the image, but this segmentation must be double-checked by making sure the recognized character string makes sense. The classification process is covered in more depth in Section 4.2.

Segmentation algorithms are primarily concerned with dividing touching characters into separate pieces. A number of structural segmentation algorithms have been developed, such as the Upper/Lower Contour [1, 16] and Drop-Falling [17] methods used in the WinBank system. These algorithms can be viewed as "black boxes" which operate on an input image and output a path of points along which to divide the image into two separate regions. (Images that are comprised of three or more connected characters can be separated by applying a succession of segmentation operations, marking off one digit at a time.) Different algorithms often result in different cut paths, and some algorithms have better results in certain situations than others do. The WinBank system uses a hybrid technique which makes use of several different segmentation algorithms. The operation of these algorithms is omitted for purposes of this overview, but they are described in more detail in Section 6.4. Several paths are built using different methods, and the best one is chosen based on some simple heuristics.

Another operation which is used to correctly assemble the pieces of the image is merging. Fragments which are too small to be full characters are examined and possibly merged to nearby neighbors if they are reasonably related to each other. Again, the method of determining such criteria is unimportant for this discussion. Merging is covered in more detail in Section 6.5.

Finally, once the image has been divided into units that are deemed acceptable by the classifier, those regions are passed to the recognition module for evaluation. If one or more digits in the string cannot be identified, they are marked as "unknown." In such a case, the check is rejected and must be passed to a human for identification.

3 Related Research

As noted above, a problem with the strategy of segmenting and then recognizing digits is that improperly segmented numerals cannot be recognized. If digits are segmented wrongly, the recognizer can do nothing but reject the string. Several strategies have been developed to deal with this problem. This section gives an overview of several different approaches used in the field.

3.1 Iterated Segmentation and Recognition

A simple approach which can be used to improve the process described above is alternation between segmentation and recognition. The basic idea of such as strategy is that an initial segmentation of the image is performed and those digits are passed to the recognition module. If any digit is unrecognized, that digit is split using a specific segmentation algorithm and recognition of the two resulting pieces is attempted. If neither piece is accepted, it is assumed that the segmentation was done improperly and another algorithm is used to separate the digit. If one or both pieces are accepted, any remaining rejected pieces are segmented in a similar fashion. This process is illustrated in Figure 2. This type of approach is used by Congedo et. al. [2] and Dimauro et. al. [3].

The advantage of this strategy is that it is clean and straightforward. However, several assumptions are made which do not reflect most real-life situations. For instance, every unrecognized block is assumed to need segmentation, which means that even properly-segmented digits which are rejected because they are unreadable will go through a process of segmentation. It also means that fragments of blocks are never treated properly because there is no strategy for merging broken pieces together. It does not take advantage of the fact that some segmentation algorithms may be better in some situations than others; it has an ordered set of segmentation algorithms and always uses them in that order. Finally, this strategy is inefficient, because it attempts recognition before doing any segmentation, while it may be possible to detect that multiple digits are touching without doing any recognition. Since recognition is usually a computationally-intensive process, simple tests for touching digits done before recognition can save a lot of unnecessary processing.

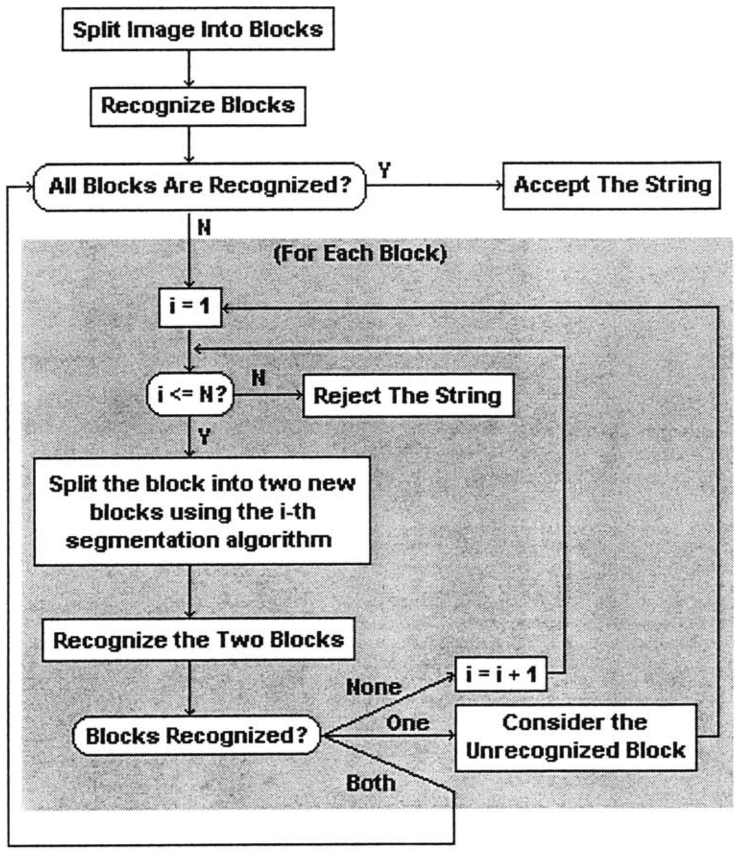


Figure 2: Program flow for the iterated segmentation and recognition strategy.

3.2 One-Step Segmentation Reconsideration

A somewhat similar strategy, employed by Blumenstein and Verma [4], uses conventional methods to generate a set of candidate segmentation paths within a string. It then makes use of a special neural network to verify those paths. The neural network accepts or rejects the candidate paths, resulting in a corrected set of paths. Thus the program goes through two phases during segmentation: path generation by heuristic means, and path selection by the neural network. No new paths are generated after the neural network phase, so no feedback from recognition is considered.

A similar strategy is used by Lee et. al. [5], who build a graph of likely cuts and search the graph for a combination of cuts which maximizes the character confidences as determined by the recognition module. See Figure 3 for an illustration of the graph-searching method of segmentation.

This relatively simple strategy has the advantage that it finds the highest-confidence segmentation from its candidate paths. One drawback is that many candidate segmentations must be passed to the recognizer, thereby resulting in extra computation which could possibly have been avoided. It also fails if one of the correct cuts was not included in the original set of candidate cuts, or if one of the cuts was slightly wrong. In that case even the best segmentation path would have a low confidence and the string would have to be rejected. Contextual information such as spelling and grammar rules, as described in Section 3.3, could be added to such a system to improve segmentation and recognition even more.

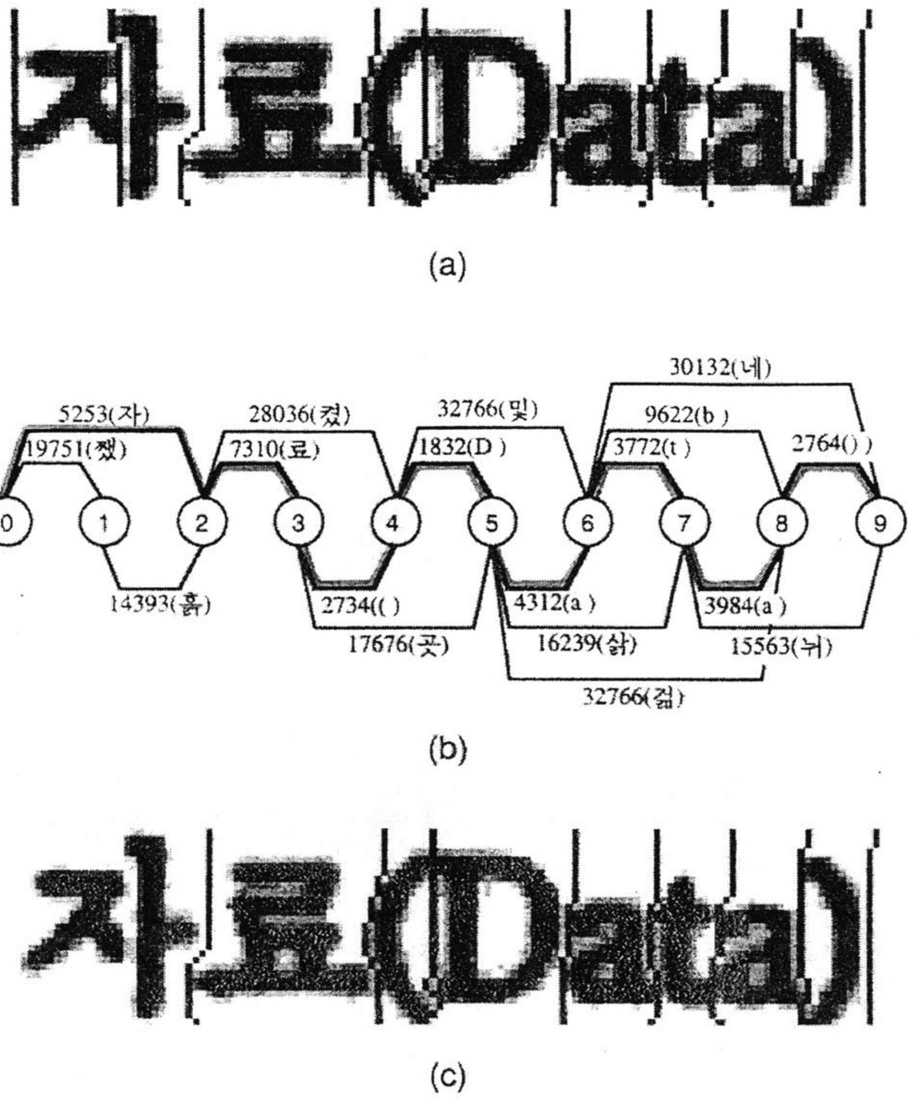


Figure 3: A set of cuts can be modeled as a graph. Searching the graph to find a path which maximizes recognition confidences and results in the best segmentation. Figure reproduced from [5]. **(A)** The initial segmentation. **(B)** The resulting graph and confidences. The best path is marked in gray. **(C)** The final segmentation.

The same concept of choosing the highest-probability segmentation can be seen in [6], where a Modified Quadratic Discriminant Function (MQDF) is used to perform recognition. The optimum splitting is chosen by mathematically evaluating the best combination, rather than searching a graph. The amount of computation necessary is minimized using dynamic programming techniques.

3.3 Use of Contextual Information

Depending on the task which is being accomplished when reading handwritten and typewritten documents, more information may be available which could be used to improve the segmentation process. For instance, when the numerals of handwritten zip codes are being read, segmentation must always result in five digits. Segmentations which lead to more or less numerals can be reexamined and corrected using this information. Such a constraint on string length is not available when dealing with reading the amounts of bank checks, however.

Other forms of contextual information may also be put to use. When full pages of text are being read, rules of spelling and grammar can be applied to detect and correct mistakes in segmentation and recognition. For instance, Hong et. al. [7] use a technique of developing *lattices* of related characters and their recognition confidences to choose words based on the most likely character breakdowns. Such use of spelling rules allows the system to tolerate a relatively high degree of error because a number of alternatives are considered concurrently.

Further information is known when recognition is restricted to typewritten text. Character sets corresponding to different fonts can be matched to segmented characters and used to diagnose segmentation errors. Average character size and spacing and average word spacing can also be put to use to improve segmentation.

3.4 Segmentation by Recognition

A completely different approach is Centered Object Integrated Segmentation and Recognition (COISR). In this approach, a single neural network is used to perform the function of segmentation and recognition [8]. It is assumed that segmentation is not a necessary precursor to recognizing a character, or to *learning to recognize* a character. Instead a "sliding window" is moved along the sequence of text, and the system is trained to recognize what is centered in its input window. The neural network recognizes centered characters and indicates that nothing is centered if it finds no character to be centered in the window. Segmentation is therefore unnecessary in this case, because touching characters can be recognized as they come into the center of the window, regardless of the connection between them.

The striking property of COISR and other similar approaches is that segmentation is essentially performed as a result of recognition. Rather than being done as a precursor to recognition, the correct segmentation of the image results from the sequence of recognized pieces.

One problem with this approach is the large amount of processing it requires. It generates too many possible segmentations to be truly efficient. The neural net is consulted with each small increment of the window location, leading to a great deal of time spent recognizing redundant information (finding a character several times or repeatedly finding that no character is centered on the window). The window slide rate can be adjusted to minimize unnecessary computation, but care must be taken to prevent sliding the window too quickly and moving past characters without recognizing them.

One method of performing such optimization is proposed by Martin et. al. [9]. The observation is made that the human eye does not scan progressively across a passage of text, but instead moves in *saccades* (distinct eye movements). The eye moves forward in *ballistic* saccades, and makes smaller jumps backward in *corrective* saccades when it moves too far. A single neural network learns to jump from character to character, making corrective jumps when necessary, and to classify the centered character when properly fixated. Thus the neural net learns to control the movement of its input window as well as to recognized what is in that window. This optimization reduces the amount of computation necessary to process a string.

A somewhat different strategy which also performs segmentation as a consequence of recognition is the Self-Organizing Integrated Segmentation and Recognition (SOISR) approach. The system described by Keeler and Rumelhart [10] also recognizes connected characters without segmenting them. Rather than requiring that the character being recognized be in a particular place, the neural network can be activated differently in different areas. A series of neural network "sheets" is used, one for each possible output digit. The neural network sheet for a digit is activated only in the area where a character is. For example, if a "3" and a "5" are connected, the "3" sheet is activated in the area of the image where the 3 is, and the "5" sheet is activated in the area where the 5 is. This concept is illustrated in Figure 4.

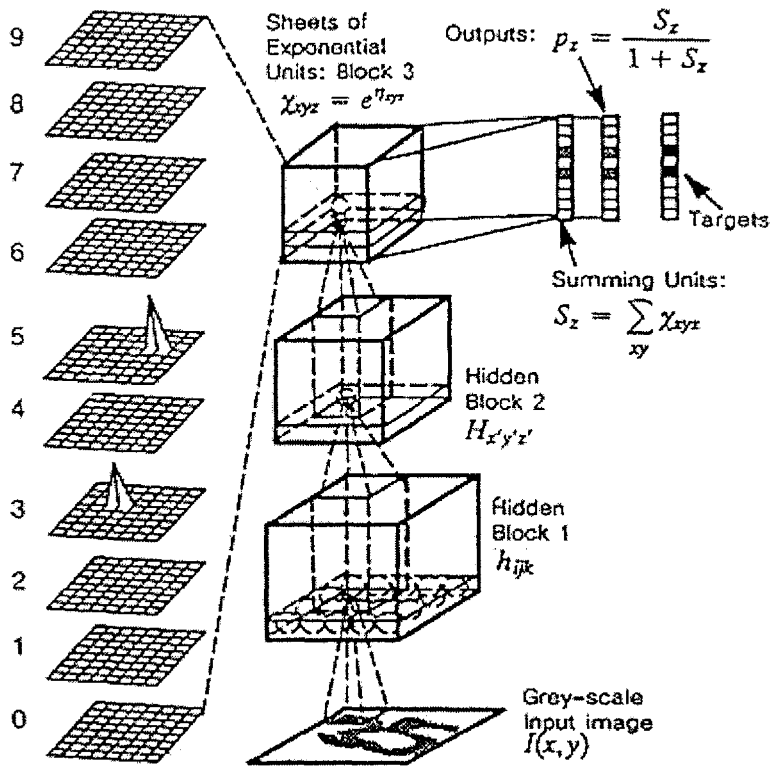

9
8
7
6
5
4
3
2
1
0
Sheets of Exponential Units: Block 3
$\chi_{xyz} = e^{\eta_{xyz}}$
Outputs: $p_z = \frac{S_z}{1 + S_z}$
Targets
Summing Units:
$S_z = \sum_{xy} \chi_{xyz}$
Hidden Block 2
$H_{x'y'z'}$
Hidden Block 1
h_{ijk}
Grey-scale input image
$I(x,y)$

Figure 4: A sheet of a self-organized neural network is activated over the area of the image where its targeted character is present. Figure reproduced from [10].

One difference between SOISR and COISR is that the self-organizing neural networks must be trained on hand-segmented characters. A rough segmentation done by manually specifying boxes around the target segments is sufficient, however. After the neural network is trained, it can recognize characters without any required position information. It recognizes connected characters in a single pass, making it more efficient than the sliding window technique used in COISR.

4 Feedback Architecture Design

4.1 Overview

Since the preexisting WinBank system used a segmentation-based approach, the feedback architecture added to it has been integrated with such operation. Its design loosely resembles the segmentation-based systems described in Section 3. It is particularly related to the iterated segmentation and recognition model. However, it makes several improvements on that model by applying heuristics at appropriate places. The iterated segmentation and recognition model initially assumes that the components of the image are correctly separated and attempts to recognize them as they are. It only resorts to segmentation if the components are not recognized. This means, for example, that connected digits are first passed to the recognizer as a single unit and only separated after they are rejected. A big drawback of such an approach is that recognition is usually a costly process involving a great deal of computation. Often simple heuristics can be applied to take the correct steps during early processing, reducing useless computation.

The feedback system used by WinBank avoids unnecessary recognition attempts as much as possible. It begins in the same way as the iterated segmentation and recognition model above by identifying the components in the image. Rather than passing them directly to the recognition module, however, the classifier described in Section 2 is used to classify the components as partial, single or multiple characters. Only single characters are sent to the recognizer. Multiple-character regions are immediately separated, and fragments of characters are immediately merged with their most promising neighbors. These adjusted characters are re-classified, and the process continues until all pieces of the image satisfy the criteria used to classify single characters. See the discussion in Section 4.2 for an explanation of the classification process.

Candidate paths to divide multiple-character regions are identified by applying several different segmentation algorithms. Rather than imposing a pre-determined order on these algorithms, the segmentor applies all available algorithms to the image. The resulting paths are ranked to identify the most likely candidates. The image is divided along the highest ranked path, and the two resulting regions are classified. The segmentation process continues as necessary, until all pieces are classified as digits. These digits are then sent to the recognizer.

If a character has been separated into two pieces and both are rejected by the recognizer, the next highest ranked path is applied until the list of available paths is exhausted, in which case the string is rejected. If one of the resulting segments is accepted and one is rejected, the rejected character may require further segmentation. This process is recursively applied until all regions are recognized or until the string is rejected.

Merging can be similarly reconsidered. If a fragment is merged to another piece of the image and the resulting character is not recognized, the merging may have been in error. It is undone and the pieces are attempted to be recognized individually. So far the WinBank system has no method of merging a fragment with a different piece of the image, although such an improvement should be feasible. Suggestions for its addition can be found in Section 7.

This strategy is similar to the approach used by Congedo et. al. [2] and by Dimauro et. al. [3] because it applies multiple segmentation algorithms and retries segmentation when it fails. However, it reduces the amount of recognition required because it relies on structural heuristics to perform a quick guideline classification to determine the plan of action rather than blindly applying recognition and proceeding based on the result.

This strategy could also be compared to the approach of Lee et. al. [5], where a graph representing combinations of candidate cuts is built and an optimum path through that graph is found. Rather than using recognition certainties to rate certain combinations as in [5], the new system selects from its available choices using a set of heuristics. A major difference is that new options can be built dynamically, so the problem is not constrained to a set of cuts determined at the beginning of processing.

There are several details which are critical to successfully applying this method. The first is that the classification heuristics should yield the best possible classification for a character. Since classifications may be re-considered to choose new courses of action, it is not necessary that the correct classification is made at every iteration. However, incorrect classifications will lead to inefficient operation. For example, a single digit that is classified as multiple touching digits will be fruitlessly separated. Touching digits that are classified as a single digit will not be separated when they should be. Even though the correct classification may be made later in the feedback process, each incorrect classification results in wasted computation.

Another important detail is the number of candidate cuts which are considered and retained when a character is being separated. Since different segmentation algorithms may yield different cut paths, applying more algorithms makes it more likely that the correct path is generated. Using more methods of separation could thus make the system more reliable, subject to the observation that each algorithm applied adds computation overhead. However, it is also possible that the paths generated are very similar to each other and add little improvement. The number of paths which are actually used should be trimmed to a low number, such as three (the number currently used in the WinBank system), so that too much recursion does not occur when characters are very difficult to separate correctly or should not have been separated at all.

Since not all paths are attempted, it becomes more important that the best cut is chosen when performing separation. This means that the paths being considered should be properly ranked. The smaller the number of paths used, the more critical it becomes to correctly rank those paths. Therefore the path ranking strategy becomes an important part of the algorithm. This ranking is done based on a number of structurally-based heuristics, such as the number of times a path cuts across writing and the detection of corners along the path.

One final detail is based on the fact that occasionally correct segmentation may be nearly impossible, and also that sometimes recognition may fail even on properly segmented characters. Such cases will result in fruitless attempts to re-segment pieces of the image. Therefore a reasonable limit should be placed on the amount of iteration through the feedback process. Allowing only two iterations through the loop may cause correct segmentations to be missed, but allowing many iterations could result in needless amounts of processing to no avail.

Performing classification and accumulating several candidate segmentations at once loses the simple sequential segment-and-recognize process in the approach used in [2] and [3]. But this loss results in a gain by avoiding unnecessary attempts at recognizing badly-segmented characters. Overall, the proposed feedback loop improves the accuracy of the existing WinBank system. It also demonstrates a new approach to feedback between segmentation and recognition that minimizes the overhead of performing unnecessary recognition before satisfactory results are obtained.

One drawback to this approach is that this feedback system assumes that with high accuracy the recognition module does "the right thing." It does not wrongly identify digits; it either correctly recognizes them or rejects them as unrecognizable. Ideally, the recognizer would also never reject correctly segmented digits. The rejection rate has little impact on the correctness of the feedback algorithm, but a high rejection rate would result in inefficient operation because separations and merges would be tried and retried unnecessarily. It is therefore assumed the rejection rate is reasonable (for example, 10% of properly-segmented characters being rejected, which is comparable to current state-of-the-art systems [6]) but this statistic is not rigidly specified.

4.2 Region Classification

The purpose of region classification is to determine whether a group of pixels represents a single character, a fragment of a character, or multiple characters which are touching. The classifier can also categorize a region as a symbol (a period or comma) or as a nonsense fragment which should be discarded. These classifications are used by the segmentor as guidelines for the operations to be performed on the regions. Since these classifications are based on simple heuristics and subsequently are inaccurate in some cases, they are reconsidered during feedback and modifications made to regions may be undone or redone. The region classifier evolved from previous work on the WinBank system but has been expanded to make several new classifications.

The classifications are made by stepping through the following rules:

1. *If the region is too small to be useful, discard it.* "Size" is judged by the region's *weight*, or the number of pixels it is composed of. The region is "too small" if its weight is below a given threshold, which varies with the size of the image.
2. *If the aspect ratio is low enough for the region to be a horizontal line, then discard it.* The "aspect ratio" is the ratio of the region's height to its width. The threshold aspect ratio for horizontal lines is set to 0.25. This value was determined through experimentation and was meant to be conservative so that important parts of the image are not discarded.
3. *If the region is small and it is near the bottom of the image, it is a comma or period.* A second weight threshold is used to determine if the region is "small." This threshold weight also varies with the size of the image. The region is near the bottom of the image if it is completely below the vertical midline of the text in the image, which is the height halfway between the highest peak and the lowest point. Note that because of this rule, no distinction is made between commas and periods. All such punctuation was arbitrarily chosen to be designated as commas.
4. *If the region is small and it is not near the bottom of the image, it is a fragment of a character.* The weight threshold used here is the same as that used in step 3. The difference is that at least part of the region is above the vertical midline. Because fragments below the midline will be classified as commas, they must be checked later

to see if they should be reclassified as fragments and merged to their neighbors to create whole digits.

5. *If the region does not cross the vertical midline, it is a fragment of a character.* The weight of the region does not matter in this case because small fragments are caught in step 4. The vertical midline is determined by the position of the whole set of characters, so any pieces which do not cross the midline are of a much smaller height than the rest of the characters.
6. *If the aspect ratio is too low, the region is composed of multiple characters.* All regions with aspect ratios below 0.65 are classified as multiple characters. Again, this aspect ratio was determined through experimentation and was intended to be conservative. If multiple characters are missed due to having an aspect ratio slightly above the threshold, they will be discovered during the feedback process.
7. *Otherwise, it is a single character.* If all of the above tests are passed, the region is classified as a single character.

These rules are guidelines, but they are not always accurate. The classification is not taken for granted to be correct. Instead, in several cases they are double-checked by further testing when the regions are being processed. Also, classifications of characters that are rejected by the recognizer are re-evaluated by the feedback module. This re-evaluation is detailed in Section 4.5.

One notable mis-classification which can occur is that fragments at the bottom of characters will be classified as commas in step 3. Regions classified as commas must be checked later to see if they could reasonably be combined with nearby regions to make a single character. The criteria for merging are beyond the scope of this section but are described in Section 6.5. If the region should be merged with its neighbors, it will be re-classified as a fragment to be processed accordingly. Obviously, care must be taken to avoid merging a comma with a character, so this change is made conservatively.

Another possible error is that commas which are drawn high enough to cross the vertical midline will be wrongly classified as fragments of characters in step 4. Again, checks are made to see if combining the fragment with nearby regions is reasonable. Currently no mechanism is in place to re-classify the fragment as a comma.

4.3 Path Evaluation

When a region is classified as consisting of multiple characters which are touching, it must be divided into its constituent parts. Several segmentation algorithms are used to find paths which divide the region into two parts. The different algorithms build different paths, and which of these paths is best varies depending on the case at hand. Ideally, the best path should be chosen from those available. In the iterated segmentation and recognition systems described in Section 3.1, the algorithms were applied one at a time in a predetermined order. Thus, those systems do nothing to compensate for the fact that their top-choice algorithm will not always produce the best segmentation path.

The WinBank system applies all segmentation algorithms at the beginning of separation and then chooses the path which has the best chance of being correct. As a result, segmentation is usually done correctly the first time, at the cost of extra processing to apply the available segmentation algorithms at the beginning. However, this processing is outweighed by the performance gained by preventing extra passes through the feedback loop and therefore through the recognition module. The choice of paths simply has to be better than the pre-determined choice imposed in iterated segmentation and recognition systems for this system to outperform them.

Some method of evaluating the paths is necessary to determine the most likely candidate. The paths which are generated by the different segmentation algorithms are ranked based on their likelihood of being correct. These judgments are made by scoring the paths according to several heuristic rules. Each path is initially assigned an integer score of 0, and points are added or subtracted for good or bad features of that path and their respective importance to the path's correctness. The number of points assigned for different cases was chosen as a part of defining the path evaluation heuristics.

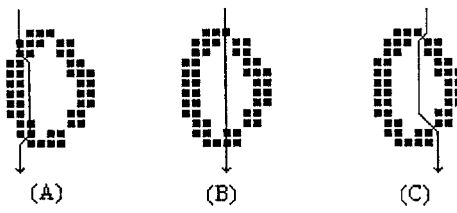


Figure 5: Detection of cuts. Each of these paths cuts through two lines. **(A)** A transition from black to white or from white to black along a path marks a cut. **(B)** Black pixels at the very top and bottom rows of the region are counted as color transitions as well, because the region is assumed to be surrounded by white space. **(C)** A cut is also formed when the path passes along white pixels diagonally between two black pixels.

The most important characteristic used to judge paths is the number of cuts made through characters. Cuts can be detected by examining the color changes along the paths. If the color of the pixels changes from white to black when traveling along the path, that transition marks an initial point of a cut. If the path transitions from black to white, a final point has been found. Figure 5 illustrates the three different cases used to detect cut points.

An important fact to note is that most of the time the proper path makes exactly one cut. If a path makes no cuts, it breaks the region into pieces which were already separate, or it does not break the region at all. Multiple cuts are rarely needed, because it is unlikely that two digits will touch in more than one place. Therefore the score of a path is unchanged if one cut is made and reduced by two points for every cut above the first one. The path score is also unchanged if no cuts are made. This decision was made because of the properties of the interaction between the different types of scores assigned, and is discussed below.

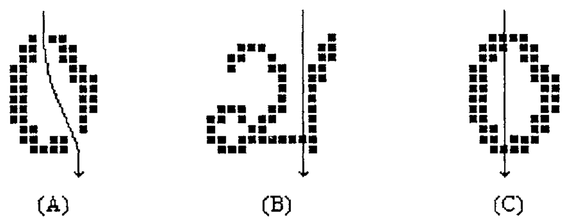


Figure 6: Number of cuts. **(A)** A path that makes no cuts simply breaks already-separate pieces. **(B)** Most common is the case when a path cuts a single line. **(C)** A path which cuts through more than one line is most likely cutting into a digit.

When two digits are written so that they are touching, the junction of the two digits is almost always characterized by two corners. The possible junctions between characters are illustrated in Figure 7. In very rare cases will the joint between two characters be smooth. Following this observation, the color transition points of a path are analyzed to determine if they occur at corners of the contour. The requirement that cuts be made at corners is restricted further to ensure that cuts cannot pass through convex corners, or portions of the digit which jut out into whitespace.

The path score is increased by one point for every concave corner encountered and reduced by one point for every cut across a relatively straight line. It is very unlikely that a cut point chosen when building the path lies at a convex corner, but to cover this possibility the path score is reduced by three points whenever a cut point at a convex corner is found, because a path cutting at such a point is unlikely to be correct.

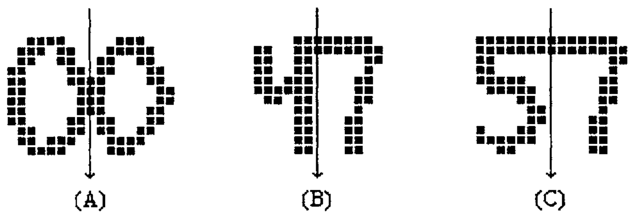


Figure 7: Path corner cases. **(A)** The most common case of two concave corners. **(B)** Less common is a path with only one concave corner. **(C)** Smooth junctions are very unlikely.

Because the score of a path is modified for every time it cuts across digits, the score of a path which makes no cuts is unchanged by corner analysis. This means that the relationship between the score of a path with no cuts and those of the other paths built for a segment may change disproportionately. For instance, when the number of cuts is used to modify the path score, a path which makes no cuts will be left unchanged, as will a path which makes one cut. However, when the angles of cuts is considered, the path which makes no cuts will be unchanged but the score of path which makes one cut may be increased twice if it cuts at two concave corners. Its score will be decreased if it cuts at relatively flat areas. Therefore the distinction between single- and trivial-cut paths really made when considering other criteria.

Another criterion which is examined is the length of the cut which is made. In the case of multiple cuts, the length considered is the distance between the start of the first cut and the end of the last one. If the cut length is too great, the path score is decreased by two points. This heuristic is used because touching characters contact each other in small junctions rather than long ones.

Algorithm-specific observations can also be made about the paths. For instance, paths generated by the top-left and top-right drop-falling algorithms are likely to make poor cuts in the lower half of the image. Similarly, cuts made by the bottom-left and bottom-right drop-falling algorithms in the upper half of the image are likely to be wrong. To allow algorithm-specific adjustments of the path scores without requiring the main segmentation module to know anything about those algorithms, the score for each path is adjusted as necessary within the path generator before the path is released to the main segmentation module. In the drop-fall path generator the segment scores are decremented by one point for cases such as those mentioned above.

The scores can also be used to bias the program toward one segmentation algorithm or another. If one path generator tends to be more accurate than the others, the likelihood of choosing the paths it creates can be improved by automatically incrementing the scores for those paths after they are generated. No such bias is currently used in the WinBank system.

In a similar way, the order in which the segmentation algorithms are applied to the image can weakly bias the choice of paths. If two paths have the same score, the first of the two which was generated is chosen first.

Once path scores have been assigned, the best path is chosen and used to divide the image into two separate segments. But the other paths are not discarded. They are ordered from best to worst and retained for possible future use. Saving the paths which have not been tried facilitates easy backtracking. If the chosen path fails to divide the image into recognizable regions, its results can be discarded and the next-best path used to make a second try. Retaining alternative paths removes the need to generate them again if separation is redone. More details of this corrective process are given in Section 4.5.

One possible improvement which could be made to this ranking strategy is the elimination of similar paths. It is possible that two different segmentation algorithms would generate paths which are nearly the same or even identical. This improvement is discussed in Section 7.

4.4 Representation and Replacement Strategy

Once all connected regions have been classified, the proper course of action is taken for each. Fragments of characters are merged with their nearest neighbor. Paths are generated for multiple-character regions, which are then divided into two parts along the most likely path. The results of these operations are then passed to the recognizer. If the image was segmented properly on the first try, all of the digits should be recognized and the process is complete. However, further processing must be done on digits which were not recognized. A mechanism for efficiently backtracking and taking new actions must be established. The operations of separation, merging, undoing separation, and undoing merging can lead to a complex series of interactions between the pieces of the image. Therefore the representation used to manage these pieces is crucial for correct and efficient operation. This section describes the data representation used to implement the feedback process.

Each segmented piece of the image can be considered to be a *block*. When connected regions of pixels are initially identified, each is treated as a single segment and stored in a single block. However, blocks can hold more than one region of pixels. If a region is classified as containing multiple digits, the two pieces resulting from its segmentation are placed into new subblocks within its block. If two regions are classified as fragments and merged together, a new block is created to store their merged result, and the parent regions are placed as subblocks within the new block. Figure 8 illustrates the concept of blocks. (See the discussion below to understand the boxes depicted within these blocks.)

Since the same block structure is used to represent these three different cases, each block is given a specified type. This allows us to determine whether the subblocks of a block are the result of a merge or a separation. The types possible are MERGE, SEPARATE, UNKNOWN or HOLD. Blocks which represent a single piece of the image are of type UNKNOWN until they are recognized, because they may need to be merged or separated in the future. Blocks which represent recognized segments are changed to type HOLD to prevent further processing on them.

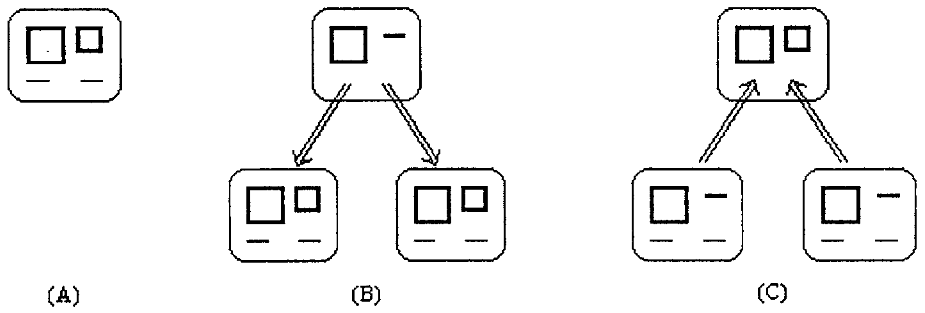


Figure 8: The three types of blocks. **(A)** A simple block containing a single digit has no subblocks. **(B)** A region which was classified as "multiple" and divided into two new pieces. The original region is stored in the original block. The new regions are stored in the two subblocks. **(C)** Two regions which were classified as "fragment" and merged into one new region. The original regions are stored as subblocks in the new block.

The smaller boxes depicted within these blocks represent the image data being stored. Each block contains one region of pixels taken from the original image (represented in these diagrams as a large box on the left). No modifications are made to this piece of the image. When a region is segmented into two pieces as in Figure 8(B), the resulting pieces of that region are copied and stored in the new subblocks, one in each block. The original piece of the image is still kept in the original block so that the segmentation can easily be undone by deleting the two subblocks. When two regions are merged as in Figure 8(C), they are copied into a new region inside a new block. Undoing the merge is then simply a matter of removing those subblocks from the newer block and placing them back into the list.

The small boxes drawn to the right within the blocks in the figure are a result of the fact that the WinBank recognizer requires normalized images of a specific size. During normalization, slant correction, line thickness correction and scaling are performed, so normalizing a piece of the image cannot be accurately undone. Therefore, when a segment is passed to the recognizer, a new normalized version of it is constructed and stored within the block. That way it is easy to revert to the original version if it is not recognized and further processing is necessary. If a region has never been passed to the recognizer, its block contains only the original region. If a normalized region is passed to the recognizer and rejected, there is no reason to continue storing that normalized version, so it is destroyed.

On the outside, the separate pieces of the image appear to be stored as a linked list of blocks. An iterator is provided which returns these pieces in order. However, behind the scenes, the treelike block structure results in a more complex representation than a linked list. The connected regions taken from the image are originally stored in separate blocks which form a simple linked list. As these blocks are merged and separated, the list becomes a list of trees. The iterator traverses this tree to return only the regions being operated on, which are the lowest leaves of the tree. Figure 9 illustrates an example representation which may be built and the "virtual" linked list it simulates.

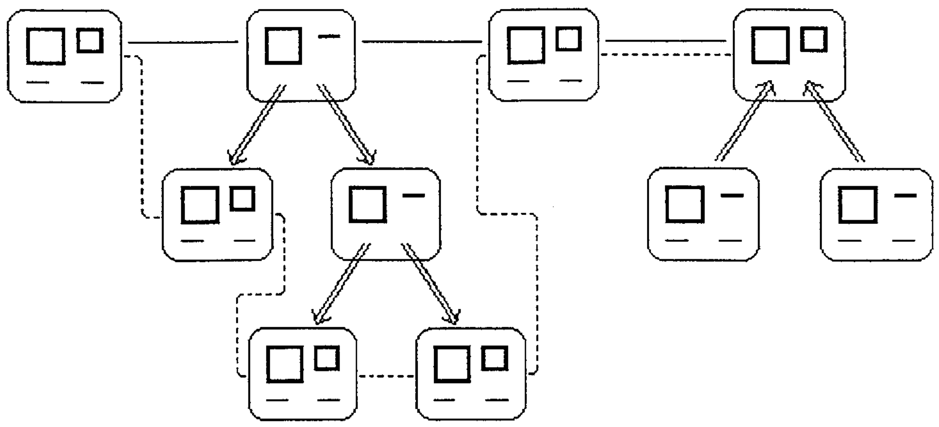


Figure 9: The list of block trees simulates a flat linked list. The solid line indicates the top-level list of blocks. The dotted line connects the blocks which are returned by the iterator. In this manner the list of four block trees simulates a flat list of six blocks. (See Figure 11 for an example where this structure would occur.)

When the image held within a block is recognized, the extra data associated with that block is destroyed. For instance, a block containing a single digit no longer needs to store the region of pixels taken from the original image, because its normalized version has been recognized. That normalized version is retained and the original version is destroyed. Similarly, if the two regions created by separating another region are recognized, the tree created by that separation is flattened. The original block is destroyed along with the un-normalized images stored inside the newer blocks, because the final result is composed of the normalized images in the new blocks. Finally, if a region which was created by merging two other regions is recognized, those two blocks are destroyed and only the normalized version of the merged region is retained. Figure 10 illustrates the way the tree is modified once certain regions are recognized.

Cases where one or more of these regions are rejected by the recognizer are handled by returning the blocks to their original state and then performing new operations on them. This correction process is described in the next section.

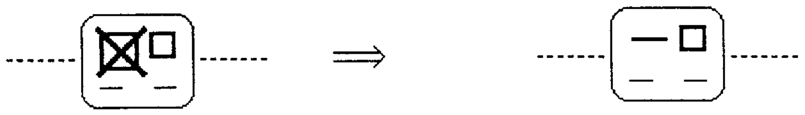
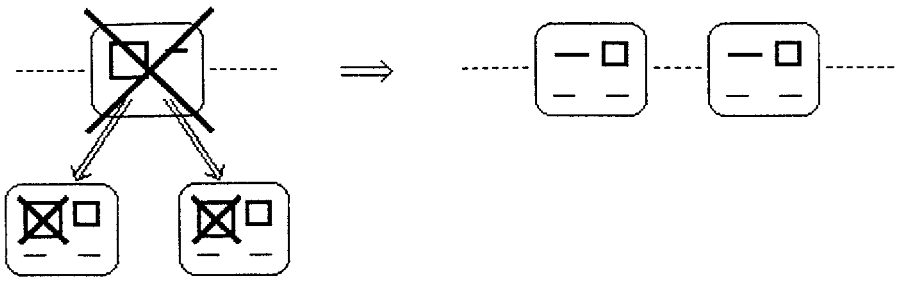
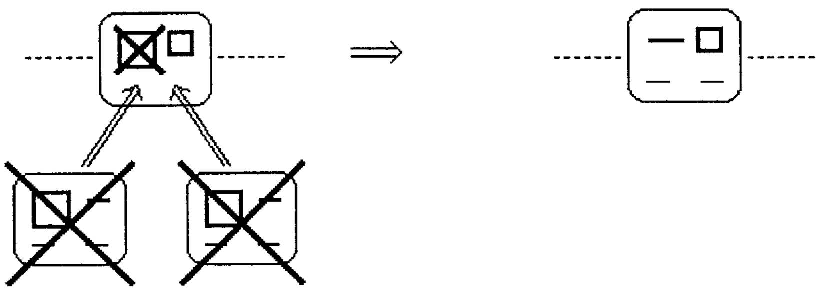


Figure 10: **(A)** When a single block is recognized, its un-normalized image is discarded and the normalized version is retained. The dotted line shown represents the block list.



(B) When the two blocks created by separating a single block are recognized, the original block is discarded and the newer blocks replace it in the list. Their un-normalized images are also discarded.



(C) When a block created by merging two blocks is recognized, the two original blocks are discarded, along with the un-normalized image.

The following example illustrates the process of constructing such a representation with many digits. The image shown in Figure 11(A) originally contains five separate regions of pixels. Two of these are single digits, one is made up of three touching digits, and two of the pieces are fragments of digits.

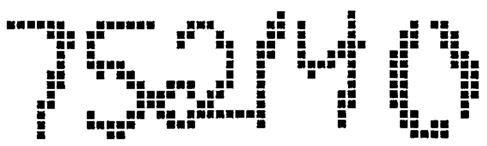
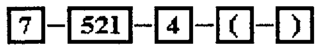
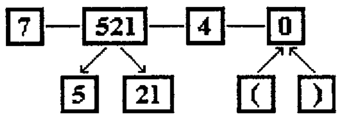


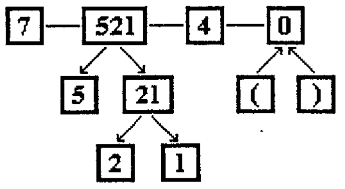
Figure 11: Example of operation. **(A)** The source image. Notice that the 5, 2 and 1 are connected and that the 0 is fragmented into two pieces.



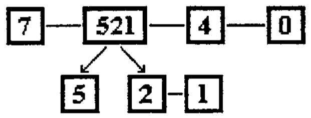
(B) The initial list of connected regions obtained from the image. The numbers shown in the figure represent the image pieces, and the boxes are the blocks containing them.



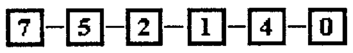
(C) The 521 block is classified as "multiple" and separated into two new pieces. The pieces of the 0 are classified as "fragment" and merged into a new block, which is inserted into the list in their place.



(D) The 21 block is classified as "multiple" and separated into two new pieces.



(E) When the segmentation and merging are complete, the resulting images are passed to the recognizer. Once regions are recognized, the trees are collapsed. This shows an intermediate stage where the fragments of the 0 have been removed and the 21 block has been flattened to a linked list.



(F) Finally, all intermediate blocks are destroyed and the tree is flattened into a straight linked list.

The list-of-trees representation allows separations, merges, and flattenings due to recognition to occur in any order. The complexity of the tree structure is hidden by the iterator, so that at all times the set of image pieces appears to be a linked list. Storing old information in the tree allows backtracking to be done easily.

The storage of two segments and two blocks within each block could become memory-intensive, but it saves a great deal of time when backing up to undo an operation. In addition, several facts limit the amount of information stored:

- The number of blocks being dealt with is usually small. If ten connected regions are found within an image, and two are merged into one segment and another is broken into two segments, at most 13 blocks are in use at any time. While the number of blocks is potentially limitless, realistically the number of blocks in existence at any time is on the order of a dozen.
- The extra information within a block only needs to be stored until the segment stored inside the block is passed through the recognizer. At that point it is either accepted, in which case all information except the normalized segment can be discarded, or it is rejected, so that the state of the block should be returned to how it was before segmentation or merging occurred.
- Extra blocks and segments are only created while performing merges or separations. The likelihood of needing to perform one of these operations is low, because the majority of digits found on checks require no segmentation. If the binarization of the image was done well, fragmentation is minimal and merging is only necessary in the case of handwritten disconnected digits, such as the character five with a disconnected top stroke. The other digits are very unlikely to be written as disconnected pieces,

due to the nature of their shapes. Separation is unlikely because a fairly low percentage of digits are written so that they are connected [18].

4.5 Correction Strategy

After the first pass through segmentation and recognition, most of the regions will be recognized but some may not. The operations performed on those regions may need to be redone to obtain correct results. This section describes the method used to reconsider the operations performed.

If a region was not recognized because it was improperly segmented, this improper processing may have resulted from one of three types of problems. First, the proper strategy may have been chosen but the result may have not been optimal. A fragment of a digit may have been merged to the wrong neighbor, or a region may have been separated along an improper cut path. Second, an action may have been made when the proper strategy would have been to do nothing: a segment may have been separated when it should have been left intact, or two segments may have been merged when they should have been left separate. Finally, an action may not have been taken when it should have been: two segments may have been left separate when they should have been merged, or a segment may have been left intact when it should have been separated.

As described in detail in previous sections, the case in which a segment was separated along an incorrect path is handled by storing alternative paths and trying them in the following iterations.

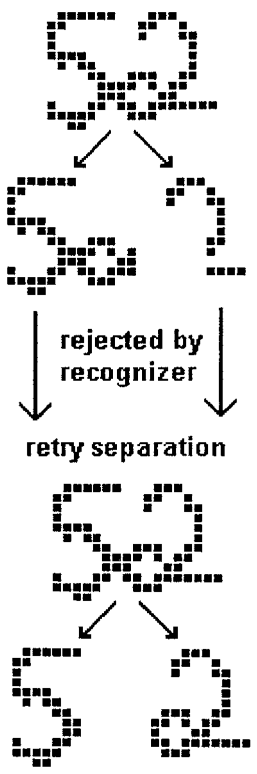


Figure 12: An example of a separation mistake being repaired by trying a new path.

The case in which a fragment was merged to the wrong neighbor is not currently handled by the WinBank system. In such a situation, the fragment which failed to be merged properly is attempted to be recognized alone and is finally rejected by the system. In this case human intervention is required. This case is infrequent but should eventually be corrected; see Section 7 for a discussion of this possible improvement.

The case in which a segment was separated when it should not have been is handled by eventually reaching a state in which the original image is used as a candidate for recognition. Separation paths are tried until they are exhausted, and then the non-separated image is passed to the recognition module. Also, on the last iteration the original image is used for recognition, even when all paths have not yet been exhausted. This ensures that the original character is always passed to the recognition module eventually. Because it will take several iterations to reach the correct state in this case, the classification of touching digits is done conservatively.

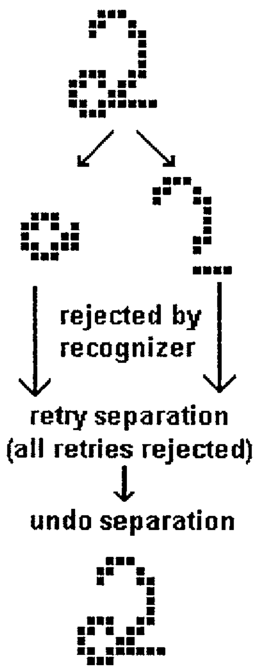


Figure 13: An example of recovery when separation should not have been done at all. This case is eventually handled when no successful separations are found.

The case in which a segment was merged when it should not have been is handled by undoing merging if the resulting image is not recognized. The two original pieces are then passed to the recognition module separately.

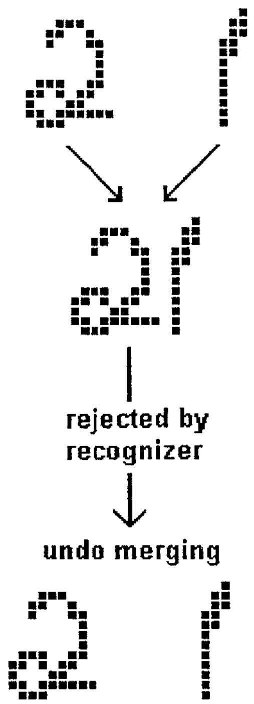


Figure 14: The case of merging when no action should have been taken is repaired by undoing the operation after the merged image is rejected.

The case in which no operation was done on a segment when it should have been separated into two pieces is covered by "relaxing" the classifications of digits. Each time a digit is re-examined, the likelihood of it being separated is increased. This is accomplished by increasing the threshold aspect ratio used to determine if a character is made of two touching digits each time it is re-examined, making separation more likely. A maximum aspect ratio is used as a limit to prevent splitting digits which are absolutely not connected. This value was chosen to be the reciprocal of the original aspect ratio, which means that a digit must be fairly tall with respect to its width to avoid eventual separation. However, since several iterations should pass before reaching this limit, only digits which were repeatedly rejected will reach this case.

The increase used to modify the aspect ratio which defines multiple digits is performed by multiplying the aspect ratio used in the first examination by 1.2 for each subsequent examination. This incrementing strategy was chosen based on experimentation. Multiplying by this number changes the aspect ratio considered by a small amount at first and increases the change with each iteration. It also allows four retries before the upper limit is reached, because the initial aspect ratio of 0.65 can be multiplied by 1.2 four times before it reaches its reciprocal, 1.54.

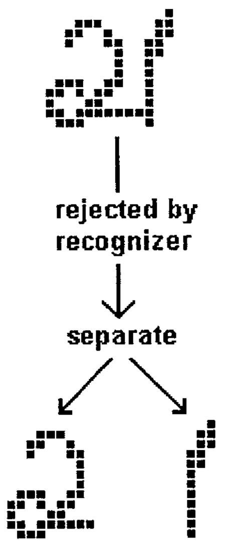


Figure 15: Touching digits which were not separated is handled by relaxing the segment classification until it is separated.

The case in which no operation was done on a segment when it should have been merged with a neighbor is currently not handled by the WinBank program. The difficulty in this case is that it is indistinguishable from the case in which a digit is properly segmented but is simply rejected by the recognition module. Correctly handling this case is described as a possible improvement in Section 7.

4.6 Retry Depth vs. Performance

If at least one segment is unrecognized after a pass, the program can loop back and attempt to do more processing on the unrecognized segment(s). Each such iteration adds more computation. However, the amount of computation necessary drops with each character recognized, because that character no longer needs to be considered. Assuming the recognizer is accurate and there are relatively few segmentation operations which must be done, most of the digits will be recognized on the first iteration. As a result, significantly less computation must be done on the second iteration than the first, and so on for each loop in which at least one segment is recognized.

However, looping still adds computation. The heuristics used are not always accurate, but they ensure that the most likely alternatives are explored. So it is arguable that if the actions taken in the first few iterations do not have favorable results, most likely the unrecognized segment being processed is a degenerate case which is so difficult to process correctly that it should be rejected. If the feedback loop is allowed to continue, finding the correct result may require many iterations, or the program may settle on an incorrect result.

Therefore a limit on the number of times the program passes through the feedback loop must be chosen to maximize results while still maintaining reasonable performance. WinBank currently loops a maximum of 5 times. This value was chosen by experimentation. It allows several attempts at segmentation to be made without reaching the point that these attempts would lead to incorrect results.

5 Performance Evaluation

The performance of the system is not easy to characterize because a number of factors interact in complex ways. For instance, because the feedback process relies on the recognition module to reject with high probability when a character was poorly segmented, recognition errors can contribute to segmentation errors. As an example, touching digits which were not separated and then were subsequently falsely recognized as a single digit are difficult to classify as problems with the segmentation, recognition or feedback portions of the system.

Also, several different criteria could be used to judge the performance of the feedback system. The number of times an erroneous operation was performed or a correct operation failed to perform are examples. Other benchmarks of the feedback process are the number of times correct paths were chosen on the first try or the number of times separation had to be redone. Even in this case it is difficult to pin down the source of errors, because it is possible that the segments being separated were touching in a way that the segmentation algorithms generated no correct paths. In such a scenario the feedback system would continue to try separation paths even though none will be successful.

To avoid the complex interactions between the segmentation and recognition which occur during feedback, the sample of check digits was recognized by manually identifying or rejecting each digit. This ensured that all correctly-segmented digits were recognized and all badly-segmented digits were rejected. Though these conditions would not be present in real-world operation of the system, the wide variety of recognition methods which could be used with these segmented digits are all designed to minimize false recognition, so that even in real-world systems nearly all badly-segmented digits would be rejected. However, failed recognition is a possibility which could lead to increased processing. A digit which is properly segmented but is not recognized would continue to be passed through the feedback process and could possibly be further separated in an attempt to correct the perceived error. However, as just mentioned, the possible erroneous segmentations resulting from such further processing would introduce few errors. These errors would cause a computational increase because the digits would be reconsidered needlessly.

Rather than going in-depth into such complex details to extract performance figures, more intuitive benchmarks of the process have been accumulated. These describe the overall performance of the system but do not describe particular details such as whether errors were due to problems with segmentation or with merging, or whether they originated from cases of performing operations badly or not performing them at all.

Information about the input set, such as the frequency with which touching digits appeared, explains the circumstances in which these operations were being carried out. The checks used had a particularly high frequency of touching digits. This is due to the fact that the program works with totally unconstrained strings of handwritten numerals. Also, the set of checks used was produced for testing purposes and was written with the intention of being somewhat difficult to handle. More common samples of check digits have approximately 86% of their digits unconnected [18]. Of the limited sample used to test this feedback system, only 74% of the digits were unconnected. This difference would translate to increasing the number of segmentation errors encountered.

The criteria used to evaluate the actions of the feedback system are simply the number of segmentation errors which were present after the first pass of segmentation and the number of errors present after the feedback loop has run its course. These factors give a simple understanding of the improvement added by the feedback system while avoiding complex issues of interactions. The average number of iterations required for the system to run its course was also determined. This lends insight into the number of retries which must be attempted to reach the final result. Table 1 describes the results accumulated from a limited sample of Brazilian checks.

Number	**TOTAL**	**% Digits**
Total checks	40	
Total digits	225	100%
Singular digits	167	74%
Connected pairs	26	23%
Three or more connected digits	2	3%
Segmentation errors before feedback	40	
Errors after feedback	20	
Average number of iterations	3.1	

Table 1: Input characteristics and feedback improvments.

As can be seen from the table, the feedback process reduced the number of segmentation errors by half, taking the segmentation accuracy from 82% to 91%. As mentioned before, these errors might have been generated from a number of circumstances, including unreadable handwriting, failures to perform separation or merging, or mistakes made during separation or merging. The relatively high error rate can be mainly attributed to the malicious nature of the origin of the checks. Many digits were written in a highly overlapped fashion such that segmentation is difficult or impossible to perform correctly.

The results above can be translated directly to improvements in the recognition rate of the check reading system. Since performing this feedback process will introduce few new errors, the ultimate result is that digits which would otherwise have been rejected will now be recognized.

6 Other Improvements Made to the WinBank System

Although the feedback loop between the recognizer and the segmentor was the major improvement made to the WinBank system, other changes were made to the system to improve its accuracy and performance, as well as to make the code cleaner and more modular. As mentioned previously, the system was completely re-implemented, so several of these changes have been made simply as consequences of the re-implementation.

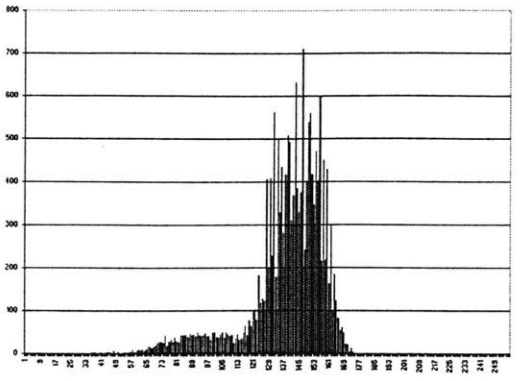
The previous version of the WinBank system was written in C with a Borland C++ user interface. It has been moved to Microsoft Visual C++. This process was not a simple port. The system was completely rewritten to make use of improved data structures. Another goal was to make the code more modular, so that it could eventually be made packaged as a handwriting recognition system that could be easily modified by other parties and adapted to particular needs. The majority of the algorithms that were implemented are the same as in the previous system. Exceptions are noted below in the following sections. The only part of the old WinBank system that has not been implemented is the "segmentation critic," a parsing module which was largely responsible for ensuring that the cents portion of the written figure is correctly segmented. The segmentation critic will eventually be added to the system again to improve performance.

6.1 Image Binarization

Image binarization involves extracting the important information from a grayscale image. The only information which is desired is the writing; the background should be ignored. The use of a black-and-white representation also greatly simplifies processing in the rest of the program, which does not have to deal with the further complexity of processing grayscale images. Therefore the image is converted from grayscale to a black-and-white representation where black pixels represent the writing to be analyzed. To separate black from white, a *threshold* color is chosen such that all colors below the threshold are become black and all colors above the threshold become white. In this manner the image is converted to a binary 0-1 representation where 0 represents white and 1 represents black.

The threshold value may be hard-coded, but such a technique does not allow variation based on the properties of particular documents. Consider a case where the same threshold value is used for a dark-colored check as a light-colored one. The dark background of the dark-colored check might be converted to black, leading to a very noisy or even unreadable image. Similarly, a check in which the handwriting is light in color may become completely white. Therefore, *dynamic* thresholding is used. This means that the threshold color is chosen based on the colors prevalent in the image.

To choose a threshold value, the program scans across an area of the image and constructs a histogram of the colors found. See Figure 16 for an example of a typical check color histogram. The histogram is smoothed so that it represents trends rather than noisy spikes [12]. A typical check histogram has two peaks: one peak represents the prevalent background color, and a lower peak represents the average color of the writing. The appropriate threshold value is between these two peaks, dividing background from writing. The threshold is chosen to be between the minimum (darkest) color found and the average color (approximately the top of the background-color peak).


(A)

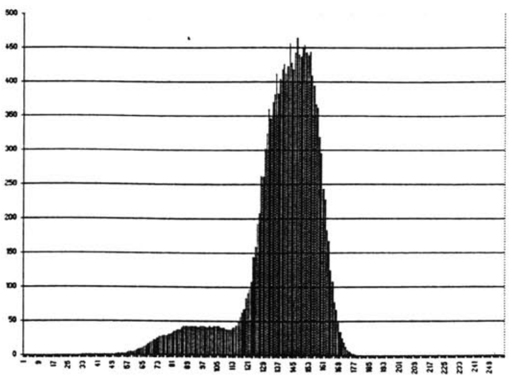

(B)

Figure 16: A typical check color histogram. The low region on the left represents darker writing while the higher region on the right represents the background fill color. The ideal threshold is between these two areas. **(A)** Unsmoothed. **(B)** After smoothing.

Simply choosing a threshold value is usually not enough, even if that threshold is chosen dynamically. The image may be of low quality, having low contrast and a certain amount of noise. Therefore a filtering process is carried out before thresholding to ensure that only the desired parts of the image are retained. Mean and median filtering were attempted, but with limited success because they tended to allow background noise to remain while causing writing to be thickened and blurred together.

The filtering algorithm currently in use is a selective median filter. The pixels of the image are considered individually, and each pixel's new value is chosen based on the range of colors in its immediate neighborhood. If a wide range of colors is found (an intersection of background and writing) the pixel is left as it is, but if a small range of colors is found (mostly background or mostly writing) the pixel is changed to be the median color of its neighborhood. The result of this selective filtering is that the area away from the writing is filtered, but the immediate vicinity of any writing is left intact. Therefore writing is not blurred but background noise is minimized. This filter has been much more successful than the other methods that were tried.

6.2 *Courtesy Amount Block Location*

Originally, the WinBank system was designed to operate on American checks. While the location of the courtesy amount block is not fully standardized in American checks, the area is isolated enough (and standardized enough) that a hard-coded region was sufficient for locating the written digits.

Brazilian checks are even less standardized when the location and size of the courtesy amount block are considered. The height, width, and location of the CAB vary between checks from different banks and even between checks within individual banks. Additionally, while most American courtesy amount blocks are indicated by closed boxes with lines on all four sides, Brazilian checks are delimited in a variety of ways, usually much more loosely. The CAB is usually indicated by lines along one or more sides, rather than being fully enclosed. Figure 17 shows a number of CAB styles common to Brazilian checks.

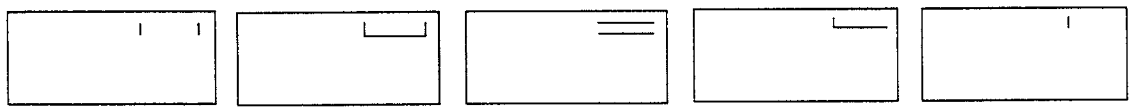


Figure 17: Common CAB styles in Brazilian checks.

Experimentation revealed that hard-coded courtesy amount block locations were not sufficient. Their use would lead to cutting off digits or including additional lines and digits that were actually parts of control codes printed nearby on the checks. In response to such variation, new functionality was added to the WinBank system to search for CAB indicators rather than to depend on hard-coded block locations. The program begins with a hard-coded estimate of the location in the image where the CAB may be, and searches for lines and other cues indicating its exact position.

A distinction is made between finding vertical and horizontal lines. It is much easier to correctly detect horizontal lines than to find vertical ones. It is much more likely that tall features of written characters will appear to be vertical lines than it is that any handwritten features will appear to be horizontal lines. Since Brazilian numbers (indeed, all major numbering systems in the world) are arranged horizontally, the presence or lack of whitespace in the horizontal direction can also be used as a cue to determine the location of the CAB. Less information is at hand when finding vertical lines, so a more careful analysis is made to detect vertical lines.

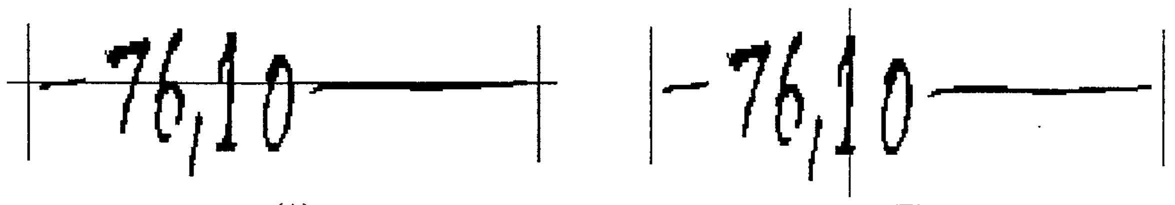


Figure 18: It is easier to find horizontal lines reliably than to find vertical ones. **(A)** Because digits are written horizontally, the amount of whitespace present in a row makes it easy to distinguish between horizontal CAB lines and human-drawn lines. **(B)** It is much more difficult to distinguish between vertical features of the handwriting and vertical CAB lines.

The horizontal organization of writing can be used to find the top and bottom of the CAB area, whether horizontal lines are present or not. If a horizontal starting line can be chosen so that it is situated within the writing in the CAB, the guess can be expanded above and below until whitespace is reached, thereby finding the top and bottom of the CAB. Care must be taken to make a wise guess about a starting position; otherwise the CAB may be missed or other pieces of the check may be mistaken for the CAB. Also the definition of whitespace must be made such that the top and bottom limits include all of the writing in the CAB. If too many dark pixels are allowed, the top or bottom of the detected CAB area may cut through writing which should be included. Such a strategy of looking for whitespace can also correctly detect failure when writing from other parts of the check projects into the CAB area. If such a case exists, it is unknown what the writing may be interpreted as and the check should be rejected to be examined by a human operator.

Since horizontal lines may be present in the CAB area as found above, the detected area is further restricted to exclude them. Horizontal lines are detected as rows with unusually high concentrations of dark pixels. To accommodate lines which are not quite horizontal, the search is also relaxed slightly to look for high concentrations in groupings of rows. To prevent such searching from finding "false" CAB lines in the image, such as long dashes drawn by the check writer, these lines are found conservatively.

Vertical CAB lines are much more difficult to detect accurately. Lines of the text may be darker and even taller than the lines of the CAB. Also, the right and left sides of the CAB cannot be found by the simple estimation-and-adjustment scheme described above for finding the top and bottom, because of gaps between digits. A vertical line can be detected by a method similar to that described for horizontal lines above. Columns which have many dark pixels in them are chosen as candidates for vertical lines, and these are tested to ensure that they are surrounded by whitespace. If a dark column is not surrounded by whitespace, it must be part of a digit, such as the back stroke of a 9. Searching continues until dark lines are found or the end of the hard-coded area estimate has been reached.

One problem which must still be worked out is that tall dark strokes surrounded by whitespace, such as the number 1, can occasionally be mistaken for CAB lines.

Heuristics such as the requirement that the vertical lines be reasonably close to the edges of the hard-coded CAB bounds could be applied to improve the algorithm, but further research is still required.

Once the lines in the image have been located, limits must be found for edges of the CAB that are not marked by lines. Top and bottom edges can be found fairly easily, as described above. The extremes of the left and right side can be detected when horizontal lines are present. This is done using the extremes of the known lines. For instance, if no vertical CAB lines are present but a horizontal line marks the bottom of the region, the right and left edges of the line can be used as bounds for the CAB. If no horizontal lines are present, only an estimate can be made using the initial bounds used to begin the search process.

The location of the courtesy amount block is an area of ongoing research. Even though the above CAB-finding technique is fairly successful, correct CAB location is critical enough to the proper interpretation of a check that its accuracy should still be boosted. Further improvements which could be made are described in Section 7.

6.3 Size Normalization Algorithms

Several normalization steps are performed after characters are segmented. The purpose of this normalization is to make characters as standardized as possible, so they look as much as possible like the other characters of the same type. For instance, ideally all instances of the number four would look alike. The first step in normalization is slant correction [13], followed by scaling. As a final step the line thicknesses are made uniform through a process of thinning and rethickening [14, 15]. The slant correction and line thickness normalization algorithms have been implemented in the new version of WinBank in the same manner as they previously were written. However, the scaling (or size normalization) algorithms were rewritten due to a lack of documentation of the old methods used.

Character size normalization is necessary because the recognition module requires 16x16 arrays as input, while there are no constraints on the size of the scanned image. Therefore the segmented characters must be made smaller or larger to fit the constrained size. The size reduction and enlargement methods used by WinBank were re-written from scratch to match the new data types and to improve code clarity.

The new enlargement algorithm is a straightforward reproduction of the space done by sampling the input image and multiplying pixels when necessary. The size reduction algorithm requires more description. The input image is converted to a smaller output image by mapping dark pixels in the input image to one or more pixels in the output image. This process can be visualized as overlaying the pixel grids of the two images and darkening the pixels in the new image that correspond to dark pixels in the old image. When a pixel from the input image overlaps one in the output image, the overlapped area of the output pixel is added to a total for that pixel. For example, 0.25 is added to the total for a new pixel if a quarter of its area corresponds to one dark pixel in the old image. All of the dark pixels in the input image are divided among the pixels of the output image. Each pixel in the output image accumulates a total between 0 and 1 representing the portion of its area which is overlapped by input pixels. If the total is sufficiently large, meaning enough of the area represented by that pixel is darkened in the input image, that pixel is darkened in the output image.

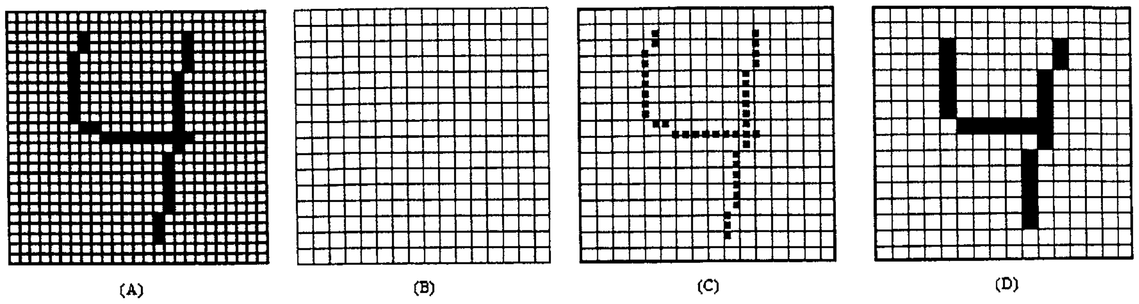


Figure 19: **(A)** The image to be reduced. **(B)** The 16x16 grid to be used in the final image. **(C)** The process of shrinking the image can be visualized as overlaying the smaller pixels of the original image onto the larger pixels of the new one. **(D)** Pixels are darkened in the new image if they are sufficiently filled.

The decision as to whether an accumulated total is large enough to darken a pixel must be made with care. Intuitively, if more than half of a pixel in the output image is darkened (i.e. if its total is greater than 0.5), it should be darkened. However, the darkening threshold must be scaled based on the scale factor by which the image is being reduced. If a 100x100 image is reduced to 16x16, its lines will be much thinner than a 20x20 image which is reduced to the same size. This is because pen widths are generally constant even though the characters are being written larger, so line widths are not proportional to the character size. Therefore the darkening threshold is reduced for larger input images in inverse-proportional manner.

This algorithm is straightforward and accurate, but its performance may not be as good as that which could be attained using linear algebra techniques. A suggested improvement for the future would be to investigate more efficient ways to accomplish this task. This algorithm was chosen for its straightforward implementation within time constraints. Since the performance depends on the size of the input image, and since the checks used so far have been consistently of reasonably limited size, the performance of this operation has not been a serious issue as of yet. However, to make the system more general in terms of input image size, performance improvements in this area would probably be worth consideration.

Another change which has been to the character normalization algorithms used by WinBank is that scaling can be done disproportionately. Rather than scaling both directions proportionally, the scale factor is evaluated differently in the vertical and horizontal directions, so that the resulting character exactly fills the area it is being scaled into. In this manner, greater amounts of information are retained for recognition. Since all western numerals are written so that they are taller than they are wide, they would otherwise not fill a square area completely. Most handwritten digits are larger than their normalized sizes (16x16 in the WinBank system), so scaling disproportionately retains extra information about them. Saving more information about the digits makes them easier to recognize.

6.4 Segmentation Algorithms

A number of segmentation algorithms are applied to separate connected digits into their distinct components. The previous version of WinBank applied an "min/max contour" splitting algorithm and a "hit-and-deflect" strategy to separate touching digits [1, 16]. The new system uses four types of "drop-falling" algorithms and retains a modified version of the "min/max contour" strategy. The "hit-and-deflect" algorithm may be added in the future as another alternative path generator. As mentioned above, all of these algorithms are used to generate a list candidate paths for separation, and the best path is then selected from the list. Adding new segmentation algorithms should improve the accuracy of segmentation. However, it will also add computation overhead, so additional path generators should only be added if they will make an appreciable difference in segmentation accuracy.

6.4.1 Contour Critical Point Algorithm

The contour critical point algorithm is an adaptation of the upper/lower contour algorithm [1, 16] employed previously by the WinBank system. This algorithm is based on the observation that in almost all cases where characters are touching, the junction between those characters forms two or more concave corners along the contour. Therefore, if the concave corner points of the image are identified, those can be searched to find the correct points to cut between.

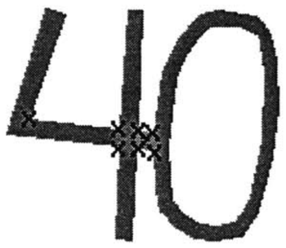


Figure 20: The set of concave corner points along the outside contours of the image almost always includes the points between which the cut should be made.

This algorithm begins at the (horizontal) center of the connected character and searches along its top and bottom contours for critical points. It moves outward from the center, searching all pairs of points until a suitable pair is found. The decision of whether a pair of points is suitable is made based on criteria such as their distance from each other, their distance from the center and the angle of the cut which would be made between them. Cuts which cross whitespace are also prohibited.

6.4.2 Hybrid Drop-Falling Algorithm

The hybrid drop-falling algorithm [2, 17] is based on the concept of a drop of water being placed at the top of a figure and rolling down along its contour. Wherever the drop of water is trapped in a corner, a cut is made downward through that part of the figure. The critical decisions which must be made during processing are the selection of the point at which the drop-fall begins and the rules governing the movement of the drop. Since the drop can fall in four different directions (for example, from the top-left toward the bottom-right, a *top-left* drop-fall), the algorithm comes in four different varieties. Cuts made in different directions may be better or worse than each other in different situations. The premise of the hybrid drop-falling algorithm is that all four strategies are applied, so this algorithm potentially outputs four candidate paths for separation.

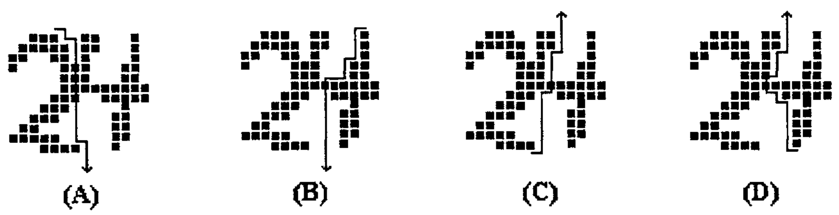


Figure 21: The hybrid drop-fall algorithm generates four paths, some of which are better than others. **(A)** In the top-left drop-fall the "drop" begins in the top left and rolls along the top contour toward the bottom right. **(B)** In the top-right drop-fall the drop travels to the left. **(C)** The drop falls upward in the bottom-left drop-fall algorithm. **(D)** The bottom-right drop-fall.

This algorithm was implemented exactly as in [17] with one exception. Rather than transposing the binary image array when performing top-right, bottom-left, and bottom-right drop falls, the direction of travel is changed in an equivalent way. Since copying pixels into a new order is a costly process, the directions of "up," "down," "left" and "right" are re-defined so that the direction of travel can be conceptualized as going from the top left down to the bottom right, as in the top-left drop-fall, while movement is actually in the appropriate direction. This makes it possible to avoid copying the array and allows code reuse so that the same code can be used for all four directions of travel.

6.5 Merging Algorithm

Merging a fragment to its most reasonable neighbor is done by evaluating which of its nearest neighbors is most suitable. The four nearest neighbors (two to the left and two to the right) of the fragment are chosen as candidates for merging with it. They are then evaluated based on proximity and factors such as overlap. If the fragment is above or below another piece of the image, it is more likely to be merged with it. This handles cases such as the disconnected top stroke of a 5, which may even be written closer to another digit. If the stroke overlaps the remaining base of the 5, it would be merged to that rather than to another nearby digit.

Since the components of the image are copied into new pieces and stored separately, their positions and relationships to each other in the original image are destroyed. But to properly maintain these relationships during merging requires that some knowledge of position be retained. Therefore, the coordinates of the original position of each segment are stored inside it. When pieces of the image are separated or merged together, the coordinates of the resulting segments are updated accordingly.

The merging algorithm has worked fairly well. However, since no alternative merges are considered after feedback, a fragment which has been merged to the wrong neighbor can only be rejected for human examination. Improvement of this behavior is discussed below in Section 7.

6.6 Syntax Verification

A new step which has been added to the WinBank system is the verification that the final output makes sense in terms of dollar amounts. For instance, if the value reported to be found on a check is "1,2,3", this is not a valid string in terms of money. Therefore simple syntax rules have been applied to ensure that the resulting value is reasonable.

The first step in the verification process is to check that the overall grouping of the characters in the string is correct. For instance, characters such as “$” may only appear at the beginning of the string. All characters except digits and punctuation (commas and periods) are then stripped away. Since commas and periods are so small, no distinction is made between them by the WinBank system, so they are all represented as commas in the program and in the following discussion.

The placement of commas and periods within the string is important for determining the amount of money in question. For instance, "1,000" and "10,00" have very different values. The next step ensures that the commas are consistent with each other and with the form of the string. Strings such as “1,00,00” and “1,0” are rejected in this step. Special cases such as leading commas and zeros are also rejected, because those do not occur when writing money.

The final step is that commas are added or when deemed necessary, because commas and periods are small enough that they may not have been found during image processing. In this case the string "1000" would become “10,00” and the string “1000,00” would become “1,000,00.” This change is purely aesthetic, but it makes the final value much easier for a human operator to interpret.

7 Possible Improvements

Due to the time required to fully re-implement the WinBank system, there are several improvements which could be made to the WinBank system as described in this document, but those improvements were considered and not implemented. Some pertain to the system as a whole, and some are specifically related to the feedback architecture which has been implemented. Those improvements are briefly discussed in this section.

It has been mentioned in previous sections that the CAB location, while fairly accurate, could still be improved. Vertical line detection is a particularly difficult task which could be improved. The CAB location method described above still has the weakness that vertical lines in the text which are surrounded by whitespace, such as the number 1, can be mistaken for CAB lines. The hard-coded limits of the area in which the CAB is searched for could be used with heuristics to require the lines to be fairly near the left and right edges of the region, but this too could be problematic. For instance, other vertical lines are also often present near the left edge of the courtesy amount block in Brazilian checks, so if the left line is not found, it is possible that the nearby (incorrect) line may be taken as the left edge of the CAB, leading to the inclusion of check character codes or other erroneous pieces of the image.

An important fact to note is that for all of the Brazilian CAB styles, there is always a line on the left or on the bottom. If no line is found on the left or the bottom, the hard-coded estimate used as a starting block was too strict. In this case the boundaries can be expanded to the left and bottom and the search for lines performed again. Such

CAB "seeking" could be added to the WinBank system if the hard-coded limits are found to be too inaccurate, but so far such an improvement has been unnecessary.

One area of research which could result in significant improvement of the classification process is the use of contextual information such as punctuation. For instance, if only two digits are found between commas, such as "D,DD,DD" (where D indicates a generic digit), it is likely that one of the two digits between the commas is actually made up of two touching digits. Discarded image pieces such as horizontal lines might also be useful for providing contextual information to use during segmentation. The WinBank system currently makes no use of the relationships between different characters to improve segmentation. This improvement has not been added to the system yet because it would be fairly complex to implement.

A small change which can be made to the re-classification done during the feedback process is the reconsideration of the classifications of commas and fragments. It is possible that commas would be classified as fragments of characters, or that fragments would be classified as commas. The mis-classification of commas and fragments has not been addressed in the existing feedback system because it is uncommon. Such reconsideration could especially benefit from the use of contextual information such as the position of other commas.

As mentioned previously, one possible improvement which could be made to the path ranking strategy used during segmentation is the elimination of similar paths. It is possible that two different segmentation algorithms would generate paths which are nearly the same or even identical. In the current system the paths would be ranked the same and applied sequentially. Repetitive computation could be reduced by removing similar paths so that only paths which have different results are considered. This improvement has not been implemented because of the complexity involved in judging whether two paths are nearly the same. They may differ in only a few places (they may even differ trivially as they travel through whitespace), but judging how different two paths are and how close two paths should be to each other to be considered similar is not a simple task.

Another improvement to the path ranking strategy would be the immediate exclusion of paths whose scores are too low. Currently the paths are ranked and used in order, but no minimum requirement is made on the score of the path used. It is possible that the segmentation algorithms do not generate many good paths, or even that they do not generate any good paths at all. A minimum score has not been imposed because the path ranking heuristics only result in fairly rough guesses, and because a fairly small range of scores is possible.

An improvement which could be made to the feedback process, as mentioned previously, is that it could also be used to correct erroneous merging. Currently, if a fragment is merged to the wrong neighbor, it cannot be recognized and is simply rejected. In the future, it should be possible to add feedback to redo merging. Care would have to be taken so that the fragment is not repeatedly merged to the same neighbor. Such an improvement could be modeled on the segmentation feedback process. The use of feedback to correct merging errors has not yet been implemented because the low probability that fragments are present has so far led to a low rate of merging errors.

One interesting observation of the feedback process is that when three or more digits are touching, the "higher" segmentations have less chances to be re-tried than the "lower" ones do. For instance, in Figure X three digits were touching. A 5, 2 and 1 were separated from each other. If the separation of the 5 from the touching 21 block was done incorrectly, it would not be retried until all separations of the 2 and 1 were tried. This means that the first separation becomes more critical to do correctly because it is less likely to be retried.

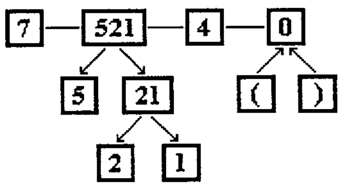


Figure 22: Repeated from Figure 11(D). A 5, 2 and 1 were touching in the original image. The segmentation which splits the 5 from the touching 21 becomes more critical to the process than the separation of the 2 and 1.

Indeed, if a piece of the 5 was attached to the 2 and 1, it is possible that the 2 and 1 could still be correctly segmented and recognized, leaving the piece of the 5 to be eventually merged back to the 5 it came from. In other words, several iterations would pass before the 5 was correctly segmented. This process could possibly be improved by limiting the number of times lower segmentations are redone before "higher" segmentations are reconsidered. In fact, in the case above, the separation of the 5 from the touching 21 block could be worth reconsideration immediately if the 5 is not recognized.

Another improvement that could be made to the WinBank system is modifying the character size reduction algorithm, as noted in Section 6.3 above. The current algorithm is conceptually easy to grasp but may be more computationally-intensive than techniques using linear algebra or other mathematical means.

8 Conclusion

The feedback system described here is a new technique for repairing segmentation errors. It was designed to be integrated with a segmentation-based approach to reading handwritten digits. It allows errors in segmentation to be reconsidered and repaired. The feedback system works with separation as well as merging operations, building a list of segmentation trees as the digits are being processed. This allows backtracking to be done quickly and easily. The feedback system improved the rate of recognition without increasing the rate of errors in the program. It also minimizes needless processing which is sometimes done in other segmentation-based systems. This new technique involves less processing than other previous segmentation-based techniques.

Since the WinBank check reading program was re-implemented based on a preexisting version of the system, some opportunities for improvements in other parts of the system were available. Most algorithms were re-implemented as they were, and any changes which were made have been explained.

Bibliography

[1] Sparks, P., Nagendraprasad, M.V., Gupta, A. (1992) An Algorithm for Segmenting Handwritten Numeral Strings. *Second International Conference on Automation, Robotics, and Computer Vision.* Singapore, Sept. 16-18, 1.1.1 - 1.1.4.

[2] Congedo, G., Dimauro, G., Impedovo, S., Pirlo, G. (1995) Segmentation of Numeric Strings. *Proceedings of the Third International Conference on Document Analysis and Recognition, Vol. II* 1038-1041.

[3] Dimauro, G., Impedovo, S., Pirlo, G., Salzo, A. (1997) Automatic Bankcheck Processing: A New Engineered System. *International Journal of Pattern Recognition and Artificial Intelligence* 11 (4) 467-504.

[4] Blumenstein, M. and Verma, S. (1998) A Neural Based Segmentation and Recognition Technique for Handwritten Words. *IEEE International Conference on Neural Networks* Vol. 3, 1738-1742.

[5] Lee, S.-W., Lee, D.-J., Park, H.-S. (1996) A New Methodology for Gray-Scale Character Segmentation and Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16 (10) 1045-1050.

[6] Kimura, F. and Shridhar, M. (1992) Segmentation-Recognition Algorithm for Zip Code Field Recognition. *Machine Vision and Applications* 5 (3) 199-210.

[7] Hong, T., Hull, J., Srihari, S. (1996) A Unified Approach Towards Text Recognition. *Proceedings of SPIE: The International Society for Optical Engineering* Vol. 2660 27-36.

[8] Martin, G. (1993) Centered-Object Integrated Segmentation and Recognition of Overlapping Handprinted Characters. *Neural Computation* 5, 419-429.

[9] Martin, G., Mosfeq, R., Pittman, J. (1993) Integrated Segmentation and Recognition Through Exhaustive Scans or Learned Saccadic Jumps. *International Journal of Pattern Recognition and Artificial Intelligence* 7 (4) 831-847.

[10] Keeler, J., Rumelhart, D. (1992) A Self Organizing Integrated Segmentation and Recognition Neural Network. *Proceedings of SPIE: The International Society for Optical Engineering* Vol. 1710 744-755.

[11] Ha, T., Niggeler, D., Bunke, H. (1995) A System for Segmenting and Recognising Totally Unconstrained Handwritten Numeral Strings. *Proceedings of the Third International Conference on Document Analysis and Recognition, Vol. II* 1003-1009.

[12] Zhao, M. and Congxiao, B. (1994) Image Thresholding by Histogram Transformation. *Proceedings of SPIE: The International Society for Optical Engineering* Vol. 2238, 279-286.

[13] Feliberti, V. and Gupta, A. (1991) A New Algorithm For Slant Correction of Handwritten Characters, Masters Thesis, Massachusetts Institute of Technology.

[14] Nagendraprasad, M.V., Wang, P.S.P., Gupta, A. (1993) Algorithms for Thinning and Rethickening Binary Digital Patterns. *Digital Signal Processing* 3, 97-102.

[15] Wang, P.S.P.and Zhang, Y. Y. (1989) A Fast and Flexible Thinning Algorithm. *IEEE Transactions on Computers* 38 (5) 741-745.

[16] Sparks, P. (1992) A Hybrid Method for Segmenting Numeric Character Strings.

[17] Khan, S. (1998) Character Segmentation Heuristics for Check Amount Verification. Masters Thesis, Massachusetts Institute of Technology.

[18] Nagendraprasad, M., Sparks, P., Gupta, A. (1993) A Heuristic Multi-Stage Algorithm for Segmenting Simply Connected Handwritten Numerals. *The Journal of Knowledge Engineering & Technology* 6 (4) 16-26.