Digitization and Search: A Non-Traditional Use of HPC

Liana Diesendruck, Luigi Marini, Rob Kooper, Mayank Kejriwal, Kenton McHenry
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
Email: {ldiesend, lmarini, kooper, kejriwa1, mchenry}@illinois.edu

***Abstract*—Automated search of handwritten content is a highly interesting and applicative subject, especially important today due to the public availability of large digitized document collections. We describe our efforts with the National Archives (NARA) to provide searchable access to the 1940 Census data and discuss the HPC resources needed to implement the suggested framework. Instead of trying to recognize the handwritten text, a still very difficult task, we use a content based image retrieval technique known as Word Spotting. Through this paradigm, the system is queried by the use of handwritten text images instead of ASCII text and ranked groups of similar looking images are presented to the user. A significant amount of computing power is needed to accomplish the pre-processing of the data so to make this search capability available on an archive. The required pre-processing steps and the open source framework developed are discussed focusing specifically on HPC considerations that are relevant when preparing to provide searchable access to sizeable collections, such as the US Census. Having processed the state of North Carolina from the 1930 Census using 98,000 SUs we estimate the processing of the entire country for 1940 could require up to 2.5 million SUs. The proposed framework can be used to provide an alternative to costly manual transcriptions for a variety of digitized paper archives.**

I. Introduction

Large digital archives of handwritten scanned documents, which might contain terabytes of data spanning millions of images, currently possess no viable or practical means of searchable access. Searching capabilities are fundamental for providing genuine access to the digitized data such as the 1940 Census forms released by the US Census Bureau and the National Archives and Records Administration (NARA), a collection containing 3.25 million images and approximately 18 terabytes of raw image data. Collections such as this are valuable historical data sources that could potentially be used in a variety of eScience efforts if the contained images of handwritten data were searchable. Census data, for example, can offer insights into climate change and population movement a century or more ago. Manual transcription of the text is currently the only means of providing searchable access to handwritten data collected by multiple authors. This process, however, involves thousands of people taking months of time and high costs to complete. Here, we focus on developing an automated, low cost, and scalable alternative for providing the desired searchable capabilities.

Handwritten text recognition [1] techniques usually leverage constraints, such as a known language model [2, 3] or very restricted vocabularies [4] and structures [5], that are unavailable when dealing with the Census forms. Therefore, a content-based image retrieval [6, 7], where text recognition is not necessary, was considered. Here, partially using the Word Spotting methodology [8, 9, 10], word images are described by signature vectors containing features of the image.

Usually, the images' feature vectors would be divided into clusters and ASCII search would be accomplished by annotating the most interesting clusters and using them as indices for the searching procedure. However, even without considering the difficulties of estimating the number of clusters needed in the case of the Census dataset, this approach cannot be used here since searchable access is needed not only for popular terms but also for unusual or obscure words, such as less common private and family names or professions. As an alternative to this labeling approach, the described system is queried by images from which feature vectors are also extracted. A naive searching procedure can be implemented by using Euclidean distance to compare the query vector to each of the signature vectors in the database. Nevertheless, this approach is impractical for sizeable collections such as the US Census and some sort of index is still necessary. Therefore, hierarchies between the signature vectors (and consequently the images) are defined and used as an alternative index during the searching process. The hierarchical structure used here is a binary tree-like structure where each tree node represents a cluster of signature vectors. These cluster trees are built based on the merging steps, or dendrogram, of the vectors' clustering using Hierarchical Agglomerative Clustering [11]. Finally, when a query image is submitted to the system, whether or not these cluster trees are used to speed-up the search, multiple results consisting of its top matches are returned. Although the user still needs to search within these results, which might include bad matches, the search is downscaled from billions to a small number of images. The human involvement in this final stage of the search is then utilized to improve future query results through a passive crowd sourcing element.

High Performance Computing (HPC) resources are used to handle three fundamental pre-processing steps that are needed to build the proposed framework but are computationally expensive (see Figure 1). In the first step, all cells have to be extracted from the form images based on their columns and rows. Each form of the 1930 Census, our test collection, contains 1900 cells producing roughly 7 billion cell sub-images for the entire collection. Next, a 30-dimensional feature

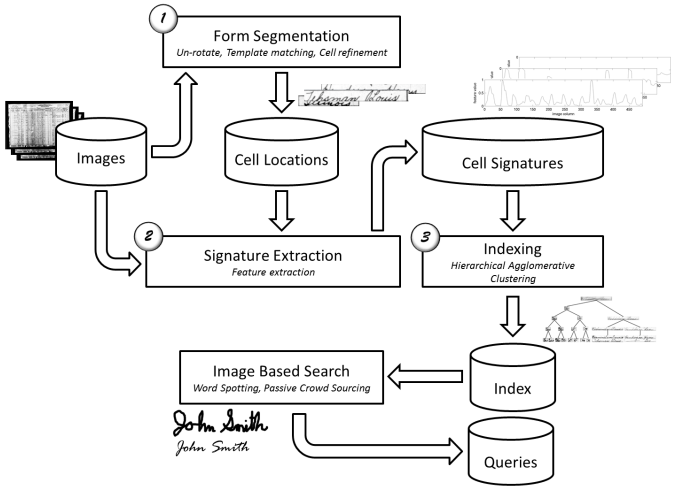

Figure 1. *A flow chart of the 3 pre-processing steps required to provide the image based search on the 1940 Census data. 1: The spreadsheet-like Census forms are segmented into individual cells. 2: A numerical signature is constructed to represent the handwritten contents of each cell. 3: A hierarchical index is constructed over the cell signatures.*

vector is extracted from each cell image. Then, in the final pre-processing step, the signature vectors have to be clustered so that the searchable cluster trees can be built.

In the following sections, the form segmentation and feature extraction pre-processing steps are briefly described. Next, the signature indexing pre-processing step is discussed including a description of different implementation approaches to the Hierarchical Agglomerative Clustering employed there. This special focus is given to the indexing procedure since it is the most computationally demanding of all pre-processing steps, resulting on the bulk of the pre-processing required time. Finally, the proposed framework is presented and experimental results are discussed. A detailed account of the first two pre-processing steps and the entire framework can be found in [12].

II. Form Segmentation

In the first pre-processing step, each form must be segmented into sub-images of the forms' cells. Since the layout of the Census forms from both 1930's and 1940's resembles a grid, the segmentation is achieved by first correctly detecting the lines that delimit the rows and columns. Prior to the detection of the lines, physical imperfections, such as tears, smudges, and bleed-throughs, and slight rotations in the images are dealt with. Previous work in form segmentation [13, 14] was adapted here paying particular attention to the resource demands required in terms of processing the 3.6 million high resolution (20-60 megapixel) Census form images.

After binarizing the image, a morphological thinning process is applied to erode the white regions around the forms. A following thinning process [15] is applied to reduce the thickness of all ink elements (lines and handwritten text alike) avoiding the creation of new artifacts [16]. Next, the image is rotated around its center by an angle estimated from the long lines detected in the image through a costly Hough transform [17]. Then, most of the desired form lines are detected by summing the amount of ink pixels along the horizontal and vertical axis of the rotated image, although it still results in extra and missing form lines (see Figure 2). With the lines detected, the process performs a search for the best alignment of the lines found in the image to the ones in a template form, which was constructed by hand labeling all horizontal and vertical form lines from one single image chosen among the entire collection. Although computationally costly, the processing time can be improved by matching horizontal and vertical lines separately, thus dealing with two 1-dimensional problems instead of a harder 2-dimensional one. At this point, the matched lines are used to delineate the position of form cells (see Figure 2), which are further refined by a small-scale version of the previous steps. Finally, the cell images are extracted and stored in the database as slices of the original image.

III. Signature Extraction

In this pre-processing step, the relevant features of the handwritten words are extracted from each of the cell images and the Word Spotting vectors are created. Before the extraction, however, some image refinement procedures are performed to reduce artifacts and increase the quality of the feature vectors. The cells are scaled to a fixed canonical height and binarized. Then, small isolated areas of ink pixels, which are not part of the text but are remains of the cell's border, are removed. Finally, the remaining ink pixels are centered within the cell's image and three features (upper, lower and transition profiles) described by Rath et al. [10] are extracted from the cell's content. The upper profile is found by scanning vertically the cell image until getting to the first ink pixel of each vertical pixel line. The bottom profile is found by a similar process starting at the bottom of the sub-image. The projection profile is calculated by summing the pixel values along the vertical axis of the cell image. These features are made invariant to some styling discrepancies by applying a cosine transform to each of them (similarly to the Fourier transform used in [10]) and using their first ten coefficients to compose the signature vectors. This not only contributes to a measure of robustness under handwriting variations, but also guarantees that every feature vector will have the exact same length.

IV. Signature Indexing

The Census data is composed of 7 billion cell images or units of information. A linear search, i.e. comparing each query to all 7 billion cells in the system, would clearly perform poorly. A very conservative estimate of 1 millisecond per comparison shows that more than 2 months would be needed just to carry out all the comparisons required for one single query. Performing smart queries such as concentrating in one specific column or state is a way of reducing the search overhead, but it is clearly not enough if the response time

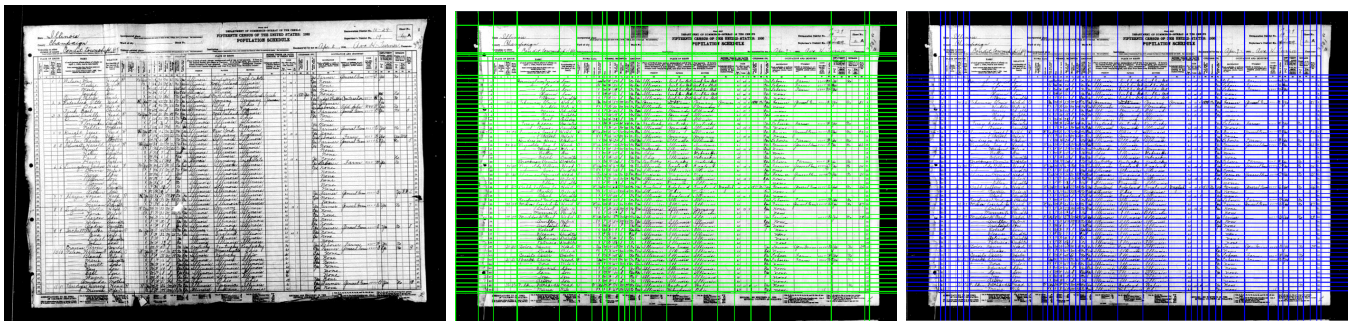

Figure 2. *Left: An example 1930 Census form image containing gray values between 0 (black) and 1 (white). Center: The form lines found within the rotation corrected image. Note the existence of missing form lines as well as non-form lines at the borders of the form. Right: The form lines obtained by matching a template to the lines obtained from the rotation-corrected image.*

to a query should be reasonable. It is evident that some kind of index of the feature vectors is needed to speed up the search procedure.

In order to generate the desired index, a Hierarchical Agglomerative Clustering [11] is performed clustering the feature vectors in a bottom-up procedure. It starts by calculating the distance between every pair of elements (in our case signature vectors) in the system and storing it in the appropriate data structures, the most basic one being a distance matrix. Every vector is represented by a cluster in the beginning of the clustering process. Then the most similar pair of clusters is merged. Next, the distance between the resulting clusters and all remaining clusters in the system have to be calculated. These two last steps are repeated until one single cluster remains. The merging steps of the clustering process are used to build a binary tree of clusters, or cluster tree, where each cluster is composed of signature vectors that were grouped together by the algorithm and is represented by an average signature vector. This cluster tree can then be used to improve the searching time; each query vector is compared to the average cluster signature vectors, descending the cluster tree until arriving at a cluster of suitable size. One of the advantages of this approach is that the size of the desired cluster is flexible and can be set and reset without having to recompute the entire hierarchy. Once the final cluster is reached, the query vector is compared to all the signatures contained in the chosen cluster in a linear manner.

The key to the considerable reduction of scale is being able to build a cluster tree that is as balanced as possible, which means that at each comparison during the descent of the three half of the remaining signatures are eliminated from the search space. From all Hierarchical Agglomerative Clustering linkages tested, the Complete Linkage generated the most balanced trees which also contained the most homogeneous clusters. Therefore, this was the linkage approach chosen despite of being very resource demanding. In the Complete Linkage approach, the distance between clusters is defined by the most dissimilar pair of vectors between them. Namely, the distance between clusters C_1 and C_2 is defined by the highest distance between any pair of vectors $v_i \in C_1$ and $v_j \in C_2$.

Clearly, the most influential computation of the algorithm in terms of time complexity is identifying the next pair of clusters that should be merged. For example, if no information about the distances between the vectors is stored, e.g. the distance matrix is not used, this step would take $O(n^2)$ time for a single iteration or $O(n^3)$ for the entire execution. When two clusters are merged, the distance between the resulting clusters and all remaining clusters in the system have to be recalculated. In the particular case of Complete Linkage, the distance between the cluster formed by merging C_1 and C_2 and any other cluster C_3 is easily computed as $max(distance(C_1, C_3), distance(C_2, C_3))$. Still, updating the distances in the relevant data structures throughout the run will take $O(n^2)$.

Several different versions of the algorithm were implemented in order to find the best fit to the available HPC resources. In all of the implemented versions, once clusters C_i and C_j are merged, the new cluster is represented by $C_{max(i,j)}$. The priority queue used here was implemented using a binary min heap which ensures that, if m elements are currently stored, adding and removing elements have an $O(log(m))$ time complexity.

The basic version ($A1$), which was used in the HPC benchmarking, is a single-threaded algorithm that uses a matrix to hold the distances between all clusters and a single priority queue. The priority queue is initialized with $n \cdot (n-1)/2$ elements, each representing a pair of clusters and the distance between them, which is used as the priority of the element. If the priority queue is kept up-to-date, meaning that after each merge the relevant elements were removed from it and new elements were added to represent the merging cost between the newly created cluster and all other clusters in the system, then the first element popped from the priority queue is guaranteed to represent the next pair of clusters that should be merged. If not, the element popped needs to be evaluated. If the element is up-to-date, meaning that since it was inserted in the priority queue the clusters it represents were not merged with other clusters, the element is guaranteed to represent the correct merge to be performed. If the element is not up-to-date and at least one of the clusters was already eliminated, the element is discarded and a new one is popped from the priority queue. And, finally, if both clusters still exist, the element priority is

updated with the new distance between the clusters and the element is re-inserted into the priority queue. One advantage of this approach is that there is no need to keep track of the elements' positions in the priority queue so to be able to perform the updates in a reasonable time. Algorithm $A1$ uses this 'lazy' approach to the updating of the priority queue since it might reduce considerably the amount of updates needed. Overall, algorithm $A1$ has $O(n^2 log(n^2))$ time and $O(n^2)$ memory complexities.

Algorithm $A2$, also single-threaded, uses the same 'lazy' approach to the updating of the priority queue. However, instead of having one single priority queue, it holds one priority queue for each row of the distance matrix, which means that once a cluster is eliminated by the merging process, the respective priority queue can be deleted, considerably speeding-up the process. Since there are n priority queues, only the priority queue with the smallest distance element in the top is popped. The popped element has to be evaluated as described in the previous paragraph. The total time complexity of this algorithm is $O(n^2 log(n))$.

Algorithm $A3$, multi-threaded, takes advantage of idle cores in the system by creating one priority queue for each and distributing the distances and their respective pairs of clusters among them. This implementation guarantees that the main thread can only observe the top element of any priority queue once the element is up-to-date. Since there is a thread taking care of each priority queue, once the top element is up-to-date it can travel down the priority queue performing additional updating until changes in the distances are made and the process has to start again.

Any of the previous algorithms can be used to create the cluster trees depending on the amount of available resources. The resulting cluster trees are stored in arrays which contain all information necessary to search the tree such as the number of elements of each cluster node, indexes to their children nodes, and the average signature vector representing each cluster node. The simplicity of these data structures and their relatively small sizes, e.g. a 50,000 vector cluster tree occupies 25 MB of storage or memory space, allow the trees to be serialized to binary arrays and easily stored in the file system or in a database.

V. A Framework for Image Based Search of Handwritten Content

We have developed the code for the above pre-processing steps and provide it as an open source framework [1]. Although the segmentation provided is designed for spreadsheet like forms, the framework could be adapted to collections of free-style text by using existing image processing algorithms (such as [18, 19]) instead of templates to segment the documents. All other subsequent pre-processing steps and system usage would remain the same.

A web interface written in the Google Web Toolkit (GWT) is also provided. This web interface allows users to submit a query to the system by either typing a string, which will be rendered by the server in a handwriting-like font prior to the search, or by drawing text in handwritten style. Once a query is submitted, the top N closest matching images, determined via the Word Spotting comparisons, are returned. A user can then scroll through these results and click on any of them to view additional information, such as the entire form with the relevant row highlighted as well as metadata associated with the form. The high resolution images are presented as image pyramids [16] through Microsoft's SeaDragon interface to allow for efficient viewing over the web. These image pyramids are generated on demand causing the first user requesting a specific form to wait for its pyramid to be constructed. The resulting pyramid is cached however so that future viewers of the form will be able to view it instantly. We have observed that image pyramid creation takes roughly 10 seconds.

One of the driving motivations for using the Word Spotting methodology was that, by returning the top N closest matches, a human is kept in the loop. Since any user must look through the presented results to find the desired matches, human transcriptions of text can be acquired with an unobtrusive passive crowdsourcing approach. Transcriptions of drawn queries, which can be acquired by methods of online handwritten recognition [1] with roughly 80% accuracy, and inputted texts are stored and later associated with the results that the user clicks on. The underlying assumption is that users will tend to click on results that match their query. Regardless, the system does not rely on the behavior of a single user, but records this information for all users. When multiple users entering the same text selected the same image, the system assumes with some level of confidence that the entered text is the same as the one displayed in the image. Thus, as users utilize the system it will improve its search capabilities.

VI. Experimental Results

In the following subsections, we analyze different versions of the Complete Linkage algorithm in terms of time and memory requirements and suitability to the HPC resources available to us, and the general computational requirements on HPC resources. For detailed results about the quality of the Word Spotting approach with and without the use of cluster trees, see [12].

A. Complete Linkage Clustering

NCSA's Ember was used to test the different implementations of the clustering method. Twelve CPUs with the combined memory of 60GB were allocated for these tests. An entire reel column containing 56500 cell images was clustered. A typical timing result can be seen on 3. All implementations presented here take advantage of the multiple available cores during the building of the matrix. Algorithms $A2$ and $A3$ also use them to initialize the priority queues required although average timing is not much improved compared to the single-threaded $A1$. Thus, until the clustering itself starts, the difference between them is irrelevant.

[1] https://isda.ncsa.illinois.edu/svn/census/trunk/

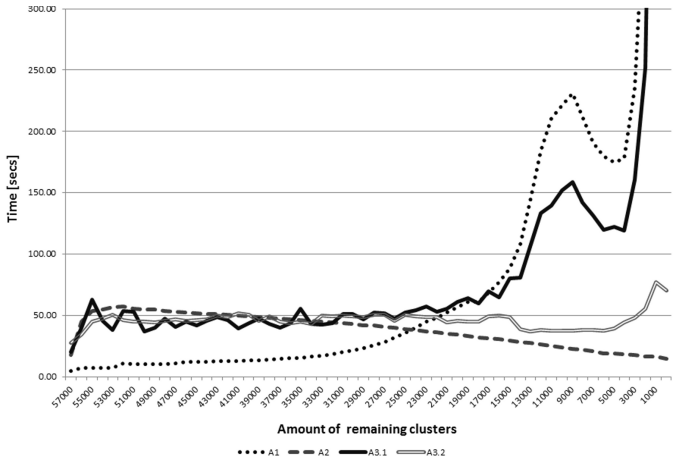

Figure 3. *Typical timing for merging clusters, in intervals of 1,000, as a function of the amount of clusters left in the system.* $A1$*: the single-threaded basic algorithm which uses one single priority queue; clustering time continues up to 4500.* $A2$*: also single threaded, uses* n *priority queues.* $A3$*: both versions are multi-threaded with the same amount of priority queues as the amount of available cores.* $A3.1$ *implements a naive approach where only the top element in each queue is kept up-to-date, while in* $A3.2$*, once the top element is up-to-date, the thread goes on to update the following elements in the queue. The clustering time for* $A3.1$ *continues to rise up to 2200 seconds in the last intervals.*

In terms of required memory, $A1$ was the most expensive using 45GB out of the 60GB available. Algorithm $A2$ required roughly 41GB of memory, while both $A3.1$ and $A3.2$ used around 37GB. For $A1$ and $A3.1$, the results indicate that most of time required by the clustering process is concentrated towards its end. Nevertheless, $A3.2$ shows that this trend can be avoided by partially updating the priority queues. In regards to the time complexity, algorithm $A2$ has the best performance even if not considerably better than $A3.2$. Overall, $A3.2$ had the shortest wall-time (around 50 minutes) due to a slight faster building of the priority queues in comparison to $A2$ (54 minutes wall-time).

While both algorithms $A2$ and $A3.2$ are promising, $A3.2$ has the advantage of not only requiring less memory but also being able to expand and use any amount of CPUs available. In systems limited in memory but with considerable amounts of free cores, this could make a strong impact in the overall time required for the indexing pre-processing step. On the other hand, algorithm $A2$ will achieve the best performance of all other implementations when running on a high-memory, low amount of CPUs environment.

Non-lazy implementations of all described approaches are also included in the available code, however managing the data structures required in order to be able to quickly update the priority queue(s) and updating elements that will never be reached make them impractical in comparison to the implementations highlighted here.

B. HPC Benchmarking

The three pre-processing steps of the framework were benchmarked on Steele at the University of Purdue and NCSA's Ember at the University of Illinois. Both systems are very different in architecture and capabilities. The Steele cluster consists of around 900 systems with 8 core processors and between 16GB and 32GB of memory. When submitting a job on Steele, it is queued based on the required resources; thus we limited ourselves to 16GB of memory usage on this system. Ember is a shared memory system that consists of 4 systems each with 384 cores and 2TB of memory. Unlike Steele the resources at ember are split in groups of 6 cores and 30GB of memory. We used in our process 12 cores and 60GB, the equivalent to a single node board. Each of the systems has some local disk space that can be accessed as well as a large parallel file system which hosts the data.

The results described here focus specifically on the state of North Carolina which consists of approximately 60 thousand images divided in 60 reels which were processed on Ember and the District of Columbia consisting of 14 reels mainly processed on Steele.

1) Form segmentation: The segmentation computation requires a little less than 3GB of memory per image that needs to be processed. Both Steele and Ember were used for the segmentation step. In the case of Steele, the jobs were split into smaller sub-jobs each running 5 threads and segmenting only 250 images, which limits the computation time and guarantees access to the system's faster queue. When using Ember, a single reel (around one thousand images) was processed per job using 12 threads and requesting a total of 8 hours of runtime. Each reel's segmentation took between 5 and 6 hours on Steele and between 1 and 5 hours on Ember. The average time to process one single image is 132 seconds on Ember and 91 seconds on Steele. A total of 2477 SUs was used for the segmentation of North Carolina's forms while 606 SUs were used for District of Columbia.

2) Signature Extraction: The process was executed both on Ember and Steele. When using Ember, a single reel was processed per job; on Steele, the jobs were once more divided into sub-jobs of 250 images each. Again, 12 and 5 threads per job were used in Ember and Steele, respectively. The processing took between 1.5 hours and 5 hours on Ember and between 4 and 9.5 hours on Steele. Some of the timing difference can be explained due to overhead of transferring the resulting sub-images from Steele to NCSA. An average of 151 seconds was needed to process one image on Ember; the average for Steele was 145 seconds per image. North Carolina's feature extraction step resulted in a total of 2840 SUs while the District of Columbia required 966 SUs.

3) Signature Indexing: Due to the high amount of memory needed during the clustering process, this third pre-processing step was run solely on Ember. Each of the 2280 North Carolina's cluster trees was processed on average in 3 hours using algorithm $A1$; an average of 123 hours were required per reel and a total of 7401 hours was needed to create the

complete index for the state's data. This pre-processing step required 88,816 SUs only for the state of North Carolina.

As described previously, many cores are available during the indexing step due to the large amount of memory it requires. Depending on the amount of idle cores and memory available in the system, algorithms $A2$ and $A3.2$ can be used to improve the running times. However, additional optimizations can be performed to utilize the extra cores. A possible improvement could be to create a pipeline to simultaneously run segmentation and signature extraction jobs with the resources requested for an indexing job (using the single-threaded $A2$ algorithm), assimilating the time required for the first two pre-processing steps in the total time of the indexing step.

VII. Conclusions

The limited accessibility of simple scanned image archives is a setback in a time when people and government are enthusiastic about freedom of access to information. The demand for searchable access to digital collections of handwritten documents will rise as the public availability of such collections grows. Nevertheless, most collections lack the public appeal required to generate the funds needed for manual content transcription. Alternatively, the proposed framework is publicly available and can be used to provide the automated search capabilities needed for public or private collections of handwritten forms. The provided code contains all the pre-processing steps, including the different implementations of the clustering procedure, ready to be run on HPC resources.

Although the system starts providing access to the handwritten information by utilizing the Computer Vision techniques described here, ideally, as more people use the system, transcriptions based on human interaction will be acquired over time through the passive crowd sourcing element. The system will continuously improve and gradually shift from a solely image based search to a hybrid of image and text based search.

The results of the North Carolina and District of Columbia benchmarking indicate that the entire 1930 Census collection could be processed in 288 days (around 0.1 days for each one of the 2878 reels). Only by incorporating the improved clustering algorithms, without taking into account the pipeline proposed in the previous section, this number could be reduced to a much faster 84 days.

Acknowledgments

This research has been funded through the National Science Foundation Cooperative Agreement NSF OCI 05-25308 and Cooperative Support Agreement NSF OCI 05-04064 by the National Archives and Records Administration (NARA). This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575.

References

[1] R. Plamondon and S. Srihari, "On-line and off-line handwriting recogntion: A comprehensive survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000.

[2] V. Lavrenko, T. Rath, and R. Manmatha, "Holistic word recognition for handwritten historical documents," *Document Image Analysis for Libraries*, 2004.

[3] T. Rath, V. Lavrenko, and R. Manmatha, "A statistical approach to retrieving historical manuscript images without recognition," *Technical Report*, 2003.

[4] R. Milewski, V. Govindaraju, and A. Bhardwaj, "Automatic recognition of handwritten medical forms for search engines," *International Journal on Document Analysis and Recognition*, 2009.

[5] S. Srihari, V. Govindaraju, and A. Shelihawat, "Interpretation of handwritten addresses in us mailstream," *Document Analysis and Recognition*, 1993.

[6] R. Veltkamp and M. Tanase, "Content-based image retrieval systems: A survey," 2000.

[7] R. Holley, "How good can it get? analyzing and improving ocr accuracy in large scale historic newspaper digitization programs," *D-Lib Magazine*, 2009.

[8] R. Manmatha, C. Han, and E. Riseman, "Word spotting: A new approach to indexing handwriting," *IEEE Conference on Computer Vision and Pattern Recognition*, 1996.

[9] T. Rath and R. Manmatha, "Features for word spotting in historical manuscripts," *International Conference on Document Analysis and Recognition*, 2003.

[10] ——, "A search engine for historical manuscript images," *International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2004.

[11] W. Day and H. Edelsbrunner, "Efficient algorithms for agglomerative hierarchical clustering methods," *Journal of Classification*, 1984.

[12] L. Diesendruck, L. Marini, R. Kooper, M. Kejriwal, and K. McHenry, "A framework to access handwritten information within large digitized paper collections," *IEEE Conference on e-Science*, 2012.

[13] R. Casey and D. Ferguson, "Intelligent forms processing," *IBM Systems Journal*, 1990.

[14] J. Liu, X. Ding, and Y. Wu, "Description and recogntion of form and automated for data entry," *ICDAR*, 1995.

[15] L. Lam and Y. Suen, "An evaluation of parallel thinning algorithms for character recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1995.

[16] D. Forsyth and J. Ponce, "Computer vision and modern approach," *Prentice Hall*, 2002.

[17] R. Duda and P. Hart, "Use of the hough transformation to detect lines and curves in pictures," *ACM Communications*, 1972.

[18] G. Louloudis, B. Gatos, I. Pratikakis, and C. Halatsis, "Text line and word segmentation of handwritten documents," *Pattern Recognition*, 2009.

[19] V. Papavassiliou, T. Stafylakis, V. Katsouros, and G. Carayannis, "Handwritten document image segmentation into text lines and words," *Pattern Recognition*, 2010.