Recognizing Text In Historical Maps Using Maps From Multiple Time Periods

Ronald Yu
Department of Computer Science
University of Southern California
Los Angeles, CA
Email: ronaldyu@usc.edu

Zexuan Luo
Department of Computer Science
University of Southern California
Los Angeles, CA
Email: zexuanlu@usc.edu

Yao-Yi Chiang
Spatial Sciences Insitute
University of Southern California
Los Angeles, CA
Email: yaoyic@usc.edu

***Abstract*—Recognizing text in historical maps is inherently difficult due to input challenges such as artifacts interfering with the text or an unpredictable rotation and orientation of the text. This paper discusses our algorithm that overcomes the limitations of the input by adding extra input consisting of multiple layers of images of the same map area but across different time periods and names of geographic entities in the United Kingdom collected from OpenStreetMap. Using our algorithm, compared to Strabo, a state-of-the-art text recognition software on maps, we obtain a 153% improvement in precision, a 31% improvement in recall, and a 75% improvement in F-score for word recognition on maps.**

I. Introduction

Recognizing text from historical maps is challenging for several reasons. First, text labels may be surrounded by noise and may intersect other text labels or non-textual artifacts on the maps such as trees or rivers. For example, in Fig. 1 the text label "St. Thomas's Church" intersects with dotted lines (roads) and is surrounded by other artifacts that could be misidentified as text. Second, text labels on a map are often at an angle because they curve with geographic features. In Fig. 1 the label "RIVER" is not oriented horizontally and bends significantly, making it difficult to differentiate the characters and the surrounding non-text objects. These are challenges that a traditional text recognition algorithms and software tools (e.g., Abbyy, FineReader, and TesseractOCR) do not need to consider [1]–[3]. Additionally, many historical maps are low-quality images where the text is blurred and squeezed together so that even a human would have a hard time recognizing it. In [4], Ye and Doermann provide a survey on text detection from scenic images (e.g., restaurant signs in Google Street View). This type of work similarly presents a problem of extracting text from a noisy background and an unpredictable orientation, but does not deal with noise directly intersecting with the text, which is a common occurrence in historical maps.

In [2], we built a system called Strabo (Section 3A) that handles these challenges to recognize text on historical maps. However, the accuracy of the Strabo can be limited by the graphical and content quality of the input maps. In a benchmark experiment on text recognition from a variety of scanned historical maps [3], Strabo obtained 48% precision and 84% recall at the character level for map labels that were not surrounded by much noise and were in the horizontal direction. For map labels that were surrounded by a large amount of noise or curved (e.g. in Fig. 1), Strabo obtained a recognition result of 32% precision and 43% recall at the character level.

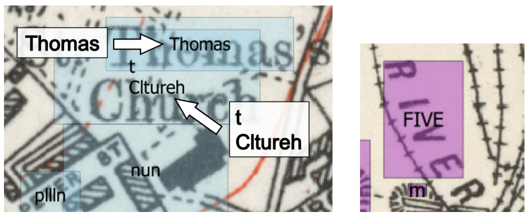


Fig. 1. On the figure on the left, the input data is extremely noisy as several artifacts intersect with the text label. Because of this, our previous text-recognition algorithm Strabo (Section 3A) has incorrectly split "St. Thomas's Church" into two results: "Thomas" and "t Cltureh". On the figure on the right, the word "RIVER" is curved and in a diagonal position, so Strabo incorrectly recognizes the word as "FIVE".

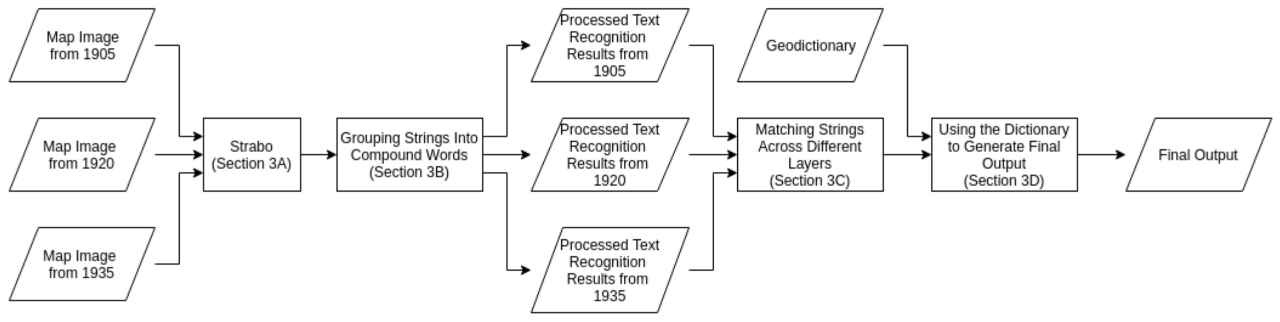
We propose an algorithm that recognizes text in noisy historical maps by using multiple images from the same geographical location but different time periods. The idea is that the noise on the maps from the same area but different time periods may affect different parts of a text label in two different maps. Hence, we can combine imperfect information from each map to recover the original text of the map labels. For example, in one map, there may be a road that intersects the beginning of a text label and in another map, there may be a group of trees that intersects the end of the text. Each map is missing some information, but combined they have enough information to recover the whole word.

In addition, our approach uses a dictionary of geographic names built from OpenStreetMap for correcting the recognized text. Our approach combines the information obtained from each of the maps along with the dictionary to generate an accurate final recognition result of the map text.

The rest of this paper proceeds as follows: In section 2, we consider related work. In section 3, we discuss our methods. In section 4, we present our experiments and results. In section 5, we conclude the paper and discuss future work.

II. Related Work

In [5], Hong and Hull cluster similar-looking images of words and combine the results of comparing each member of the cluster to a dictionary. This algorithm assumes that the



Map Image from 1905
Map Image from 1920
Map Image from 1935
Strabo (Section 3A)
Grouping Strings Into Compound Words (Section 3B)
Processed Text Recognition Results from 1905
Processed Text Recognition Results from 1920
Processed Text Recognition Results from 1935
Geodictionary
Matching Strings Across Different Layers (Section 3C)
Using the Dictionary to Generate Final Output (Section 3D)
Final Output

Fig. 2. An overview of our algorithm. We first run the maps through a text recognition algorithm such as Strabo (Section 3A) and refine these results by grouping certain strings into compound words (Section 3B). We then merge these results together (Section 3C) and combine it with a geodictionary (Section 3d) to generate the final output.

whole document uses the same font style and size and that multiple instances of the same word look similar, which is not practical when processing historical maps because multiple instances of the same word on a map can be rotated in any direction and represented in different fonts and sizes.

In [6], Lund and Ringger run multiple Optical Character Recognition (OCR) engines on the same word image and combine the output from the various OCR engines. This approach is not sufficient for reading text in historical maps. For example, if a line on the map intersects a character 'l', then it may look like a 't' and all the OCR engines would make the same mistake. In contrast, if the text label with the character 'l' appears in other maps of the same area, our approach can use this mutual information to correctly recognize the label.

In [7], Lund et. al use multiple binarization thresholds to create several images of the same word, run each of them through an OCR engine, and combine the multiple OCR results to generate a single output. This algorithm is limited if the input from a map is so poor that even using the optimal binarization levels cannot yield a remotely accurate output.

Others have approached the problem of recognizing text in maps by using a geographical dictionary of known words. In [8], Weinman uses a gazetteer of geographic entities and their locations to create a probabilistic model of possible correct words for each text label. For each map area, his algorithm queries a gazetteer for *Civil* and *Populated Places* entities in the same local geographical area as the map. However, the paper processes small-scale maps, which cover a large area and are less detailed. Since geographic entities in small-scale maps can usually be found in a gazetteer, this allows them to make spatial queries to the gazetteer. Our method can handle large-scale maps, which cover a small area but have more detail (e.g. text labels of windmills or nurseries). If we only query a gazetteer for entities within the same area as the map being processed, the query result would miss the majority of the text labels that are not place names, such as "Sewage Works" Additionally, most of the gazetteers are based on current geographical entities. Because of this, a geographic entity (e.g., a street) in a historical geographical area may not necessarily be in that same area today and therefore may not appear in the gazetteers. To solve this problem, instead of querying for entities only within the map's local geographical area, we query a dictionary of two-million entities from all over the UK.

One problem with using such a dictionary with a large number of entries is that a dictionary query may yield false-positives. A simple dictionary query cannot use the geographical location or any other method to resolve ties among several similar dictionary words and to distinguish the correct dictionary word from false-positives. In Fig. 1, Strabo incorrectly recognizes "Church" as "t Cltureh". "Cltureh" is similar to several dictionary words including "Church" and "Culture". If we only use the gazetteer on a single map layer, we do not have enough information to select the correct dictionary word. However, because we use multiple maps as inputs, we are able to resolve dictionary ties by comparing dictionary words to the recognized text on all the map layers being processed.

III. Methods

Our algorithm takes results from text recognition software for several maps covering the same area but from different time periods as input and outputs the text of what is on the maps. For a map area, we call each different map image from a different period a map layer. We first run Strabo—the system we developed in our previous work—on each of the maps to generate a Strabo result for each map label on each of the map layer images. We then run our algorithm using the Strabo results as input (Section 3A). Our algorithm contains three main stages (Sections 3B – 3D): grouping nearby strings in a map layer into compound words, matching strings that are likely to represent the same entity in different map layers, and finally comparing the matched results to the dictionary to generate the final output. A flowchart showing the stages of our algorithm can be seen in Fig. 2.

A. Strabo

In our previous work, we developed a system called Strabo to recognize text labels in map images [2]. When certain conditions are satisfied, Strabo groups nearby characters into strings. These conditions are based on cartographic labeling principles such as that all the characters in a map label have a similar height and width and that two characters in the same text label are usually closer together than two characters from different labels [2]. Strabo then detects the text label orientation and rotates every label to the horizontal direction [2]. Finally, Strabo uses an OCR package to recognize the text. For each map label detected on the map image, Strabo outputs the detected text of the label and a bounding box of the location of the label. Examples of what the output of Strabo looks like can be seen in any of the figures in this paper. In this paper, we use the output from Strabo as the initial recognition results and then use these results from multiple maps to improve the overall recognition accuracy.

B. Grouping Strings Into Compound Words

One challenge in text recognition from maps is that recognition algorithms commonly split what should be a single string into separate strings when the map label covers two or more lines on the map or when the spacing between two words is exceptionally large. For example, if a map contains the entity "St. Thomas's Church" (Fig. 1), the label may be incorrectly separated into two independent misspelled strings "Thomas" and "t Cltureh". This is a loss of potentially useful information, so our algorithm first groups text in the initial recognition results from the same map image by concatenating them into a compound string.

The first step in our grouping process exploits several properties of compound strings. First, if a compound string is split into two strings in the initial recognition results, those strings are likely to be separated by a small geographic distance both in the x and y directions. Second, if two strings indeed represent substrings of the same map label, the characters in the label should have a similar size. If the characters are of a similar size, the bounding boxes of the two results should have a similar height. The two strings do not necessarily need to have the same width since the width is also affected by the number of characters in the string.

To make use of these properties, our process takes a string in the initial recognition results, and based on that string, builds a chain of strings that are located spatially near each other on the map. The algorithm does so by considering the bounding box of each of the strings in the initial recognition results and appending them to the end of the chain if the following conditions are satisfied:

$$d_x < T_x * w_{res} \tag{1}$$

$$d_y < T_y * h_{res} \tag{2}$$

$$\frac{1}{T_{ratio}} < \frac{h_{link}}{h_{res}} < T_{ratio} \tag{3}$$

where *res* is the bounding box of a string in the the initial recognition results, *link* is the bounding box of the final link in the chain, *w* and *h* are the width and height of the bounding box, *d* is the Euclidean distance on the map between *res* and *link*, and T_x, T_y, and T_{ratio} are tunable thresholds.

In Fig. 1, the string "Thomas" is taken as the base of a chain. Since "Thomas" has a similar x-value and y-value to the string "t Cltureh", "t Cltureh" is appended to "Thomas". Moreover, since the string"nun" is directly below "t Cltureh" and they have similar x-values, the algorithm appends "nun" to the end of the chain comprised of the other two strings. Next, although "pllln" is close to the other three entities, its height is much smaller than the height of of "nun". Because of this, "pllln" is not appended to the chain, and our final chain from this step is "Thomas t Cltureh nun".

As seen in our example, this process may incorrectly append unrelated strings to the end of a chain. In order to solve this problem, we segment the chain into smaller chains of words, which we call segments, such that each of the smaller chains is similar to an entity in the dictionary. A queried word is considered similar to a dictionary word if the Levenshtein Edit Distance between the words d_{LED} satisfies:

$$d_{LED} \leq \max(2, \min(5, L) - 2) \tag{4}$$

where *L* is the number of characters in the query word.

If a string is not similar to any entries in the dictionary, it most likely does not represent any actual entity and is instead more likely to be two unrelated nearby strings and should not be grouped together into a compound string.

To segment the chain, we initially treat the chain of strings as a single compound word and check if it is similar to any entry in the dictionary. If the compound string is similar to any set of entities in the dictionary, then the chain is accepted as a compound string. We can now process this compound string in the next stage to determine which dictionary word the map label actually represents. If no similar string exists in the dictionary, the string at the end of the chain is discarded. We continue to compare the remaining part of the chain to the dictionary and discard the end until the chain matches an entry in the dictionary.

In the case of Fig. 1, "Thomas t Ctlureh nun" is not similar to any string in the dictionary, so "nun" is discarded from the end of the chain, and the process repeats on the remaining chain. "Thomas t Ctlureh" is similar to several dictionary strings such as "Thomas Church" and "Thomas Culture", so it is accepted as a valid compound string and processed in the next stage of the algorithm to determine which dictionary word its map label most likely represents. This process is repeated on the discarded words until the original chain of words is completely segmented into dictionary words. Since "nun" is in the dictionary, it is kept as a valid string, and the final output of this example is two strings: "Thomas t Ctlureh" and "nun".

C. Matching Strings Across Different Layers

This step groups labels from different map layers that could represent the same entity in order to use the information from each of these layers in a later step. Given a compound word from a single map layer, we search the compound words from other map layers to find words that are most likely to express the same geographical entity. The process is based on two metrics: the geographical distance and string similarity.

For two strings from different layer to be considered similar, they first must be geographically close, meaning that their physical distance in meters should be less than a tunable parameter T_{geo}. We consider geographic distance because map entities usually do not physically move very far away over time, so we expect a single entity to remain around the same physical spot on maps from all time periods.

However, T_{geo} is usually large enough so that each compound word will usually have multiple geographically close strings in other map layers, so we filter the nearby compound words using the string similarity because having a high similarity score indicates that two strings are likely expressing the same thing. If no string has a similarity score higher than a tuneable threshold T_{sim}, we conclude that there are no meaningful matches and ignore the strings.

In this step, string similarity is determined by our implementation of the Needleman-Wunsch algorithm, which we have enhanced to account for common OCR mistakes. For example, text-recognition algorithms commonly misrecognize an 'n' as a 'u', so we lower the scoring penalty incurred when an 'n' in the initial recognition results is replaced with a 'u' in a dictionary word and vice versa. These enhancements make our implementation of Needleman-Wunsch more accurate but slow down the comparison process because we check for many different kinds of common OCR mistakes. To overcome this challenge, we first use Elasticsearch to index and generate a list of similar strings from a dictionary of two-million entities. Then we use the slower but more accurate Needleman-Wunsch algorithm to identify similar compound words.

In the example from Fig. 3 and Fig. 4, there are several entities on the 1935 layer that are geographically nearby to the map label "homa urch" from the 1900 layer. Of these geographically similar strings, we group "homa urch" with the string from the 1935 map layer that has the highest string similarity score, which in this case is "Thomas t Cltureh".

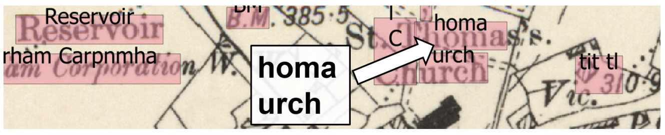


Fig. 3. Map Layer from 1900. The initial recognition result of "St. Thomas Church" is "homa urch".

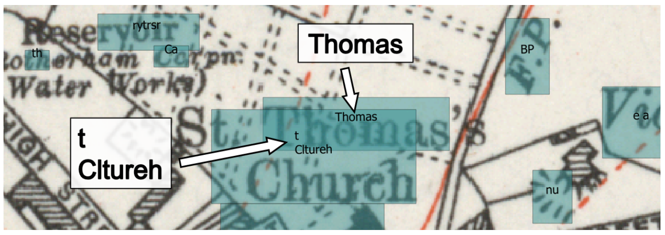


Fig. 4. Map Layer from 1935. The initial recognition result of "St. Thomas Church" is "Thomas t Cltureh".

D. Using the Dictionary to Generate Final Output

After we group the compound words from different layers, for each group we select the entity in the dictionary that is most likely to represent the actual text in the map. Again, we store the dictionary using Elasticsearch, a highly indexed database that can process search queries quickly. Using the Levenshtein-Edit Distance to measure similarity, an Elasticsearch query can return a set of several hundred similar strings immediately. We first query the database for each compound word in the group to immediately return a few hundred candidates out of the more than two-million database entries. We then narrow down the candidates using the slower but more accurate Needleman-Wunsch algorithm. In the example group in Fig. 3 and 4, we query "homa urch" and "Thomas t Cltureh" to the dictionary, yielding two sets of several hundred dictionary candidates.

We now merge the sets of candidates into a smaller set of candidates such that each element of the final resulting set is similar to all of the compound words in the group. Since we only need to make several hundred comparisons at this stage, we now run the Needleman-Wunsch algorithm, which is slower but more accurate, to select the correct candidate. We compare each set of dictionary candidates to its corresponding compound word and assign a similarity score. We then merge the set into a single list by taking the union and then sorting the list by similarity score. If a dictionary candidate appears in the sets of more than one compound word, the similarity score of the candidate in the final merged set is the sum of the scores that the candidate receives in each of the individual candidate sets of the compound word group. We use this summation method to favor words that appear in multiple compound words, which usually indicates that a candidate is similar to multiple map layers and is likely to be the correct result.

In our example from Fig. 3, "homa urch" has a high string similarity with "Rhema Church", so "Rhema Church" is included in the final set (Table 1). "Thomas t Cltureh" has a high similarity with "St Thomas Church", so "St Thomas Church" is included in the final set. "Thomas Court" is moderately similar to both "homa urch" and "Thomas t Cltureh", so "Thomas Court" appears on the list. Table 1 shows a sample of how some of the top scores in the merged set are generated. The first column is the dictionary candidate, the second column is the candidate's normalized Needleman-Wunsch similarity score with "homa urch", the third column is the candidate's normalized similarity score with "Thomas t Cltureh", and the fourth column is the candidate's final score, which is the summation of the second and third column. A "N/A" in an entry indicates that the dictionary candidate was not in the original candidate set of the compound word group corresponding to that column.

TABLE I
MERGING PROCESS FOR "HOMA URCH" AND "THOMAS T CLTUREH"

Dictionary Candidate	Score with "homa urch"	Score with "Thomas t Cltureh"	Final Score
Thomas Court	0.25	0.467	0.717
Thomas Close	0.25	0.444	0.694
Rhema Church	0.65	N/A	0.65
Thomas Greatorex	N/A	0.622	0.622
St. Thomas Church	N/A	0.611	0.611

We now truncate the set so that only the top 20 results remain. We do this because we have empirically observed that the correct result never falls outside of the top 20. Using this truncation speeds up the algorithm as there are less Needleman-Wunsch comparisons to perform.

We then compare the string from all layers to each of the words in this final set to compute the final score for each candidate. Each element in the final set is assigned its final score based on the summation of its Needleman-Wunsch similarity score to the compound words from each of the layers. We do this second round of comparison in case a dictionary candidate is similar a compound word with respect to the Needleman-Wunsch algorithm, but is not close in Levenshtein Edit Distance and therefore does not appear in the compound word group. This step essentially fills in the "N/A" entries in Table 1. In Table 2, we show Table 1 again but this time with the "N/A" entries replaced with the appropriate values. The updated entries in Table 2 are in bold.

TABLE II
SECOND PASS OF NEEDLEMAN-WUNSCH COMPARISONS

Dictionary Candidate	Score with "homa urch"	Score with "Thomas t Cltureh"	Final Score
Thomas Court	0.25	0.467	0.717
Thomas Cook	0.35	0.444	0.694
Rhema Church	0.65	**0.506**	**1.156**
Thomas Greatorex	**0.356**	0.622	**0.978**
St. Thomas Church	**0.667**	0.611	**1.278**

Table 2 shows how "St. Thomas Church" is correctly selected as the final output because of its high similarity to both of the compound words. This scoring system favors dictionary candidates that are similar to the compound words from multiple layers. Since the correct dictionary word is usually similar to all layers, there is a high chance that our algorithm will select the correct word.

The actual text on the map is "St. Thomas's Church", which is not in the dictionary, so instead we output an almost equivalent string in "St. Thomas Church". In this example, our algorithm took incomplete input from two layers and combined the partial information from each layer and a dictionary to output the correct text on the map label.

IV. EXPERIMENTS

We tested the performance of our algorithm on the Historical Ordinance Survey Maps of UK. Each map area covered 1,000x1,000 meters of the British National Grid and each image was 1512x1512 pixels. We tested on nine map areas with 440 map labels. For map areas 1 to 4, we were able to obtain map layers from the years 1905, 1920, and 1935 as input. For map areas 5 to 9, we were able to obtain map layers from the years 1900, 1920, 1935, and 1945 as input.

We first ran Strabo on all the maps and calculated the precision, recall, and f-score at the word-by-word level for the earliest map layer of each map area (either 1900 or 1905).

We then ran our algorithm on each map area using the following parameters: T_x=2, T_y=1.5, T_{ratio}=1.8, T_{geo}=500, and T_{sim}=0.57. We calculated precision and recall based on the earliest map layer of each map area.

We compared the precision and recall at the word-by-word level from running Strabo on a single map and from running our algorithm on multiple map layers while using the dictionary as input. We display the results in Table 3. The first column is the ID number of the map area. The second, third, and fourth column are the precision, recall, and F-score, respectively when our algorithm processed multiple layers. The fifth, sixth, and seventh column are the precision, recall, and F-score, respectively when only Strabo was used to process layer 1905 for map areas 1-4 and layer 1900 for map areas 5 to 9. The eighth column is the number of total words present on the map.

TABLE III
COMPARING THE PRECISION AND RECALL AT THE WORD-BY-WORD LEVEL OF WHEN STRABO IS RUN ON A SINGLE MAP LAYER AND WHEN OUR ALGORITHM IS RUN ON MULTIPLE LAYERS

ID	Prec.	Recall	F-Score	Strabo Prec.	Strabo Recall	Strabo F-Score	GT
1	53.19%	29.76%	38.16%	26.76%	22.61%	24.51%	84
2	66.67%	29.85%	41.23%	28.84%	22.38%	25.20%	67
3	72.72%	44.44%	55.17%	41.67%	37.03%	39.21%	54
4	68.18%	38.46%	49.17%	41.17%	35.89%	38.34%	39
5	60.00%	37.50%	46.15%	14.51%	22.50%	17.64%	40
6.	84.00%	43.75%	57.53%	25.00%	18.75%	21.43%	48
7	30.76%	33.33%	31.99%	8.53%	29.17%	13.20%	24
8	44.00%	25.58%	32.35%	24.07%	30.23%	26.80%	43
9	62.50%	24.39%	35.08%	18.60%	19.51%	19.04%	41
AVG	59.84%	33.86%	43.24%	23.65%	25.91%	24.72%	48

We can see how our algorithm compared to running Strabo and dictionary post-processing on a single map. On average, we obtained a 153% increase in precision, a 31% percent increase in recall, and a 75 percent increase in F-score.

In cases such as the example used in Fig. 1, our algorithm was able to recover the map label's actual text even when noise made the text impossible to recognize accurately on each individual layer. While the noisy input would have caused Strabo to incorrectly recognize the text on all layers, our algorithm combined the partial information from each layer and correctly recognized the label for all layers.

dwarhe
Aldwarke

Fig. 5. The map label on the left is from the 1900 layer. There are two lines intersecting the beginning of the word "Aldwarke", so only the end of the word is recognized as "dwarhe". The map label on the right is from the 1945 layer. There is little surrounding noise image, so the label is correctly recognized as "Aldwarke".

In other cases, the input was noisy on several but not all layers, so although Strabo obtained an inaccurate result on the layers where noise interfered with the text, Strabo correctly recognized the text label for the layer without noise. With the help of the correctly recognized layer, our algorithm recovered the original text on the map label successfully. In Fig. 5, in the 1900 layer on the left, a line intersected with the 'A' and 'l' in

"Aldwarke", causing Strabo to only recognize the end of the string as "dwarhe". However, in the 1945 layer on the right, there were no noisy artifacts intersecting with the label, so Strabo successfully recognized the label as "Aldwarke". Our algorithm combined these results and determined that the map label on the 1900 layer said "Aldwarke".

Our algorithm was also more robust to noise. Basic dictionary post-processing on Strabo was able to filter out some of the noise by putting a minimum similarity threshold T_{sim} that a dictionary word must have to the Strabo result. However, T_{sim} was fairly low, so many noise inputs were still recognized as dictionary words. Because we expected the Strabo result on at least one layer to be fairly similar to the correct dictionary word, we were able to use a higher T_{sim}, allowing our algorithm to better filter out noise.

The remainder of our discussion will focus on common errors in our algorithm.

A. *Ground Truth Not In Dictionary*

When the map's ground truth was not in our dictionary, we always obtained an incorrect result. This error comprised of up to 27 percent of our recall errors on certain map areas. For example, in Fig. 6, "Garrowtree Farm" was recognized by Strabo as "Garrmcfree Farm" in the 1900 layer and as "Garrowtree Farm" in the 1920 layer. Since the Strabo result was similar to the ground truth in the 1900 layer and was a correct in the 1935 layer, our algorithm should have output "Garrowtree Farm". However, because "Garrowtree Farm" was not in our dictionary, our algorithm did not consider "Garrowtree Farm" as a dictionary candidate, and instead wrongly output the dictionary candidate "Sparrow Lee Farm". One potential solution is to use the dictionary as a suggestion and still output a text recognition result that is not in the dictionary if the text recognition confidence is high enough.

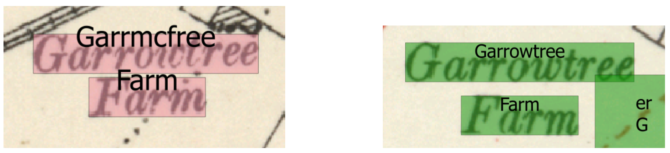


Fig. 6. In left-hand image, Strabo incorrectly labeled "Garrowtree Farm" as "Garrmcfree Farm" in the 1900 layer.In the right-hand image, Strabo made an exact match and correctly recognized "Garrowtree Farm" in the 1920 layer.

B. *Incorrectly Identifying Noise As Text*

Another issue that lowered precision was that Strabo would recognize noisy, non-textual artifacts as text. This comprised of up to 32 percent of the precision error for certain map areas. In map images such as Fig. 7, there were many trees that were arranged in a way that resembled characters. Thus, Strabo frequently recognized these trees as text. Although our algorithm was more robust to noise than Strabo and ignored most noise inputs they were not similar to any entries in the dictionary database, sometimes all layers would detect noise that was similar to a dictionary entry, so our algorithm recognized the noise as text instead of ignoring it. Generally, Strabo results that were actually just trees tended to follow certain patterns such as consisting mostly of 'm' and 'n' or of taller letters like 'l' and 'k', so we may consider taking advantage of these properties in the future in order to solve this problem.

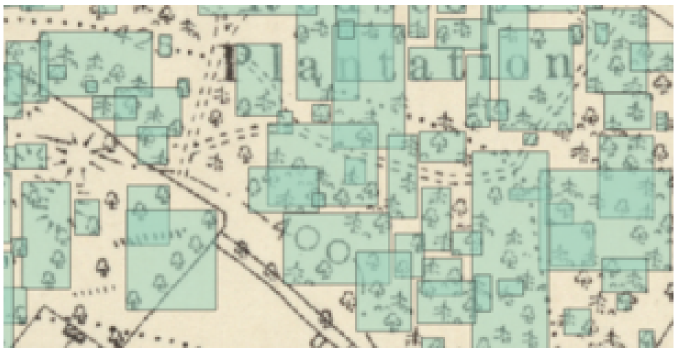


Fig. 7. In rural areas, Strabo incorrectly recognized many trees and non-textual artifacts as text, which lowered the precision of our algorithm.

V. Discussion and Future Work

Our proposed algorithm in this paper makes strides in extracting text from historical maps by using additional input in the form of multiple map layers of the same area but from different time periods and a large dictionary database of geographical entities in the UK. Future works may include solving the common errors described in Section 4. One area of research may be exploiting the properties of noise data described in section 4 and using a machine learning algorithm such as Support Vector Machines to correctly ignore noise inputs. Moreover, most of our thresholds used in Sections 3 are currently empirically determined. In the future, we may develop a method to automatically determine these thresholds.

Acknowledgment

The authors would like to thank Dr. Craig A. Knoblock, Narges Honarvar Nazari, and Tianxiang Tan for their contributions to this project.

References

[1] Y.-Y. Chiang, S. Leyk, and C. A. Knoblock, "A survey of digital map processing techniques," *ACM Comput. Surv.*, vol. 47, no. 1, pp. 1:1–1:44, 2014.

[2] Y.-Y. Chiang and C. A. Knoblock, "Recognizing text in raster maps," *Geoinformatica*, vol. 19, no. 1, pp. 1–27, 2015.

[3] Y.-Y. Chiang, S. Leyk, N. Nazari, S. Moghaddam, and T. T. T., "Assessing impact of graphical quality on automatic text recognition in digital maps," *Computers & Geosciences*, In Revision, 2016.

[4] Q. Ye and D. Doermann, "Text detection and recognition in imagery: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 7, pp. 1480–1500, 2015.

[5] T. Hong and J. J. Hull, "Improving ocr performance with word image equivalence," in *Fourth Symposium on Document Analysis and Information Retrieval*, (Las Vegas, NV, USA), pp. 197–199, 1995.

[6] W. B. Lund and E. K. Ringger, "Improving optical character recognition through efficient multiple system alignment," in *Proceedings of the 9th ACM/IEEE-CS Joint Conference on Digital Libraries*, pp. 231–240, 2009.

[7] W. B. Lund, D. J. Kennard, and E. K. Ringger, "Combining multiple thresholding binarization values to improve ocr output," *Proc. SPIE*, vol. 8658, pp. 86580R–86580R–11, 2013.

[8] J. Weinman, "Toponym recognition in historical maps by gazetteer alignment," in *Proceedings of the 2013 12th International Conference on Document Analysis and Recognition*, pp. 1044–1048, 2013.