Proceedings of the Seminar

Artificial Intelligence, Natural Language Processing and Information Retrieval

University of Colorado, Colorado Springs

August 5, 2011

Editors: Jugal Kalita and Terrance Boult

Funded by

National Science Foundation

Automatic Extension of a Lexical Ontology Using Web Resources

James Austrow
The Ohio State University
154 W 12th Avenue
Columbus, OH 43210
austrow.1@osu.edu

ABSTRACT

Lexical ontologies, or databases of words and their relationships, are valuable tools for a variety of natural language processing applications. Their costly construction and maintenance times, however, limit their scope and ease of development. This causes them to be difficult to keep up to date with current terminology and concepts. We develop and compare several procedures for automatically updating an existing lexical ontology, focusing on WordNet, based on Wikipedia articles. An appropriate hypernym for each article must be found in order to maintain the hierarchical structure of WordNet, so the different methods we compare focus on different ways to determine this hypernym.

Categories and Subject Descriptors

D.2.8 [**Software**]: Software Engineering—*performance metrics*; H.4 [**Information Systems**]: Applications—*miscellaneous*

General Terms

Experimentation, Performance

Keywords

ontology

1. INTRODUCTION

Lexical ontologies have proven to be very useful in the field of natural language processing. The structure of grouping synonymous word senses together into synsets and arranging them in a graph of relationships such as hypernym-hyponym pairs provides much semantic information that is not apparent from the words themselves. However, one drawback of these systems is that they are typically constructed and maintained by hand, requiring costly effort. One such ontology in common use today is WordNet [1], which will be the focus of this research.

We aim to investigate approaches overcoming the costly maintenance time of lexical ontologies by automatically integrating information found on the web. Wikipedia is a user-edited source of words and concepts, updated more or less in real time. The fact that it is a relatively current source of information makes it attractive for the purpose of keeping a comprehensive ontology such as WordNet updated. Various methods of incorporating word relation data from Wikipedia will be explored.

The remainder of this paper is organized as follows. Section 2.1 describes previous work that is related to this research. Section 2.3 describes the general approach this research is based on and outlines improvements we have made. Section 2.4 details how the methods are implemented. Section 2.5 explains how the results of this research are tested and details the results of the experiment. Section 2.6 shows what future improvements we plan on making. Finally, Section 3 is the conclusion.

1.1 Related Work

The ideas of this research are based heavily on the work of Jiang et al, who propose a method of incorporating articles from Wikipedia into WordNet [3]. They rely on category information from the article to determine its hypernym and achieved encouraging results. However, in a few cases their algorithm detects an incorrect sense of the hypernym within WordNet or identifies a bad category as the most likely hypernym and thus misclassifies the article. Therefore, alternate methods of hypernym extraction are considered. Sang [6], building on the work of Hearst [2] and Snow et al. [7], describes a method of hypernym extraction based on fixed patterns of text. Additionally, Kleigr et al. [4] note that there are several patterns unique to Wikipedia articles that can aid in hypernym identification.

2. EXTENDING WORDNET

2.1 Method

The method of Jiang et. al. is used to automatically extend WordNet with Wikipedia entries. This involves compiling the categories of each article that appear in WordNet as potential hypernyms of the article. The definition of each potential hypernym (from WordNet) and the text of the article are compared for concept similarity to determine the best match. If no category of the article appears in WordNet, the head term of each category is considered instead, and the original category of the chosen hypernym is inserted into WordNet as well.

One of the difficulties of this method is that it is quite slow. A similarity measure must be computed between each pair of concepts appearing in the Wikipedia article and in the WordNet definition of the current candidate, and the total number of concepts in this set is frequently over two thousand. We suggest that the time to run this algorithm can be vastly reduced while maintaining most of the accuracy by only using the concepts from a summarization of the Wikipedia text, such as the first paragraph [5]. As the first paragraph is typically an overview of the topic, most of the more relevant concepts should appear there.

However, the main difficulty of their method is hypernym determination, so additional means of hypernym extraction from text are applied and compared. These new sources of potential hypernyms improve the pool of candidates, leading to an increased likelihood that the correct hypernym can be selected. Looking for fixed patterns in the text of the article should help to identify the hypernym [6]. Furthermore, the first sentence of most articles generally indicates a good hypernym, giving it as "*[title of article]* is a *[hypernym]* which *[elaboration]*" or something similar. This pattern is generally consistent across Wikipedia, especially for larger and more popular articles [2].

2.2 Implementation

The general algorithm for extending WordNet is as follows:

1. Obtain a Wikipedia article which is not already in WordNet
2. Collect potential hypernyms and add all synsets that contain words found this way to a candidate pool
3. For each candidate synset, compute the matching score between its definition and the text of the Wikipedia article [3]
4. Insert the article into WordNet under the synset with the best matching score

This process is graphically illustrated in Figure 1.

We examine improvements that can be made in steps two and three. In step three, we look at improving the speed of the matching score computation by limiting the number of concepts that need to be compared to only those in the first paragraph of Wikipedia text. This has an interesting impact on the accuracy of the resulting hypernyms, which will be further elaborated in the Experiment section.

Furthermore, in step two, we look at ways to improve the quality of the hypernym candidates. The original method uses the categories of the article as hypernym candidates, but the categories do not always make good hypernyms. For example, that method often chooses "birth" or "death" as hypernyms for people because their article has a category such as "1963 Births." Thus, alternate sources for the hypernym must be considered. The second noun phrase in the first sentence of the article tends to be a good hypernym candidate and generally appears in a predictable location in the parse tree of that sentence, as shown in Figure 2. In this example, we would like to extract "tone poem" from the tree; to do

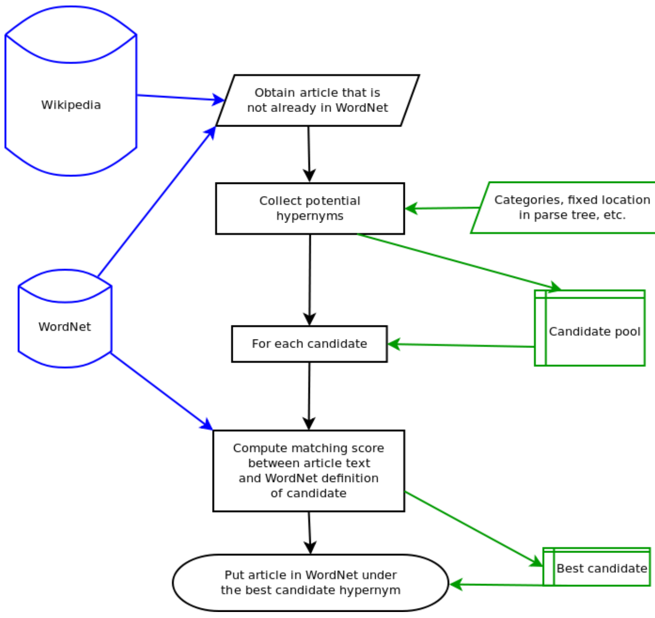

Wikipedia
Obtain article that is not already in WordNet
Collect potential hypernyms
Categories, fixed location in parse tree, etc.
WordNet
Candidate pool
For each candidate
Compute matching score between article text and WordNet definition of candidate
Best candidate
Put article in WordNet under the best candidate hypernym

Figure 1: Flowchart for adding Wikipedia articles to WordNet.

this, we can retrieve the head of the noun phrase under the verb phrase under the root of the tree. The tree in Figure 3 illustrates this. Any article whose first sentence follows this "is a" construct will have this kind of structure in its parse tree. After retrieving this word or phrase, we take the best-matching synset that contains it as the hypernym for the article.

To extract the hypothesized hypernym from the parse tree, a specific set of rules is followed for descending the children of each node in the tree, based on the part of speech tag. Those rules are as follows:

1. Start at the S root
2. If there is an S child, select it (to select the first clause of complex sentences)
3. Select the VP child
4. Select the NP child. If successful, stop
5. If no NP child, try VP child, then NP child
6. If still unsuccessful, try S child, then VP, then NP

The goal of this procedure is to find the first noun phrase after the subject of the sentence, which is likely to be a hypernym [2]. It was found experimentally that these patterns cover the majority of sentences that we have some hope of parsing using this method. After obtaining this noun phrase, the following procedure is applied:

1. If this node has a PP child and it is the word "of," select that node
2. Select the NP child. If there is no NP child, select the original NP node

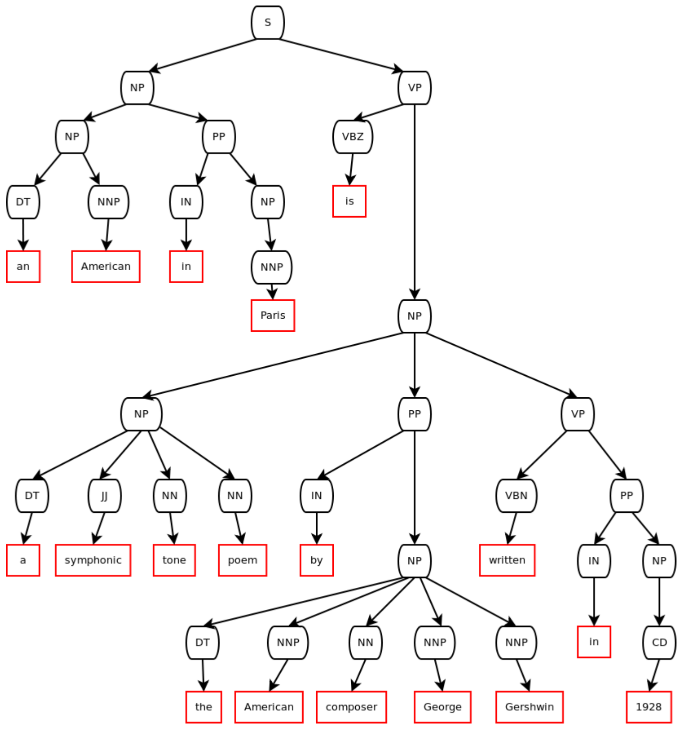


Figure 2: Tree structure of a typical first sentence in a Wikipedia article: "An American in Paris is a symphonic tone poem by the American composer George Gershwin, written in 1928."

3. If a new NP node was found this way, repeat the procedure

The goal of this step is to follow the "[article] is a type of [hypernym]" pattern, which seems to be fairly common. After performing these steps, we end up with a noun phrase node in the parse tree. To obtain the WordNet hypernym from the noun phrase, the following is performed:

1. Retrieve the head word of the noun phrase. This is the working hypernym
2. Find the word just before the head word in the original sentence and add it to the front of the working hypernym
3. If this working hypernym does not appear in WordNet, remove the newly added word and return the working hypernym. Otherwise, repeat from step 2.

This handles cases such as the "An American in Paris" example above, where "tone poem" appears in WordNet but "symphonic tone poem" does not.

Sometimes the original head term does not appear in WordNet. This is corrected for in a hybrid method we also tested, which simply places the output of the parse tree method in the pool of candidates. If the retrieved hypernym is not in WordNet, it will simply use the categories as per the Jiang et. al. method.

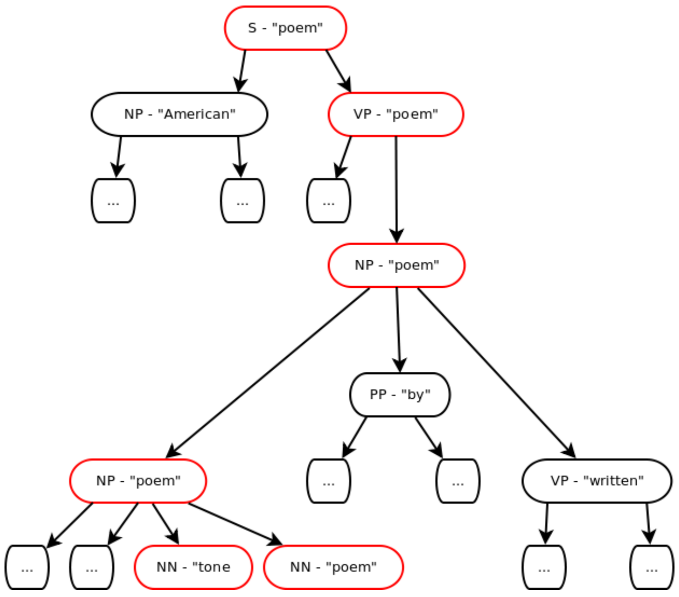


Figure 3: Relevant portion of the same tree with head words annotated and path to hypernym highlighted.

2.3 Experiment

As the method of extending WordNet with Wikipedia is be based on the work of Jiang et. al. [3], the results were tested using the same method as them. A random sample of the new additions were be taken and scored by human readers on a 5-point scale. The original results scored an average of 4.26 (Good). The three methods we tested are: a version of the original method which only uses the first paragraph of the article in concept matching; the parse tree extraction method described previously; and a hybrid method which combines the two approaches by inserting the result of the parse tree method into the candidate pool of the first method.

The version which uses only the first paragraph for concept matching proves to be much faster than using the whole text of the article as shown in table 1. Case studies showing the change in accuracy are given in Tables 2 and 3. Accuracy seems to have fallen for a few articles, but interestingly, accuracy improved for some other articles. We suggest that this improvement is caused by the fact that the body of the article may contain elaboration about a specific feature of the main topic and skews the matching computation towards that facet. This can be seen for the "America the Beautiful" article especially, which was mistakenly put under "Pikes Peak" in the old method. "Barbra Streisand songs" is not the best hypernym, but the new algorithm at least was able to put it under "song, strain" rather than "peak, crown, crest,

Table 1: Speed comparison between different amounts of article text used

Text Used	Articles Processed in Five Hours
Whole Article	7
First Paragraph	3347

Table 2: Case study of the original method of hypernym extraction

Wikipedia Article	Category	WordNet Hypernym
An American in Paris	1928 compositions	musical composition, opus, composition, piece, piece of music
Animalia (book)	Puzzle books	book, volume
Allan Dwan	1981 deaths	death
America the Beautiful	Pikes Peak	peak, crown, crest, top, tip, summit
Albert Sydney Johnston	1862 deaths	death, last

Table 3: Case study of the faster version of the original method

Wikipedia Article	Category	WordNet Hypernym
An American in Paris	1928 compositions	constitution, composition, physical composition, makeup, make-up
Animalia (book)	Puzzle books	book, rule book
Allan Dwan	Writers from Ontario	writer
America the Beautiful	Barbra Streisand songs	song, strain
Albert Sydney Johnston	United States Military Academy alumni	alumnus, alumna, alum, graduate, grad

top, tip, summit," so it certainly is an improvement. Using only the first paragraph may have the effect of keeping the concepts used to compute the matching score closer to the overall topic of the article. The new method in which the hypernym is extracted from a parse of the first sentence is showing mixed results. It is able to correctly identify hypernyms that the original method do not, but it also misses many others that the original method identifies correctly. Case studies of this method are shown in Table 4.

The three methods were each run long enough to process 100,000 Wikipedia articles. Of these, a random sample of 300 results were selected from each method. Three volunteers were recruited to grade these samples, using the same scale as used by Jiang et. al. The same 300 entries were used across all three methods in order to better compare results. The results from each volunteer were then averaged to obtain the final scores for each method. The average score over the whole dataset for each method is given in Table 5, while Table 6 shows the individual score breakdown. All of the new methods have a lower average than the original (Jiang et. al.) method. The loss in accuracy for the summarized method is understandable given its dramatic speed increase, but the parse tree search method clearly is underperforming in its current state. The hybrid method improves on it slightly. One issue with the parse tree method is that the rule for following the preposition "of" down to the next noun phrase frequently caused the algorithm to go right past the real hypernym; thus, the rule is not applicable in all situations. This highlights one of the weaknesses of the fixed rule approach that is used here.

2.4 Future Work

The main improvement that could be made to these methods would be to make the parse tree search much more flexible by using machine learning to decide where in the parse tree the hypernym is most likely to be. The fixed rule approach is not nearly flexible enough to handle the large variety of sentence structures that occur in Wikipedia articles, even just in the first sentence. If a good set of features to test could be developed, it seems likely that large improvements could be made to this method.

Table 4: Case study of the parse tree method

Wikipedia Article	WordNet Hypernym
Allan Dwan	conductor, music director, director
America the Beautiful	song, strain
Albert Sydney Johnston	career, calling, vocation
Citizen Kane	film
Commonwealth of England	republic
Groucho Marx	comedian, comic
Miss Marple	character, reference, character reference

The matching score computation also stands to be improved. In some cases, even when the correct hypernym is chosen out of the candidates, an incorrect sense of that word is chosen from WordNet because of its matching score. For example, as seen in Table 3, the algorithm correctly found "character" as the hypernym word for "Miss Marple," but chose the sense "character, reference, character reference" rather than the more correct "fictional character, fictitious character, character." This suggests that the degree of matching between the Wikipedia text and the candidate's WordNet definition may not be the best measure to use. Furthermore, the use of a true word sense disambiguation algorithm is likely to improve these results.

3. CONCLUSION

The usefulness of lexical ontologies is well documented, their main downside being only the cost and human effort required to create and update them. If such an ontology could be produced automatically, this downside would be all but eliminated. Furthermore, the expansion of user-edited sources of information on the web has greatly incentivised their use as comprehensive lexical ontologies, but so far no technique exists to effectively automatically compile the information they contain. This research has taken steps towards the

Table 5: Average scores for the three methods (and original method)

Method	Average Score
Original	4.26
Category	3.88
Parse Tree	3.42
Hybrid	3.45

Table 6: Evaluation results for the three methods (and original method)

Method	Excellent	Good	Fair	Neutral	Bad
Original	193	47	26	12	22
Category	102	100	64	29	5
Parse Tree	39	134	64	40	23
Hybrid	45	111	91	41	12

automatic extraction of word relations required to build an ontology from web sources like Wikipedia. The testing done here demonstrates a few approaches that are not quite flexible enough for this purpose but hint at improvements to be made in the future.

Acknowledgement

The research reported in this document has been funded partially by NSF grants CNS-0958576 and CNS-0851783.

4. REFERENCES

[1] C Fellbaum. *WordNet: An Electronic Lexical Database.* The MIT Press, Cambridge, MA, 1998.

[2] Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics - Volume 2*, COLING '92, pages 539–545, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[3] S Jiang, L Bing, B Sun, Y Zhang, and W Lam. Enhancing ontology actively and learning the concept granularity agilely: Keeping yourself current.

[4] Tomáš Kleigr, Vojtěch Svátek, Krishna Ch, Jan Nemrava, and Ebroul Izquierdo. Wikipedia as the premiere source for targeted hypernym discovery, 2010.

[5] Aleksander Kolcz, Vidya Prabakarmurthi, and Jugal Kalita. Summarization as feature selection for text categorization, 2001.

[6] Erik Tjong Kim Sang. Extracting hypernym pairs from the web. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, pages 165–168, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.

[7] Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. Learning syntactic patterns for automatic hypernym discovery. In *Neural Information Processing Systems*, 2004.

Evaluating Methods for Summarizing Twitter Posts

Gary Beverungen
St. Mary's College of Maryland
16800 Point Lookout Rd.
St. Mary's City, MD
gebeverungen@smcm.edu

Jugal Kalita
University of Colorado at Colorado Springs
1420 Austin Bluffs Pkwy
Colorado Springs, CO
kalita@eas.uccs.edu

ABSTRACT
Microblogs like Twitter[1] are becoming increasingly popular and serve as a source of ample data on breaking news, public opinion, etc. However, it can be hard to find relevant, meaningful information from the enormous amount of activity on a microblog. Previous work has explored the use of clustering algorithms to create multi-post summaries as a way of understanding the vast amount of microblog activity. Clustering of microblog data is notoriously difficult because of non-standard orthography, noisiness, limited sets of features, and ambiguity as to the correct number of clusters. We examine several methods of making standard natural language processing techniques more amenable to the domain of Twitter including normalization, term expansion, improved feature selection, noise reduction, and estimation of the number of natural clusters in a set of posts. We show that these techniques can be used to improve the quality of extractive summaries of Twitter posts, providing valuable tools for understanding and utilizing microblog data.

Keywords
microblogs, normalization, extractive summarization, term expansion, clustering

1. INTRODUCTION
Microblogging is a relatively new form of communication, providing both new opportunities and new challenges for Natural Language Processing (NLP). Microblogs such as Twitter may have as many as 2.5 million posts per day about a variety of topics and from a diverse set of users. One could mine this data to discover public opinion [13], breaking news, sentiment analysis [14], or even predict the stock market [4]. Clearly there is the potential for vast amounts of useful data to be found from microblog posts. Unfortunately, standard approaches to NLP often fail in the domain of microblog posts, and it is not clear which techniques for extracting and utilizing microblog data are most useful. Clearly it is necessary to determine which tools will be most helpful in making use of microblog data. We explore the use of clustering as a means of detecting important subtopics in sets of Twitter posts and selecting posts which are representative of the activity on that topic.

[1]http://www.twitter.com

WSDM'11, February 9–12, 2011, Hong Kong, China.

Several obstacles stand in the way of processing microblog posts such as those on Twitter. First, Twitter posts are highly non-standard. While most standard NLP techniques were developed for long, structured, grammatical text, Twitter is short, colloquial, and ungrammatical. Users frequently misspell words either unintentionally (*teh, waht*) or intentionally, by expanding words, abbreviating words, or using lexical/numeric substitutions (*loooovveeeee, rly, c u l8r*). Twitter posts also frequently contain other non-standard tokens such as acronyms (*lol, smh*), hash tags (*#beatcancer, #iusuallylieabout*), user tags (*@nina1983*), or Twitter specific terminology indicating "re-tweeted" posts (*RT*) and trending topics (*TT*). This poses a problem for NLP techniques, since two posts with alternate spellings of some word may not be considered related, when in fact they are. While normalization of Twitter posts remains a difficult problem, progress has been made by those like Kaufmann and Kalita [12] and Han and Baldwin [10]. We use the techniques developed in Kaufmann and Kalita to normalize Twitter posts and hopefully improve the effectiveness of other NLP techniques.

Second, Twitter posts are very short, no more than 140 characters and typically not more than ten words or so. Unfortunately, this means that Twitter posts are feature sparse, and that comparisons between posts will be difficult. This is especially problematic for clustering, which is highly sensitive to the features chosen for comparison. This problem could be alleviated by expanding terms in the twitter posts to include relevant similar terms (essentially adding additional features), selecting more descriptive features (e.g. just named entities), using n-grams instead of unigrams, or some combination of the three.

Additionally, even with good features Twitter posts could be hard to cluster. One challenge of clustering in general is determining how many clusters are "inherent" in the data set. Most clustering algorithms require the number of clusters to be specified ahead of time, but it is not always obvious what that number should be. Choosing incorrectly can lead to suboptimal clustering of data, splitting coherent clusters into multiple clusters or grouping distinct clusters into one.

Luckily, several methods for determining the "correct" number of clusters exist, including those by Tibshirani et al.[18] and Ben-Hur et al. [2]. Ideally, these methods would allow us to determine the number of salient subtopics and generate summaries that more accurately reflect the posts.

However, even if we can identify relevant subtopics in a set of Twitter posts, there will still be many posts which do not fit well into any subtopic or cluster. Undoubtedly, there will be many posts that are largely unrelated to other posts in the data set. These outliers may have a negative impact on the ability of the clustering algorithms to correctly identify subtopics, and hurt summarization overall. Thus it may be beneficial to try to remove outliers and other noisy posts from the data set before using other NLP techniques.

This paper presents preliminary attempts at making the domain of microblog posts more ammenable to NLP techniques by combining techniques for tackling each of the challenges described above.

2. PREVIOUS WORK

2.1 Normalizing Posts

As mentioned, microblog posts are notoriously hard to process computationally, containing frequent misspellings, unfamiliar named entities, OOV words, improvised abbreviations, slang, and novel lexography. Converting the post to standard English would improve processing. Research has used the noisy channel model, wherein normalizing noisy text T to a standard form S by assuming T is an error of S. The most likely S is found by finding the probability of T being an error of S time the probability of S occurring. In other words, $S_{max} = argmax(P(S|T)) = argmax(P(T|S)P(S))$. Various approaches have been made to characterize the error model, $P(T|S)$, including edit distance [5] and letter transformation [7]. Machine translation may be able to assist with normalization [12]. Part-of-speech (POS) parsers created for Twitter [9] might be able to give additional information about ambiguous words. We utilize the normalization tool developed by Kaufmann et. al. [12].

2.2 Clustering

Previous work by Sharifi et al. [17, 16] has explored the topic of multi-post extractive microblog summarization. They explored frequency based, graph based, and cluster based methods of selecting multiple posts that conveyed information about a given topic without being redundant. They found that ROUGE-N scores and human evaluation did not provide an obvious choice of one summarizer over another [1]. In fact, most multi-post summarizers did not perform significantly differently from a simple Most Recent summarizer. However, clustering algorithms could be improved in a number of ways, as described here.

2.2.1 Determining the Number of Clusters

The clustering algorithms used in Sharifi et al. [1] were fairly basic, and there remains room for improvement. One limitation of the clustering algorithms used in Sharifi et al. [17] is that they generate a specific number of clusters that must be determined before running the algorithm. This means that an arbitrary number of clusters must be chosen for a set of microblog posts, regardless of the actual distribution of posts. Although Sharifi et al. determined that most users thought four clusters was appropriate, the optimal number of clusters likely varies from topic to topic. Ben-Hur et al. have used a stability based method for determining the number of clusters in data, building off of the idea that a good clustering should be stable, consistent, and robust to noise. [2] Alternatively, Tibshirani et al. have used the gap statistic, a measure of within cluster dispersion of a clustering compared to an expected value, to determine the correct number of clusters. [18] Using these methods, we may be able cluster microblog posts in a way that more accurately reflects relevant sub-topics.

2.2.2 Choosing Features

In addition to using different clustering techniques, it may be possible to improve results by improving the way in which posts are compared. It may be the case that simple word level similarity doesn't capture what humans perceive to be the important aspects of sentence similarity. Hatzivassiloglou et al. have shown that including information about the NP heads, named entities, events, and other information included in the sentences, it is possible to improve the quality of clusters [11]. Alternatively, limiting features to certain parts of speech (Nouns, Verbs) may significantly cut down on the extraneous features in the posts. Using a Part-of-Speech (POS) tagger, like the one developed by Gimpel et al. would allow us to do just that. [9] Additionally, some authors have looked at expanding posts by adding highly related terms, thus overcoming the feature sparsity of Twitter posts. Perez-Tellez et al. use pointwise mutual information to determine which words are most similar to ones already in the post. [15] Chen et al. use a similar technique, but also use inforation from Wikipedia to expand posts. [6] By improving the feature vector to more accurately reflect perceived similarity, we may be able to improve the effectiveness of clustering, and thus, the quality of the resulting summary.

3. METHODS

3.1 Data

Our data set includes 50 topics selected from Twitter's list of Trending Topics. For each topic, posts are selected by taking 1500 posts from the Twitter API and processing them as follows:

1. Convert HTML encoded characters to ASCII.
2. Discard any posts that aren't in English. (Defined as containing at least 40% English Words.)
3. Discard a post if there has already been another post by the same user.
4. Discard a post if it is spam.
5. Reduce number of posts by taking the most recent 100.

3.2 Normalization

This process can is described in greater detail in [**?**]Sharifi-MS.

For normalization, we utilize the normalized developed by Kaufmann and Kalita. [12] Their method uses a combination of lexical normalization, syntactic disambiguation, and statistical machine translation to convert Twitter posts from their noisy, non-standard, ungrammatical form to something resembling more standard texts.

3.3 Clustering

Previously, Sharifi et al. have experimented with several types of clustering algorithms. [1] They found that, of the algorithms they tested, bisecting k-means was the most effective at producing summaries. Thus, that is the clustering we will be using for our purposes. For the remainder of this paper, when we refer to clustering a set of Twitter posts, we a referring to bisecting k-means clustering.

3.4 Determining the Optimal Number of Clusters

3.4.1 Non-counting Method

As a control, and to facilitate comparing our results with those in Sharifi et al. [1], we implement a trivial cluster counter which determines that there should be four clusters regardless of the data. Four clusters was chosen to imitate the method of clustering in Sharifi et al.

3.4.2 Stability Based Method

In order to determine the number of "inherent" clusters in a set of Twitter posts, we chose to implement Ben-Hur et al.'s stability based method. [2] The crux of the algorithm is that a good clustering of data should be relatively stable and robust to noise. While a suboptimal clustering may be clustered differently every time the clustering algorithm is run, a good clustering should produce roughly the same result every time. Additionally, even if a small amount of the data is removed, as long as all the clusters are adequately represented, a good clustering should still be able to find the "correct" clutsering. Thus, but clustering random subsamples of a data set and comparing the similarity of the clusterings for different numbers of clusters, we should be able to get a good estimate for which number of clusters is the most stable, and thus most apt to fit the data. A description of the implementation of the algorithm is as follows:

```
Given: Data Set ← $X, Float \leftarrow f$ for k = $2 to k_{max}$ do
  for $i = 1$ to $num\ iterations$ do
    Sub1 = subsample(X, f)
    Sub2 = subsample(X, f)
    Cluster1 = cluster(Sub1)
    Cluster1 = cluster(Sub2)
    Similarity(k,i) = Sim(Cluster1, Cluster2)
  end for
end for
```

: Algorithm for determining the average similarity of two samples of the data set for each number of clusters, k

The similarity function returns a measure of the similarity of two clusterings. To compute said similarity, first we construct an nn matrix, where n is the number of posts in the clusterings and each entry in the matrix, (i,j) is defined as:

1: if posts i and j are in the same cluster

0: otherwise.

Once we have obtained matricies for each clustering, M_1 and M_2 respectively, we let N_{ij} be the number of entries in which M_1 and M_2 have the values i and j, respectively. The similarity measure is then defined as the Jaccard coefficient:

$$\frac{N_{11}}{N_{01} + N_{10} + N_{11}}. \tag{1}$$

Thus, after running the algorithm described above, we have a list of i similarities between clusterings for each possible number of clusters k. The k we ultimately choose is the one for which the average of each of the i similarities is the greatest.

3.4.3 Gap Statistic

As an alternative to the Stability Based method we implement the Gap Statistic method described in Tibshirani et al. [18] The gap statistic makes use of the measure of within cluster dispersion. For each cluster C_r in a clustering, $n_r = |C_r|$ and $D_r = \Sigma_{i,i' \in C_r} d_{ii'}$ where $d_{ii'}$ is the squared Euclidean distance between posts i and i'. Within cluster dispersion for a clustering of k clusters is then

$$W_k = \Sigma_{r=1}^{k} \frac{1}{2n_r} D_r. \tag{2}$$

Conventional wisdom has it that when there is a sharp decrease in the within cluster dispersion, the correct number of clusters has been found. Tibshirani et al. calculate an expected withing cluster dispersion using a null reference data set, and determine how far below that value the real data falls. As our null reference, we generate a set of posts each with length equal to the average length of posts in the real data set. Each word in each null reference post is chosen at random, uniformly across all words seen in the real data set. For each value of k, we cluster the null reference set b times and compute the withing cluster dispersion, W_k^*, for each clustering, as well as average within cluster dispersion, $W_{k\,avg}^*$, and the standard deviation of the dispersions, $W_{k\,stddev}^*$. We then cluster the real data set and calulate the within cluster dispersion, W_k. The gap statistic is defined as

$$Gap(k) = W_{k\,avg}^* - W_k \tag{3}$$

and the chosen value of k is the smallest value for which the following inequality holds

$$Gap(k) \geq Gap(k+1) - W_{k\,stddev}^*. \tag{4}$$

3.5 Feature Selection

3.5.1 Term Expansion

In order to overcome the small size of microblog posts, we expand the post to include terms similar to other terms in the post. We follow the methods described in Tellez-Perez et al. [15] Given a set of posts, $X = p_1, p_2, \ldots p_n$, we find the Pointwise Mutual Information (PMI) for each pair of terms, $t_i, t_j \in p_n$ found in the posts

$$PMI(t_i, t_j) = \frac{P(t_i, t_j)}{P(t_i)P(t_j)} \tag{5}$$

For each term t_i in a post p_n, Tellez-Perez et al. find the PMI between that term and each other term found in X. Any term t_j for which $PMI(t_i, t_j)$ is greater than some threshold value is added to the post p_n. This process is repeated for each post in X. While Tellez-Perez et al. set the threshold value manually, we found that the PMI between a pair of posts varied greatly depending on the set of posts X. Depending on the topic, the average PMI and the variation in PMI could vary greatly. Thus, we set it as the one standard of deviation above the average PMI of all pairs of posts in X.

3.5.2 N-Grams

In effort to pick up on the more nuanced relationships between terms in a post, we used n-grams as features instead of simple unigrams. Posts that contain the same words in the same order are more likely to be related than posts that simply have the same words contained somewhere in the post. By comparing posts using n-grams instead of or in addition to unigrams we may be able to get a more nuanced measure of similarity between posts. For a post p with terms $t_1, t_2, \ldots t_i \in p$ we define the n-grams of p as

$$n_1 = (t_1, t_2, \ldots t_n), n_2 = (t_2, t_3, \ldots t_{n+1}), \ldots n_{i-n} = (t_{i-n+1}, \ldots t_{i-1}, t_i).$$

We experiment with using unigrams, bigrams, and trigrams, and a combination of unigrams, bigrams, and trigrams.

3.6 Noise Reduction

As another method of improving the validity of cluster, we investigate noise reduction. The goal of noise reduction is to remove posts that do not fit well into any cluster. This can be determined in a number of ways, but for our purposes we define this as any post for which the average distance to another post is more than one standard of deviation more than the population average. Average distance from post i to another post is defined as:

$$D_i =$$

$1n\Sigma_{j\in S}d(i,j)(6)$where S is the set of posts to be summarized, $n = |S|$, and d(x,y) is the squared Euclidean distance between posts i and j. This value is calculated for every post in S, and any post for which the value is one standard of deviation above the average value is removed.

3.7 Evaluation

3.7.1 Cluster Validity

The end goal of each of these methods is to produce good clusters of Twitter data and thus descriptive summaries of each Twitter topic. Therefore, we will evaluate the effectiveness of each technique in terms of the cluster quality and quality of the overall summary. To measure the validity of a particular clustering we use a modified Dunn's Index, as described in Bezdek Pal. [3] Dunn's Index is the ratio of the minimum between cluster distance to the maximum within cluster dispersion. The assumption is that a good clustering will produce well separated clusters and clusters that are densely packed. While Bezdek Pal offer several definitions of cluster distance and dispersion, we use the following definitions, which they cite as among the most effective. Intercluster distance $D_{is,t}$ of two clusters, S and T, is defined as

$$D_{is,t} = \frac{1}{|S||T|}\Sigma_{x\in S,y}d(x,y) \tag{7}$$

where d(x,y) is the squared Euclidean distance between posts x and y. Within cluster dispersion, D_{wS}, of a cluster, S, is defined as

$$D_{wS} = 2(\frac{\Sigma_{x\in S}d(x,\mu_s)}{|S|}) \tag{8}$$

where μ_s is the mean of the feature vectors of the posts in S. Thus, D_w is essentially twice the average distance of a post in S from the mean of S. Given a set of clusters $X = C_1, C_2, \ldots C_n$ the modified Dunn's Index is then

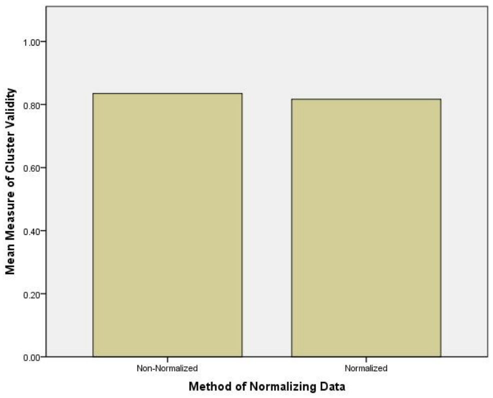
$$DI = \frac{argmin_{s,t\in X}(D_{is,t})}{argmax_{s\in X}(D_{ws})}. \tag{9}$$

4. RESULTS

To determine the effectiveness of each strategy, we perform an ANOVA and test for main effects for each strategy (normalization method, cluster counting method, noise reduction method, and feature selection). Thus, we perform the clustering process for each of the 50 data sets for every combination of methods (2 normalization methods 3 cluster counting methods 2 noise reduction methods 5 feature selection methods = 60 total methods of clustering). SPSS version 19 was used to perform the ANOVA with $\alpha = .05 and post-hoc tests performed using Tukey's HSD. The results below show the main$

4.1 Normalization

Normalization had a small but statistically significant effect on overall cluster validity. Surprisingly, normalizing the posts led to decreased clustering performance. Normalized posts had an average cluster validity of .817, whereas the non-normalized posts had a validity of .835, as seen in Figure 4.1. Thus, we note that normalizing posts does not drastically impact the clustering of Twitter data.



Graph.jpg

Figure 1: The mean cluster validity for normalized and non-normalized posts. The effect is small but significant.

4.2 Cluster Counting Method

We found that the Gap Statistic method of evaluating the number of clusters significantly outperformed both the baseline and stability based methods. Surprisingly, the stability

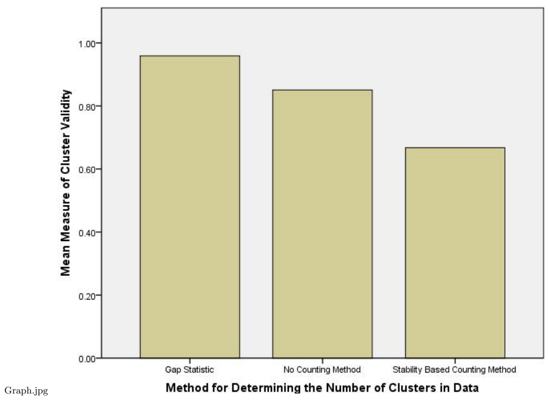


Figure 2: The mean cluster validity for each of the different methods of counting the correct number of clusters. There were significant differences between each method of counting clusters.

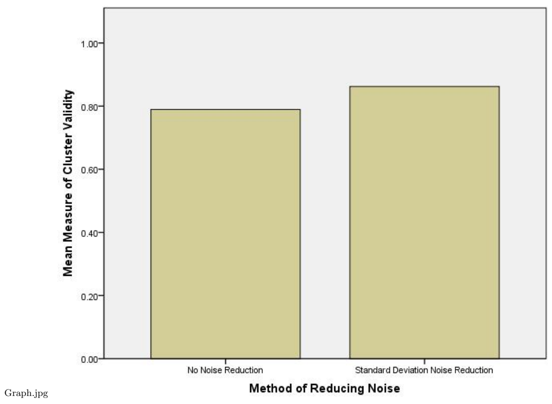


Figure 3: The mean cluster validity with and without noise reduction. Noise reduction significantly improved results.

based method produced the worst clusters. Upon further analysis, we found that the stability based method tended to favor larger numnbers of clusters, drastically decreasing the intercluster distance and increasing Dunn's Index. The gap statistic produced an average cluster validity of .959, the stability based method .668, and the baseline method .851, with significant differences between each method, as shown in Figure 4.2

4.3 Noise Reduction

The noise reduction method described above significantly improved cluster validity scores. The noise reduction method had an average cluster validity score of .862 as opposed to the a cluster validity of .789 for clusters with no noise reduction. These results can be seen in figure 4.3.

4.4 Feature Selection

Two feature selection methods showed significant improvement over the rest, bigram feature selection and trigram feature selection. A table with the means for each feature selection method can be found in table 4.4. Bigrams performed significantly better than either unigrams, the combination of unigrams, bigrams and trigrams, and term expansion techniques. Likewise, Trigrams significantly outperformed unigrams, the combination of unigrams, bigrams and trigrams, and term expansion techniques. There was no significant difference between bigrams and trigrams, nor were there significant differences between any of the three remaining techniques. A graph of the results can be seen in Figure 4.4.

Feature Selection Method	Average Cluster Validity
Unigrams	.792
Bigrams	.869
Trigrams	.868
Combination	.806
Term Expansion	.793

Table 1: The mean cluster validities for each method of feature selection.

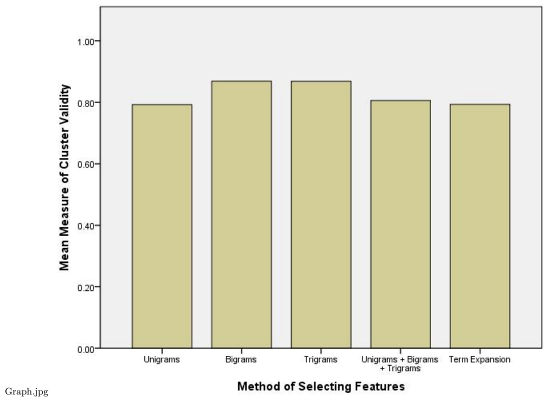


Figure 4: The mean cluster validity for each method of feature selection. Both bigrams and trigrams performed significantly better than trigrams, combination, and term expansion. There were no other significant differences.

5. FUTURE WORK

Once we have obtained a multi-post extractive summary from a set of microblog posts, there remains the question of how to order the posts in a way that maximizes the coherence of the overall summary. We have done some preliminary analysis of the feasibility of ordering posts, but found that, when humans were asked to manually order posts selected for a summary, the inter-rater correspondence was only slightly more than the value expected by random ordering. However, the assumption that there is a single "correct" ordering is perhaps unfounded. There may be several plausible, coherent orderings for a set of posts. More research is needed to determine a good method for measuring the coherency of a summary and finding good orderings.

We have focused exclusively on k-means clustering, specifically bisecting k-means. However, other types of clustering algorithms exist. Heirarchical or density based algorithms could just as easily be used to find structure in microblog data. Additionally, density based clustering has the advantage of being fairly robust to noise and obviates the need to choose a number of clusters. However, density based clustering has its own challenges and parameter that need to be fine tuned. Since all of the mentioned clustering algorithms still depend on good features and similarity measures, most of the work in this paper could still apply, but further investigation is necessary to determine how efffective these other clustering algorithms are at summarizing microblog data.

With term expansion, we have used only the pointwise mutual information (PMI) technique, but other methods of expanding the number of features in a post exist. Future work could look at adding WordNet synonyms and/or hypernyms to the posts to increase the number of features. Additionally, some authors have looked at using linked web content to find more information about a particular post. [8] Lastly, we have looked at term expansion for the addition of unigrams based on the PMI with other unigrams. However, this need not be the case. If a particular n-gram has a high measure of PMI with any other n-gram, there is grounds for adding it as well. Given the success of bigrams and trigrams in generating good clustering, it might be worthwhile to look into term expansion with bigrams and trigrams as well.

6. CONCLUSION

In this paper, we have explored several means of mitigating the difficulty of processing microblog posts. We have examined methods of normalizing posts as a way of reducing noise, extracting descriptive features from microblog posts, and improving the effectiveness of existing clustering techniques. As a result, we have generated relatively descriptive summaries of particular topics in microblogs. Furthermore, the techniques we have examined here could be used to make many other NLP techniques more effective in the microblog domain.

Acknowledgement

The research reported in this document has been funded partially by NSF grants CNS-0958576 and CNS-0851783.

7. REFERENCES

[1] David Inouye Beaux Sharifi and Jugal Kalita. Extractive summarization of twitter microblogs. *Under Revision for ACM Transactions for Speech and Language Processing*, 2011.

[2] Asa Ben-Hur, Andre Elisseeff, and Isabelle Guyon. A stability based method for discovering structure in clustered data. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, pages 6–17, 2002.

[3] J. C. Bezdek and N. R. Pal. Some new indexes of cluster validity. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 28(3):301–315, January 1998.

[4] Johan Bollen, Huina Mao, and Xiao-Jun Zeng. Twitter mood predicts the stock market. *CoRR*, abs/1010.3003, 2010.

[5] Eric Brill and Robert C. Moore. An improved error model for noisy channel spelling correction. pages 286–293, 2000.

[6] Qing Chen, Timothy Shipper, and Latifur Khan. Tweets mining using wikipedia and impurity cluster measurement. In *ISI'10*, pages 141–143, 2010.

[7] Bingqing Wang Fei Liu, Fuliang Weng and Yang Liu. Insertion, deletion, or substitution? normalizing text messages without pre-categorization nor supervision. To be published in the proceedings of the Association of Computational Linguistics, 2011.

[8] Yang Liu Fei Liu and Fuliang Weng. Why is "sxsw" trending? exploring multiple text sources for twitter topic summarization. To be published in the proceedings of the Association of Computational Linguistics, 2011.

[9] Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A. Smith. Part-of-speech tagging for twitter: Annotation, features, and experiments. In *ACL (Short Papers)*, pages 42–47. The Association for Computer Linguistics, 2011.

[10] Bo Han and Timothy Baldwin. Lexical normalisation of short text messages: Makn sens a # twitter. 2011.

[11] Vasileios Hatzivassiloglou, Luis Gravano, and Ankineedu Maganti. An investigation of linguistic features and clustering algorithms for topical document clustering. In *SIGIR*, pages 224–231, 2000.

[12] Joseph Kaufmann and Jugal Kalita. Syntactic normalization of twitter messages. pages 149–158, December 2010.

[13] Brendan O'Connor, Ramnath Balasubramanyan, Bryan R. Routledge, and Noah A. Smith. From Tweets to Polls: Linking Text Sentiment to Public Opinion Time Series. In *Proceedings of the International AAAI Conference on Weblogs and Social Media*, 2010.

[14] Alexander Pak and Patrick Paroubek. Twitter as a corpus for sentiment analysis and opinion mining. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*, Valletta, Malta, may 2010. European Language Resources Association (ELRA).

[15] Fernando Perez-Tellez, David Pinto, John Cardiff, and Paolo Rosso. On the difficulty of clustering company tweets. In *Proceedings of the 2nd international workshop on Search and mining user-generated contents*, SMUC '10, pages 95–102, New York, NY, USA, 2010. ACM.

[16] B. Sharifi, M.-A. Hutton, and J.K. Kalita. Experiments in microblog summarization. In *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, pages 49 –56, aug. 2010.

[17] Beaux Sharifi. Automatic microblog classification and summarization. Master's thesis, University of Colorado at Colorado Springs, 2010.

[18] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a dataset via the gap statistic. 63:411–423, 2000.

Analysis of Mental Health Expression on Twitter

Michael Billot, University of Colorado at Colorado Springs

Abstract—**Since 2006 Twitter has existed as a platform which allows users to broadcast brief textual messages of no more than 140 characters. These short pieces of text are known as tweets. The most common purposes of tweets are daily conversations, information sharing, news critiques, and updates about a Twitter user's life. By facilitating such content Twitter promotes a wide array of emotional expression. In this research Twitter is queried for tweets containing the keyword "depressed." To begin analysis, a collection of personal, expressive tweets will be gathered. These collected tweets will contain content where the Twitter user appears to be sincerely writing about their depression. Analysis will be done by using human judges to score these expressive tweets along the Profile of Mood States (POMS) six dimensions of mood. A corpus of words will be produced based on the magnitude of scores for the six mood dimensions: tension, anger, depression, vigor, confusion, and fatigue.**

I. Introduction

THE amount of information presented by Twitter is staggeringly vast. The number of daily posts has continuously grown since it's introduction in 2006. In November of 2010, gigatweet was unable to continue its counting of tweets due to technical changes made by Twitter. However, over the course of the year prior, gigatweet documented a sustained increase in tweets from a rate of around 300 tweets per second to a rate of over 1000 tweets per second[1]. Within this mountain of information there are a large number of tweets where users are discussing and writing about mental health issues. An interesting subset of these tweets are those in which a Twitter user explicitly shares his or her feelings about an experience or affliction with a mental health issue. A desire to further understand the nature of these tweets is what motivates this research.

II. Motivation

The primary motivation of this research is to better understand the degree at which mental health issues are expressed by Twitter users. The expressive tweets scored by POMS will illustrate which dimensions of mood are being experienced by people sharing content about personal mental health. A measure of mood could be complemented by lexical analysis to draw conclusions about how a Twitter user's language is reflective of his emotional condition. Machine learning techniques using the collection of tweets could help automate the detection of similar ones in the future. The most profound product of this research might be a computational tool that automatically detects users who are habitually expressing negative sentiments or mental health problems. Automatic detection would be invaluable for a longitudinal analysis of these twitter users. A long term analysis could show the reasons why people choose to user Twitter as platform to share mental health issues. A long term analysis could also give insight into the benefits and positive effects that people experience from their expressive writing on Twitter.

[1] http://gigatweeter.com/analytics

III. Related Work

Bollen et al. performed sentiment analysis research of all public Twitter posts over a period of four months [1]. They used a syntactic term based approach to measure the sentiment of tweets via a psychometric instrument called POMS. They found that spikes in certain dimensions of sentiment could be correlated with critical events such as the 2008 presidential election and stock market fluctuations. This approach showed that supervised learning is not the only viable way to perform sentiment analysis of Twitter.

//Bollen and Pepe analyzed the mood expressed in 10,741 emails to the future [2]. To score the moods of the emails, they used an extended Profile of Mood States metric. They extended POMS original 65 adjectives with WordNet 3.0 synonyms. The extended list of POMS words were then stemmed using the Porter Stemmer. They scored the words

Pak and Paroubek used Twitter as a corpus for sentiment analysis and opinion mining [3]. Their method studied the POS tag distribution differences between positive, negative, and neutral tweets. A multinomial Naive Bayes classifier based on POS tags and n-grams was used. They concluded that a Twitter user's emotion is reflected in the syntactic structures of their tweets.

Lu et al. created a framework which automatically constructs a context dependent sentiment lexicon [4]. An unambiguous, gold standard sentiment lexicon is used as the basis. The polarity of these sentiments are propagated into other aspect-word pairs through language clues, a synonym antonym dictionary, and overall review ratings. In conclusion they found the framework could successfully learn new aspect dependent sentiments. Also, the coverage and accuracy of the general lexicon was greatly improved by the sentiment lexicon.

IV. Approach

Searching Twitter for posts containing the keyword "depressed" returns many tweets. Between February 11, 2011 and March 9, 2011 over 247,000 tweets were returned by continuously querying Twitter for "depressed." A large portion of this data is not relevant to the analysis of mental health expression. Only a subset of these tweets contain examples of users sincerely expressing their depression. The initial task is to isolate at least 1000 of these relevant tweets in which users seem to be expressing the emotional impact of depression.

The next step will be to score the set of at least 1000 expressive tweets. Scoring will be done by a group of human judges. The tweets will be scored along the six dimensions of mood that are used in the Profile of Mood States scoring. It is important to ensure the validity of the scoring by confirming that the human judges are all scoring with like minds. A sample of tweets will be presented to the judges to see how well their POMS scores agree with one another. If their POMS scoring is in agreement, then POMS should be a good measure to score the emotions and moods expressed in tweets.

The collection of POMS scored tweets will then be lexically analyzed. Through analysis a lexicon of common words and syntactic groups of words could be shown to be common features of these tweets. The weights of words in the lexicon could correspond to the POMS scores of their containing tweets. This lexicon of features would be used to help accurately identify similar tweets. The similarity of these tweets could be determined by scoring them on how relevant their words are to the content in the lexicon.

Once emotionally expressive tweets can be selected for with some accuracy, then Twitter users expressing such emotions will be monitored over time. The idea is to observe users who are continuously and habitually writing tweets that are related to their depression or negative feelings. Studying this type tweeting behavior could give an understanding about what effect users experience by sharing such emotions publicly on Twitter. By studying them over time it would be possible to see fluctuations in their sentiments.

One other potential angle of approach would be to observe the networks that Twitter users are embedded in. A twitter user's network would include those that she follows or her followers. Perhaps a user who is continuously tweeting about depression and negative sentiments is influenced by their peers on Twitter. The idiom “misery loves company” could be a phenomenon among groups of followers on Twitter. Alternatively, users might express feelings of depression as a way to call for help and support from their friends. Either way, there is a lot that can be learned about the group interaction and the role it plays in mental health expression on Twitter.

A. Extending POMS

The POMS test consists of sixty-five mood related adjectives. Each of these sixty-five words are related to one of this six dimensions of mood. The purpose of the POMS list of words is to find the words in tweets. For example, if the word ”angry” is contained in a tweet, then an anger-hostility mood is probably expressed in the tweet. However, the sixty-five words have limited coverage of the wide range of words that can be expressive of mood. The POMS list of words will be extended with WordNet synonyms. The extended list should contain more words that are representative of the six dimensions of mood.

B. Scoring Tweets

The goal of scoring tweets is to score them along the six dimensions of mood expressed in POMS. The two options for scoring methods are automatic scoring and human judge scoring. The automated scoring will be done by using the extended POMS word list. For each tweet, a six dimension POMS mood vector will be calculated. Each vector will be of the form tension-anxiety, anger-hostility, depression-dejection, confusion-bewilderment, fatigue-inertia, vigor-activity. If a tweet contains a word from the extended POMS list, then the respective dimension in its mood vector will be incremented.

The human judge scoring will involve scoring individual tweets on a scale of one to five along the six mood dimensions. Compared to the automatic scoring, human judges will be able to spot subtle expression of mood in the text.

C. Lexicon Construction

The Twitter mood lexicon will be a set of features that are most common in tweets. The features will be weighted based on their POMS scores. The feature weights could be based on a combination of the automatic scores and human scores. For example, all the words in a tweet scored as ”anger-hostility” would contribute to their ”anger-hostility” weight in the lexicon.

V. Future Work

There are two major possible applications for the Twitter mood lexicon. Understanding the mood trends in a population. Secondly, understanding an individual's mood. Investing an individuals mood could be most valuable when studied over a period of time.

VI. Acknowledgment

NSF grants CNS-0958576 and CNS-0851783 have funded the research reported in this paper.

References

[1] J. Bollen, A. Pepe, and H. Mao, “Modeling public mood and emotion: Twitter sentiment and socio-economic phenomena,” *CoRR*, vol. abs/0911.1583, 2009.

[2] A. Pepe and J. Bollen, “Between conjecture and memento: shaping a collective emotional perception of the future,” *CoRR*, vol. abs/0801.3864, 2008.

[3] A. Pak and P. Paroubek, “Twitter as a corpus for sentiment analysis and opinion mining,” in *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*, N. C. C. Chair), K. Choukri, B. Maegaard, J. Mariani, J. Odijk, S. Piperidis, M. Rosner, and D. Tapias, Eds. Valletta, Malta: European Language Resources Association (ELRA), may 2010.

[4] Y. Lu, M. Castellanos, U. Dayal, and C. Zhai, “Automatic construction of a context-aware sentiment lexicon: an optimization approach,” in *Proceedings of the 20th international conference on World wide web*, ser. WWW '11. New York, NY, USA: ACM, 2011, pp. 347–356. [Online]. Available: http://doi.acm.org/10.1145/1963405.1963456

Implementation of Soft Keyboards for Indic Languages

Albert Brouillette

***Abstract*—The arrangement of letters on a keyboard determines the ease and efficiency of text input. On devices with limited space, the keyboard layout can have an even greater impact on effective data entry. Much research has been done proposing techniques for optimizing Roman-alphabet keyboards, including some for small devices. However, the large number of letters in other alphabet systems makes this problem more complex. Some alphabets can have as many as three times the number of characters as English. This paper investigates techniques for reducing the size of Indic keyboards while creating an optimized layout. To facilitate this, we propose the implementation of machine learning techniques such as the genetic algorithm in developing optimized soft keyboards for mobile phones.**

***Index Terms*—Soft keyboards, Indic languages, optimization, genetic algorithms, Android development.**

I. Introduction

As technology has progressed over the past several decades the world has gained the ability to process a large amount of information in a short time. With these developments comes the need for keyboards that allow users to input text more efficiently. This need for faster keyboards has been the focus of much research in recent years. While much progress has been made in the development of optimized keyboards for Roman-alphabet based languages, there has been little work done with languages based on other alphabets. As a result, many Indic languages have only rudimentary, unoptimized keyboards. For the most part, these keyboards have been inefficient and difficult to use.

The use of soft keyboards would allow users to input information without the actual existence of a physical keyboard. The concept of the soft keyboard is that data can be input through mouse clicks on an on-screen keyboard, or through a touch-screen device[1]. In addition, the virtual keyboards can be mapped to receive input from a standard physical keyboard. The development of efficient soft keyboards is becoming an increasingly attractive alternative for numerous applications.

Many of the current keyboards for Indic languages have been developed around some form of the QWERTY-based layout. While these keyboards can be functional, the greater number of characters in the Indic languages make them cumbersome and inefficient. Since soft keyboards do not have any physical limitations, they can easily be modified and programmed to reach a much more reasonable solution. These soft keyboards can then be adapted and customized for specific applications and devices.

At its root, the primary goal in any keyboard optimization is simply allowing a user to choose the desired characters as quickly as possible. In optimizing keyboards for Brahmic scripts the most important obstacle to overcome is the large number of characters in the languages. In an Indic language, there can be well over 60 individual characters which can be combined to form over 200 different ligatures. Many of these ligatures bear only slight resemblance to the original characters. Including every character and ligature would be highly impractical because of its large size and the difficulty in finding a specific letter. However, the opposite extreme, a small keyboard with only vowels and consonants, would be similarly unreasonable since it would require several characters to be chosen at a time. An optimal solution would logically involve a compromise between these extremes.

A. Brouillette is with the Department of Computer Science, University of Colorado, Colorado Springs, CO, 80918 USA e-mail: (abrouil2@uccs.edu).

II. Related Research

Most of the previous research in the area of keyboard optimization has focused on optimizing English soft keyboards. While there has not been extensive research specifically for optimizing Indic keyboards, the research into English keyboards is valuable in finding techniques for optimizing any keyboard.

A. Keyboard Optimization

The earliest soft keyboards focused on variations of alphabetic and QWERTY layouts. While these keyboards were effective for the technology at the time, many of the layouts were apparently arbitrary attempts at organizing the characters to conform to mechanical limitations. Since then, much progress has been made, starting with MacKenzie's development of one of the first character-frequency optimized layouts, the OPTI keyboard[1]. His approach was essentially an application of Fitts' law with a trial and error approach to hand placing the characters based on frequently occurring bigraphs. Those results were improved through the use of algorithms adapted from applications of physics such as Hook's Law. The use of the *Metropolis random walk algorithm* has further increased the efficiency of soft keyboards[2]. The use of this algorithm with machine testing has enabled the development of English keyboards with theoretical top typing speeds of up to 42 wpm. More recently, soft keyboards have reached a new level of efficiency through the development and use of genetic algorithms[3]. These layouts have so far produced the best results for optimal keyboards. Because of the effectiveness of this algorithm, it is thought that similar optimization techniques might be developed to further improve these keyboards. While not widely used, ant colony optimization has been implemented in developing keyboards for handicapped users[4]. In comparison, research into the development of soft keyboards for Indic languages is relatively primitive. Most examples seem to be designed simply for utility with no thought toward efficiency. Some work has been done in designing single layered keyboards similar to the English layouts[5]. However, there are some problems with this approach, due to the greater number of characters in these languages. In working to optimize keyboards for Brahmic scripts, the techniques developed for English keyboards are inadequate and need to be modified. Recently it was proposed that the development of layered keyboards would give a better solution to the difficulty of a large number of necessary characters[6][7][8]. At this point, this approach seems to give the best results. These keyboards use pop-up menus to allow users to access multiple characters from a single key. This technique could potentially increase users input speed by reducing the search time and distance between each character.

B. *Adaptations for Cell Phones*

Gong has done considerable research in optimizing English keyboards for cell phones. One of the primary techniques used for optimizing cell phone text input has been the idea of putting multiple letters on each key. Letters are selected either by pressing a key multiple times or by using a word-prediction program to allow the selection of possible words from a given combination of key-presses[9][10].

In our project, we have used a combination of these word-prediction techniques with the layered keyboard research in our efforts to reach an optimal Indic keyboard for mobile devices.

III. OPTIMIZING KEYBOARDS

A simplistic approach for keyboard optimization would be to simply use an exhaustive search and evaluate every possible keyboard layout. However, for any given number of characters n, there are $n!$ possible combinations of the characters. For languages of 60+ characters, this number quickly reaches 10^{100}, making this approach impossible.

So far the best results for keyboard optimization have been reached through the use of machine learning techniques. Our approach to optimization in this project involves the use of the Genetic Algorithm to generate optimized keyboard layouts.

A. *Specific Design Decisions*

The creation of these keyboards required some arbitrary decisions to be made from the start. These decisions are explained here.

Based on the results of earlier research it was decided that the space-bar should be placed below the keyboard layout. The distance from a given character to the space-bar is calculated as the distance to the center of the space-bar. This compensates for the fact that users will not necessarily choose the shortest distance every time. Although, theoretically, multiple optimized space-keys would give better results, the results of experiments done by Zhai et al. showed that, given the choice of four space-bars, users chose the optimum space bar only 38-47% of the time. As an extra benefit, an arbitrarily placed space-bar prevents inaccuracies in calculations due to "free-warping", a common error in keyboard evaluation where the stylus enters the space-bar in one location and leaves in an unrelated random location[2]. Finally an easily accessible space-bar improves the users ease of learning a layout.

Another decision was to use a rectangular 'grid' keyboard layout with each character occupying a square key. While some other designs such as the Metropolis Keyboard have used a hexagonal, honeycomb design, this rectangular layout is the most familiar to users[2]. Additionally, this layout and simplifies the comparison of our test results with results from previous research which uses this layout.

B. *Implementing the Genetic Algorithm*

The genetic algorithm is a heuristic designed to imitate the concept of natural selection. This algorithm is used to optimize a function for which there might not actually be an optimal solution. The general idea behind the genetic algorithm is to create a random population of potential solutions. These solutions are combined and mutated with the objective of keeping the 'good' parts of each solution and combining them toward a theoretical 'optimum'.

There are three essential parts of this algorithm. First, there is a population of potential solutions. These possible solutions then need to be evaluated using a fitness function. Finally there needs to be a method of reproduction that will change the population of solutions over time. In any given generation, the individual solutions are evaluated by the fitness function and assigned a score. This score is based on its distance from the theoretical optimum solution. A new population is created including a percentage of the best solutions and adding some new solutions made by combining and mutating the best individuals. This process is repeated until an acceptable maximum is reached.

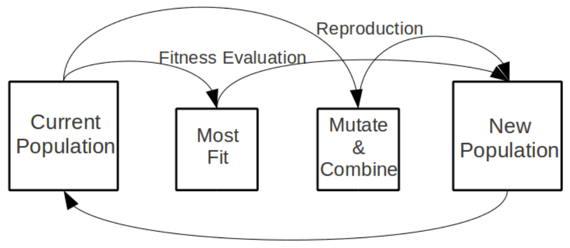


Fig. 1. Illustration of the evolutionary cycle of a population of chromosomes in a genetic algorithm.

In our implementation of the genetic algorithm, each individual, known as a chromosome, represents a different keyboard layout. Each of these chromosomes holds a number of genes equal to the number of characters in the given alphabet. Each character in the alphabet is assigned an integer. Each of the genes then contains one of these integers.

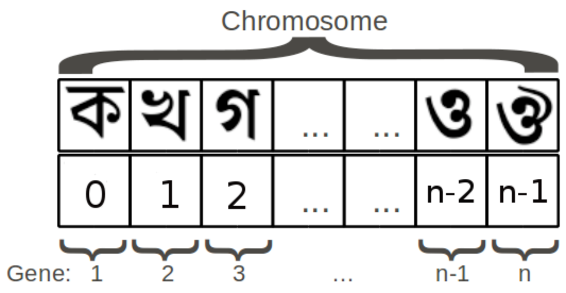
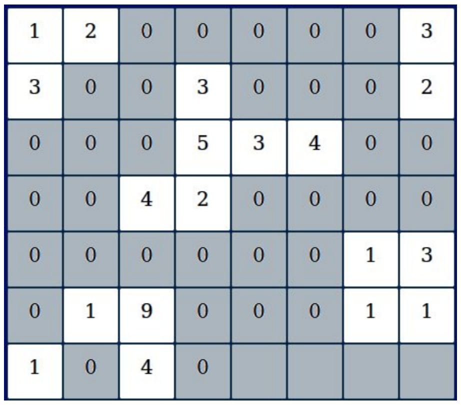


Fig. 2. Illustration of a chromosome for the Assamese keyboard. The genes are stored in a one dimensional list of integers while the genetic algorithm is running. Once finished, it is converted to an $x \times y$ rectangular keyboard layout with the first x genes representing the first row of keys, the second group of x genes represents the second row, etc.

The chromosomes are scored with the highest score being given to the layout with lowest mean time per character. This way the fastest layouts are kept for the next generation. Some of the chromosomes are then randomly selected to be combined or mutated. These new chromosomes are included with the fastest layouts in the next generation.

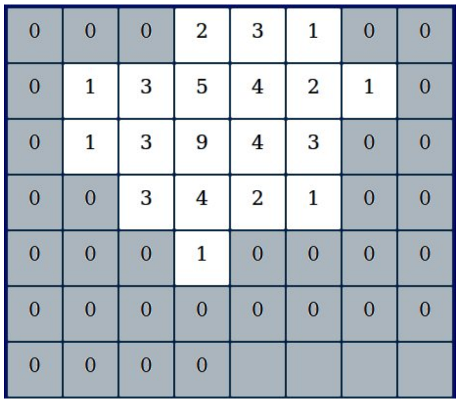
In testing the effectiveness of this algorithm, we used a diagram to track the positions of the most frequently occurring characters on the keyboard. In the first generation, the high frequency characters were spread randomly across the layout.



1	2	0	0	0	0	0	3
3	0	0	3	0	0	0	2
0	0	0	5	3	4	0	0
0	0	4	2	0	0	0	0
0	0	0	0	0	0	1	3
0	1	9	0	0	0	1	1
1	0	4	0				

Fig. 3. A diagram representing the layout of the most frequently used characters in the first generation of the genetic algorithm. The characters are represented by the numbers from 0-9 with 0 being the lowest frequency and 9 being the highest.

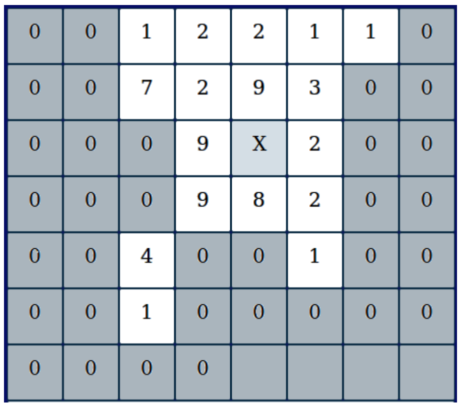
However, after 400 generations, it became apparent that the most frequent characters were being clustered together near the center of the keyboard.



0	0	0	2	3	1	0	0
0	1	3	5	4	2	1	0
0	1	3	9	4	3	0	0
0	0	3	4	2	1	0	0
0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0				

Fig. 4. A diagram representing the layout of the most frequently used characters after 400 generations of the genetic algorithm. The characters are represented by the numbers from 0-9 with 0 being the lowest frequency and 9 being the highest.

Logically, it can be expected that most of the frequently occurring digraphs consist of combinations of the most frequent characters. A diagram tracking the position of the most frequent character in relation to its most common digraphs shows the digraphs getting closer together as the algorithm progresses.



0	0	1	2	2	1	1	0
0	0	7	2	9	3	0	0
0	0	0	9	X	2	0	0
0	0	0	9	8	2	0	0
0	0	4	0	0	1	0	0
0	0	1	0	0	0	0	0
0	0	0	0				

Fig. 5. A diagram representing the layout of the most frequently occurring character in relation to its most common digraphs after 400 generations of the genetic algorithm. The character is represented by an X its digraphs are represented by the numbers from 0-9 where 0 is the least common and 9 is the most common.

These results are consistent with the theoretical optimal positioning of the frequently used characters and their digraphs close together. Based on test results by earlier research, it is expected that this pattern will continually be improved with testing at a greater number of generations.

C. Parameter Changes

There are several parameters in the genetic algorithm that can be adjusted to achieve optimal results for a given application. These can include changes to the population size as well modifications to the mutation rates and crossover functions.

In determining the best parameters for the genetic algorithm, we performed several tests with the Bengali language, varying the population size. The result of these tests showed that simply creating larger populations will not always give the best results. The best keyboards were generated with a more moderate population size.

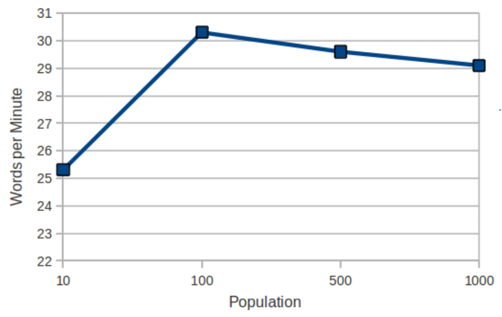


Fig. 6. This graph shows the variation in typing speed with changes in population. The typing speed was calculated after 100 stable generations.

Further testing showed that looking for a greater number of stable generations gave the best indication of an optimal keyboard. Stable generations are defined as generations with a consistent highest score. Populations with a large number of stable generations consistently generated the best keyboards.

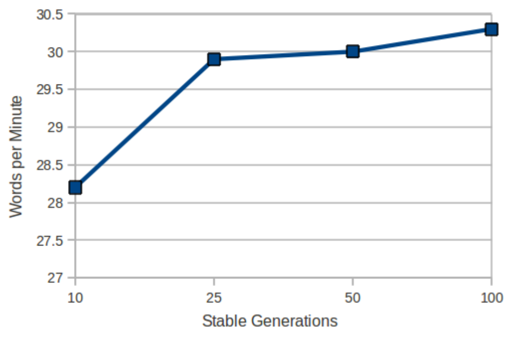


Fig. 7. This graph shows the change in typing speed after different numbers of stable generations. The typing speed was calculated after populations of 100 chromosomes.

IV. Evaluation of Generated Keyboards

As keyboards are developed, they need to be evaluated in order to compare them and determine how much improvement has been made. By definition, an efficient keyboard is one that allows users to input their text as quickly as possible. Although human testing is necessary to determine the actual effectiveness and learn-ability of a keyboard, the first step is to compute theoretical upper and lower limits of typing speed. The upper-bound is commonly calculated by using Fitts' Law. This number gives us an estimate of the typing speed for an experienced user with minimal search time on each character. The lower-bound can also be estimated using a combination of Fitt's Law and the Hick-Hyman Law. The result of this calculation estimates the decision time for new users.

A. Fitts' Law

Fitts' Law models human movement and can be used to predict the time required to move to a given target point. For our application, we use this as a technique for determining the average time in seconds the enter a character. This number can then be used to calculate a theoretical upper-bound in words per minute of input for the keyboards. This is the most widely used method for evaluating English keyboards. An adaption of Fitts' Law has been made in order to find the average time required to move between two characters, i and j, for a given alphabet of n characters. This is done by looking at the distance between the characters, D_{ij}, as well as the frequency of occurrence for that particular digraph, P_{ij}. After assigning a key width W_j, and an index of performance IP, the equation for Fitts' Law becomes:

$$\bar{t} = \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{P_{ij}}{IP}\left[\log_2\left(\frac{D_{ij}}{W_j}+1\right)\right]. \quad (1)$$

We will use an IP of 4.9, as determined by earlier research, in order to maintain a consistent comparison of our progress[1].

In its most basic state, Fitts' Law will only work for single layered keyboards. However it has since been adapted to return reasonable results for other layouts as well, including multi-layered and menu based keyboards[11].

B. Hick-Hyman Law

The Hick-Hyman Law is used to predict the time required for a human to make a decision based on a given number of choices. In our application, this number is used to estimate the typing speed for a novice user who would need to search the keyboard to determine their next key-press. This number gives us a more realistic estimate to compare to human testing results.

In using the Hick-Hyman Law to evaluate our keyboards, we are given n as the number of possible choices (in our case the number of keys) and p_i being the probability of a given key being selected. The number b is a constant that is determined through experimentation. This reaction time can then be described by the equation:

$$T = b\sum_{i=1}^{n} p_i \log_2\left(\frac{1}{p_i}+1\right). \quad (2)$$

The Hick-Hyman Law can be adapted to estimate the decision time for both the base layer of the keyboard and the hierarchical menus.

C. WPM calculation

In calculating an average number of words per minute, the corpus of words is processed to determine the average number of characters per word. From our corpus of the Assamese language this number turns out to be approximately 6 characters per word. Given the calculated mean time per character, the calculation for average words per minute is simply: $wpm = \frac{60}{6t}$

V. Testing on Other Indic Languages

In this section, we discuss our the results we have obtained while developing soft keyboards for several other Indian languages. The languages we work with are Bengali, Hindi, Gujarati, Punjabi, Oriya, Kannada and Telugu. Of all the languages we have investigated for this paper, the scripts used by Assamese, Bengali, Hindi, Gujarati, Punjabi and Oriya belong to the Northern branch of Brahmic scripts. Assamese and Bengali use two variants of the Eastern Nagari script. Hindi uses Devanagari script. Punjabi uses the Gurmukhi script although it can be written using the Shamukhi script as well. Gujarati and Oriya have their own individualized scripts. Two of the languages, Kannada and Telugu use scripts that belong to the Southern branch of Brahmi scripts. Each of these languages have its own script.

For each language, we develop alphabetically sorted keyboards, a flat GA-based soft keyboard and a layered GA-based soft keyboard using the techniques we used for Assamese.

A. Bengali

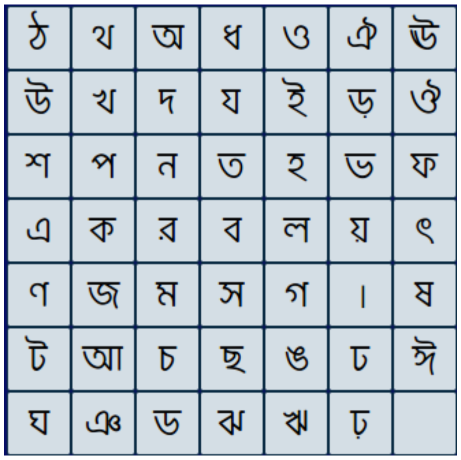
For Bengali, each alphabetic layout tested for WPM results listed the vowels before the consonants in alphabetical order. The row ordering of three alphabetic layouts were tested: one with the diacritics listed after the vowels and consonants, one with the diacritics listed in between the vowels and consonants, and one with the diacritics listed before the vowels and diacritics. Row ordering means that each of the characters was listed alphabetically left to right from the top row to the bottom row. The best arrangement was row ordered and listed the diacritics after the vowels and consonants, which yielded an expected input speed of 22.19 WPM.

Diacritic Arrangement	Words per minute	Time per char
First-row diacritic	20.4	0.490
Center-row diacritic	21.0	0.476
End-row diacritic	22.2	0.451

Fig. 8. Variations in input speed for alphabetic layouts with different arrangements of diacritics.

The best results of our genetically designed Bengali flat keyboard yielded a theoretical input speed of 30.35 WPM as predicted by Fitts Law. This was an 8 x 8 square keyboard constructed with the following genetic algorithm parameters: a population of 100 and 100 stable generations.

As discussed earlier in the paper, greater improvements to input speeds can be achieved through the use of layered keyboards. Thus for Bengali, we kept the consonants and vowels on a base layer and a diacritic menu as a second layer that could come up whenever a user clicked on a consonant as done earlier for Assamese. The best genetically designed layered keyboard for Bengali had a base layer of dimensions 7x6 and was genetically constructed from a population of 500 and 15 stable generations. It yielded an expected input speed of 36.13 WPM. Adding a vowel menu in addition to the diacritic menu for the layered keyboard made very little difference for the layered keyboard. For Bengali, taking the vowels out of the base layer of consonants and putting them in their own separate menu decreased the expected speed by only 0.04 WPM.



ঠ	থ	অ	ধ	ও	ঐ	ঊ
উ	খ	দ	য	ই	ড়	ঔ
শ	প	ন	ত	হ	ভ	ফ
এ	ক	র	ব	ল	য়	ৎ
ণ	জ	ম	স	গ	।	ষ
ট	আ	চ	ছ	ঙ	ঢ	ঈ
ঘ	ঞ	ড	ঝ	ঋ	ঢ়	

Fig. 9. The best layered Bengali keyboard. This keyboard was designed using the genetic algorithm with a population of 500 and 15 stable generations.

B. Hindi

For Hindi, the same six alphabetic layout arrangements as in Bengali were tested with Hindi characters. The layout with the best expected input speed, which was 22.04 WPM, listed the diacritics before the vowels and consonants and was column ordered.

The best theoretical input speed generated from the genetically designed Hindi flat keyboards was 29.01 WPM as predicted by Fitts Law. This was also an 8 x 8 square keyboard constructed with a population of 100 and 100 stable generations. The same diacritic menu was made for the Hindi layered keyboard using Hindi characters. The best genetically designed layered keyboard for Hindi had a base layer of dimensions 7 x 5 characters and was genetically constructed from a population of 500 and 25 stable generations. It yielded an expected input speed of 37.03 WPM. Both of these layered keyboards offered an average 6.9 WPM improvement over the expected WPM scores of the flat keyboard layouts and an average 14.47 WPM improvement over the WPM scores of the alphabetic layouts. For Hindi, having a vowel menu decreased the expected input speed by only 0.7 WPM.

Future research may include testing the layered keyboard layouts with and without a vowel menu on actual users to determine whether a separate vowel menu can significantly improve or worsen the efficiency of a layered keyboard for the Bengali and Hindi languages.

Keyboard Type / Language	Flat Alphabetic (WPM/CPM)	Layered Alphabetic (WPM/CPM)	Flat GA-Designed (WPM/CPM)	Layered GA-Designed (WPM/CPM)
Assamese	25.1 / 0.399	33.9 / 0.295	34.2 / 0.292	40.2 / 0.249
Bengali	22.7 / 0.440	26.6 / 0.377	30.3 / 0.330	36.1 / 0.277
Hindi	25.8 / 0.465	34.5 / 0.348	34.4 / 0.349	43.7 / 0.275
Gujarati	22.2 / 0.449	24.5 / 0.407	29.8 / 0.335	31.7 / 0.315
Punjabi	26.5 / 0.453	29.9 / 0.402	34.9 / 0.343	39.9 / 0.300
Oriya	16.7 / 0.450	19.4 / 0.386	23.5 / 0.319	26.7 / 0.281
Kannada	14.7 / 0.455	17.1 / 0.386	20.2 / 0.330	22.8 / 0.292
Telugu	15.8 / 0.474	19.1 / 0.393	22.7 / 0.331	25.6 / 0.294

Fig. 10. Expected Upper Bound of Input Speeds in WPM/CPM for Various Languages. The numbers were computed using Fitt's Law only.

Keyboard Type / Language	Flat Alphabetic (WPM/CPM)	Layered Alphabetic (WPM/CPM)	Flat GA-Designed (WPM/CPM)	Layered GA-Designed (WPM/CPM)
Assamese	9.70 / 1.031	13.3 / 0.754	10.8 / 0.924	14.1 / 0.708
Bengali	9.42 / 1.061	12.5 / 0.800	10.5 / 0.956	14.3 / 0.700
Hindi	10.8 / 1.111	15.4 / 0.782	12.1 / 0.992	17.5 / 0.685
Gujarati	9.24 / 1.082	11.9 / 0.838	10.1 / 0.986	13.2 / 0.757
Punjabi	11.0 / 1.092	14.0 / 0.857	12.0 / 0.997	15.7 / 0.764
Oriya	6.99 / 1.073	9.24 / 0.811	7.73 / 0.970	9.44 / 0.795
Kannada	6.06 / 1.101	7.75 / 0.860	6.85 / 0.974	8.44 / 0.790
Telugu	6.87 / 1.092	8.84 / 0.848	7.69 / 0.976	9.89 / 0.758

Fig. 11. Expected Lower Bound of Input Speeds in WPM/CPM for Various Languages. The numbers were computed using Fitt's Law and Hick-Hyman's Law.

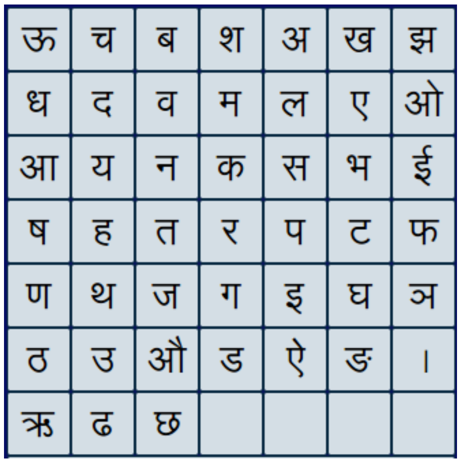


Fig. 12. The best layered Hindi keyboard. This keyboard was designed using the genetic algorithm with a population of 500 and 25 stable generations.

C. Other languages

We tested our algorithm with five other languages in order to detect some trends in their development and draw conclusions about the variations in input speed. Each language was tested with four different keyboard arrangements. Our first step was to evaluate keyboards with an unoptimized, alphabetic arrangement as a basis for comparison. These keyboards were developed with both flat and layered designs. We then used the genetic algorithm to develop optimized flat and layered keyboards.

1) Gujarati: For the Gujarati language, our corpus had an average of approximately 6 characters per word which we used to calculate words per minute from the average time per character. The alphabetic layouts yielded an upper-bound of 0.449 seconds per character or 22.2 WPM for the flat keyboard and 0.407 seconds per character or 24.5 WPM with the layered keyboard. After optimization using the Genetic Algorithm, the results improved to 0.335 seconds per character or 29.8 WPM for the flat keyboard and 0.315 seconds per character or 31.7 WPM with the layered keyboard.

2) Punjabi: For the Punjabi language, our corpus had an average of approximately 5 characters per word which we used to calculate words per minute from the average time per character. The alphabetic layouts yielded an upper-bound of 0.453 seconds per character or 26.5 WPM for the flat keyboard and 0.395 seconds per character or 30.4 WPM with the layered keyboard. After optimization using the Genetic Algorithm, the results improved to 0.343 seconds per character or 34.9 WPM for the flat keyboard and 0.300 seconds per character or 39.9 WPM with the layered keyboard.

3) Oriya: For the Oriya language, our corpus had an average of approximately 8 characters per word which we used to calculate words per minute from the average time per character. The alphabetic layouts yielded an upper-bound of 0.450 seconds per character or 16.7 WPM for the flat keyboard and 0.386 seconds per character or 19.4 WPM with the layered keyboard. After optimization using the Genetic Algorithm, the results improved to 0.319 seconds per character or 23.5 WPM for the flat keyboard and 0.281 seconds per character or 26.7 WPM with the layered keyboard.

4) Kannada: For the Kannada language, our corpus had an average of approximately 9 characters per word which we used to calculate words per minute from the average time per character. The alphabetic layouts yielded an upper-bound of 0.455 seconds per character or 14.7 WPM for the flat keyboard and 0.386 seconds per character or 17.1 WPM with the layered keyboard. After optimization using the Genetic Algorithm, the results improved to 0.330 seconds per character or 20.2 WPM for the flat keyboard and 0.292 seconds per character or 22.8 WPM with the layered keyboard.

5) Telugu: For the Telugu language, our corpus had an average of approximately 8 characters per word which we used to calculate words per minute from the average time per character. The alphabetic layouts yielded an upper-bound of 0.474 seconds per character or 15.8 WPM for the flat keyboard and 0.393 seconds per character or 19.1 WPM with the layered keyboard. After optimization using the Genetic Algorithm, the results improved to 0.331 seconds per character or 22.7 WPM for the flat keyboard and 0.294 seconds per character or 25.6 WPM with the layered keyboard.

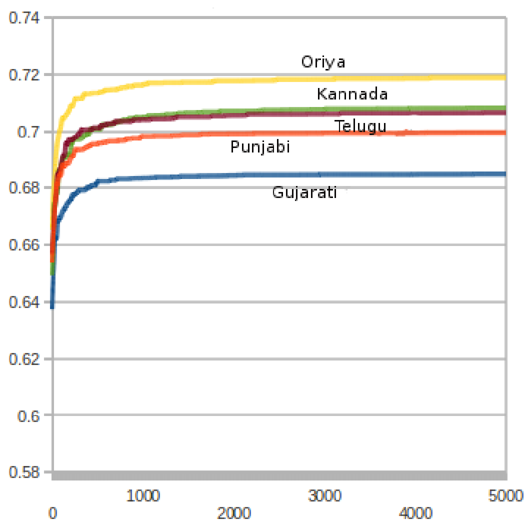


Fig. 13. A graph of the improvement in input speed for the 5 languages over 5000 generations.

D. Summary

In analyzing these test results, we were able to draw a few basic conclusions regarding the optimization process.

Looking at the numbers, we notice that the Oriya language shows the greatest improvement after optimization, giving the lowest time per character, while the improvement for the Gujarati language was somewhat less significant. One explanation for these results considers the relative frequencies of characters in the two languages. The most frequently occurring character in the Oriya language has a relative frequency of approximately 8.6%. In Gujarati, the most frequent character has a relative frequency of 5.9%. Considering Figure 4 we can see that the optimized keyboard has the most frequent characters clustered together near the center. Our conclusion is that languages with a small number of high frequency characters have a greater potential to be optimized. It can be surmised that languages with characters that have nearly equal frequencies require the user to travel a greater average distance between each character.

Additionally, it can be seen that Gujarati has a smaller improvement between its layered and flat layouts when compared to the other languages. One reason for this could be the higher frequency of the diacritics in this language. The Kannada language, which showed the highest improvement in its layered layout, also has the lowest frequency of diacritics. Essentially, the smaller number of diacritics means less use of the menu making the layering more effective.

Other than these minor variations, the results from the optimization tests of these languages are all very similar in their progress. Graphing the results of the tests all of the languages show very similar improvement over an increasing number of generations.

VI. Android Keyboard Design

As we worked to develop Android soft keyboards from our optimized layouts we had several challenges to overcome first. Our first step was to find a suitable format to install the keyboards on the Android phone. Once we had actually tested the keyboards in their basic form, we did several experiments in an attempt to modify the designs for more practical use on the actual device.

A. General Keyboard Framework

There has been considerable research done in the development of frameworks for soft keyboards on Android phones and there have been many keyboard applications created. The tool we are using for this project is the AnySoftKeyboard app[1]. This is a free application that is easily available to users. In addition, it supports the development of plug-ins for other languages. Using this tool allows us to quickly test our keyboard layouts and make them available on-line for human testing.

Using this tool as our structure, we have created an AnySoftKeyboard plug-in for the Assamese language. This keyboard was designed using the genetic algorithm test code.

[1] $https://market.android.com$

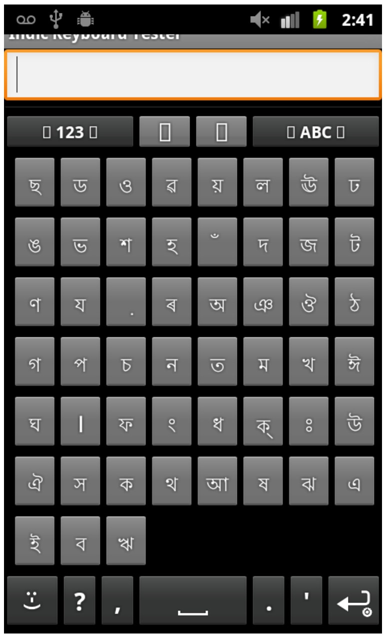


Fig. 14. This Assamese keyboard was implemented as an AnySoftKeyboard plug-in. It was designed by the genetic algorithm after 50 generations. The diacritics are placed in a pop-up menu.

The keyboard was designed to be used with a diacritic layer. This diacritic layer is implemented as a menu that pops up when a consonant key is held down.

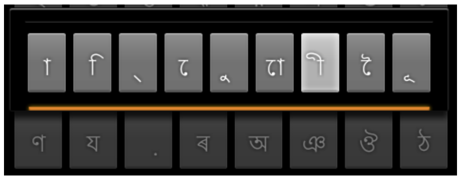


Fig. 15. The diacritic pop-up menu. The menu appears when a consonant key is held down.

An additional feature supported by AnySoftKeyboard is predictive test entry. We implemented this by running a program to process the corpus and create a list of words and their frequencies. This data was then used to create the binary dictionary used by the AnySoftKeyboard program. The word predictions appear as a menu at the top of the screen after at least two characters have been entered.

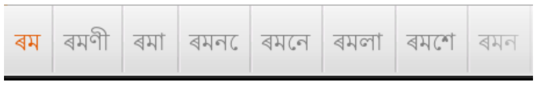


Fig. 16. The word prediction menu. This menu appears after at least two characters have been entered.

B. Experiments With Keyboard Dimensions

Our initial keyboard designs were functional on the Android device. However, the dimensions of the keyboard ended up obstructing parts of the text entry. In an effort to make the keyboard more practically usable, we tried several different variations in the dimensions of the keyboard. The best layout we found consisted of longer, more narrow format.

Our original design consisted of a nearly square layout of 8×8 or 8×7 keys. We ran the genetic algorithm and tested a layered keyboard with dimensions of 5×10 keys. The results of these tests showed only a minimal reduction in typing speed, while the ease of using the keyboard was greatly improved. Running a test over 500 generations, we got results of 36.9 WPM for the rectangular keyboard compared to the 40.2 WPM with the square layout.

The main side-effect of this layout is the key-size in this format. In order to position the keys in this layout we had to reduce their size. This reduces the accuracy for most users and results in more errors.

C. Text Input From Two Points

After running tests with rectangular keyboards, we experimented with modifying the genetic algorithm program to optimize for text input with two fingers.

The basic layout we chose consisted of a long rectangular shape with the assumption that each finger or input point occupying an essentially square section of the keyboard. The logic behind the implementation of the genetic algorithm relies on the assumption that each input point is only used in its square.

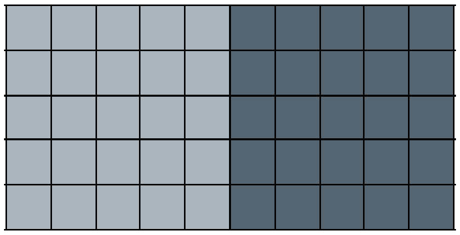


Fig. 17. Basic shape of keyboard for two finger input. Each finger is responsible for pressing the keys in one of the squares.

Because of the complexity of this problem, at this point we have only been able to roughly estimate a lower bound for text entry speed. There are essentially two cases to be considered in the calculation of the fitness function: First the case of digraphs that consist of two characters in the same square. And secondly, the case of digraphs where the characters occur in different squares.

To take care of the first case, the fitness function uses Fitt's Law to calculate an average time per character for each input point in each square. In the second case, it is assumed that each character requires a decision time calculated using the Hick-Hyman Law as well as the time for the input point to move to the desired key.

Using this algorithm, we were able to develop a two input keyboard for the Assamese that we estimate to have a lower bound of text input around 22.6 WPM.

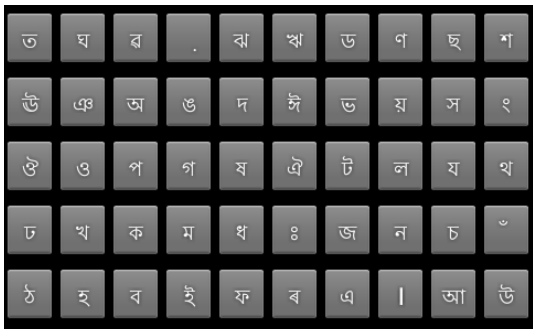
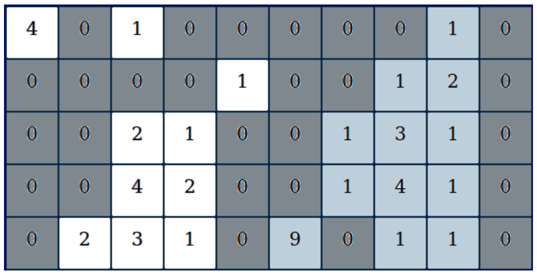


Fig. 18. Android keyboard designed for two finger input. This keyboard has a lower bound of text input estimated to be 22.6 WPM.

Looking at the locations of the most frequent characters we can see two clusters being form in the location of the two input points. There seems to be a nearly equal number of high frequency characters on each side.



4	0	1	0	0	0	0	0	1	0
0	0	0	0	1	0	0	1	2	0
0	0	2	1	0	0	1	3	1	0
0	0	4	2	0	0	1	4	1	0
0	2	3	1	0	9	0	1	1	0

Fig. 19. A diagram representing the layout of the most frequently used characters after optimization by the genetic algorithm. The characters are represented by the numbers from 0-9 with 0 being the lowest frequency and 9 being the highest. The frequently used characters appear to be forming two clusters around the two input points.

These results are consistent with a comment made in a paper by Zhai et al.[12]. In this paper, they mention that the QWERTY keyboard is most effective when used for two-handed input because of the frequency of alternation between the two hands. The algorithm that we implemented gives a higher score to keyboards that more frequently alternate hands.

D. Future Work

The optimized keyboards that we generated for the Android phone performed considerably better than the alphabetic alternatives. However, even after adding the diacritic layer, the number of keys is simply to large for the size of the Android phone. An area for future research would be to experiment with putting multiple characters on each key. It would be desirable to be able to optimize keyboards given a physically constrained number of keys. This might be the best option for creating effective cell phone keyboards for languages with large numbers of characters.

Another area for continued research would be to investigate the development of keyboards optimized for the specific needs of other devices such as the iPad and the iPhone.

VII. Conclusion

The versatility of soft keyboards makes them an ideal research tool in seeking optimal layouts for Indic languages. The programs developed to predict the best keyboard layouts can be easily reused to generate optimal keyboard layouts for other languages. Analysis of the keyboards generated by implementing the genetic algorithm was done by using Fitts' Law and the Hick-Hyman Law to estimate input speed. Based on these results, we were able to demonstrate how the efficiency of a keyboard is improved when the keys are arranged based on character and digraph frequencies.

The results of our tests with the various Indian languages show the keyboards developed by the genetic algorithm to be comparable in efficiency for all of the languages tested. It can be assumed that these techniques will be easily adapted to create optimized keyboards for many other languages that have large numbers of characters.

Acknowledgment

The research reported in this document has been funded partially by NSF grants CNS-0958576 and CNS-0851783.

References

[1] S. MacKenzie and S. X. Zhang, "The design and evaluation of a high-performance soft keyboard," *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, pp. 25–31, 1999.
[2] S. Zhai, M. Hunter, and B. A. Smith, "The metropolis keyboard - an exploration of quantitative techniques for virtual keyboard design," *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pp. 119–128, 2000.
[3] M. Raynal and N. Vigouroux, "Genetic algorithm to generate optimized soft keyboard," *CHI'05 extended abstracts on Human factors in computing systems*, pp. 1729–1732, 2005.
[4] S. Colas, N. Monmarché, P. Gaucher, and M. Slimane, "Artificial ants for the optimization of virtual keyboard arrangement for disabled people," pp. 87–99, 2007.
[5] V. Varma and V. Sowmya, "Design and evaluation of soft keyboards for telugu," *ICON 2008: 6th International Conference on Natural Language Processing*, 2008.
[6] A. Rathod and A. Joshi, "A Dynamic Text Input scheme for phonetic scripts like Devanagari," *Proceedings of Development by Design (DYD)*, 2002.
[7] S. Shanbhag, D. Rao, and R. K. Joshi, "An intelligent multi-layered input scheme for phonetic scripts," *Proceedings of the 2nd international symposium on Smart graphics*, pp. 35–38, 2002. [Online]. Available: http://doi.acm.org/10.1145/569005.569011
[8] L. Hinkle, M. Lezcano, and J. Kalita., "Designing soft keyboards for brahmic scripts," *ICON 2010: International Conference on Natural Language Processing*, pp. 191–200, 2010.
[9] J. Gong, "Improved text entry for mobile devices : alternate keypad designs and novel predictive disambiguation methods," *Northeastern University Boston, MA, USA*, 2007.
[10] M. Selander and E. Svensson, "Predictive text input for indic scripts," *Citeseer*, 2009.
[11] S. Matsui and S. Yamada, "Genetic algorithm can optimize hierarchical menus," pp. 1385–1388, 2008.
[12] S. Zhai, P. Kristensson, and B. Smith, "In search of effective text input interfaces for off the desktop computing," *Interacting with Computers*, vol. 17, no. 3, pp. 229–250, 2005.

Named Entity Extraction From the Colloquial Setting of Twitter

Cassaundra Doerhmann
University of Colorado at Colorado Springs,
Colorado

Abstract—**This paper suggests a study of Named Entity Recognition (NER) as it applies to Twitter and strategies that can be used to make NER systems more successful in colloquial settings such as Twitter. Named Entities are named nouns which fall into the catagories following: People, Locations, and Organizations. The strategies explored are useing a text normalizer to shape the text into a format that NER programs can recognize and cross checking classifiers to increase the precision of NER tools.**

I. Introduction

Named Entity Recognition is widely used in many different kinds of natural language processing tasks. Named Entity Recognition (NER) is the process of extracting and organizing the names of people, places, and organizations into groups based on commonality [4]. In the area of natural language proccessing many research projects need to identitfy the named entities in order to extract information and relations from texts. Therefore, this process of identifying named entities is necessary for research in the area of natural language proccessing. These named entities can be found by grammar and capitalization patterns but is improved when machine learning is implemented to capture different phrasing of these sentences. For example terms such as "graduated from", "worked at", and "studied at" all suggest that there is a high probability of the prior word being a name [8]. Furthermore, the personal titles such as "Mr.", "Dr.", and "President" suggest that a named entity follows immediately [2].

TABLE I
Example Sentences

President Obama spoke to the troops today.
Mary graduated from Brown.
John vacationed in Spain for the summer.

TABLE II
Classification of Named Entities

Person	Organization	Location
Obama	-	-
Mary	Brown	-
John	-	Spain

Interest in Named Entity Recognition is growing rapidly because of its overwhelming relation to information extraction and to AI strategies. In this project, we attempt to identify and categorize the named entities from Twitter posts.
Twitter posts, also called tweets, have a maximum length of 140 characters, and users often, due to the limited length, forsake grammar and capitalization rules, replacing grammatically correct phrases for slang. Twitter is a huge source of information and therefore it is necessary to discover a way to make these tweets understandable and extract the named entities which are mentioned in these Twitter posts. This will greatly improve the ability of natural language tools to accurately execute the tasks that they are designed to perform. This will allow for better research in the area of natual language processing, especially when in relation to non-normal texts such as blogs, twitter posts, and other texts not written in standard English. Named entity recognition has been used in Natural Language Proccessing (NLP) before, but the focus of these projects has been primarily on texts which are written in standadrd English.
The task of NLP becomes much harder when the text is not written in standard English. Thus it is necessary to have a new system of Named Entity Recognition which takes into account the non-standard laguage used in these colloquial sources.

II. Motivation

Because of the popularity of social networking in today's society, Twitter is quickly becoming the fastest updated source of news in our world today. Not only is it so quickly updated, but the amount of data held within these tweets can be used as a huge resource. With an average of 200 million tweets a day, Twitter often reports the major happenings of the world before the news has the time to broadcast. Because of the immense amount of information contained within these tweets, they, with a little analysis, could be used for many different research projects. However, analysis of this data could be much faster and more accurate if the text was first normalized and the named entities were identified.
The normalization and named entity recognition of Twitter posts could be beneficiall in many different settings. One of the main applications of NER to the real world is information extraction. Because Twitter is such a large source of data with a huge range of topics, it is a great text to draw from in Information Extraction (IE). NER with a basis of

normalization will make information extraction much simpler to accomplish.

III. Twitter

Twitter is a micro-blogging social networking site which is greatly useful for Information Extraction and other such Natural Language Proccesing tasks because it is a huge database for information written by the average person. These tweets are also a source of information for research because they are posted very quickly after the involved events have occurred. However, this massive amount of data is very hard to analyze because of a few differences between Twitter text and the text of standard English.
Tweets are restricted to 140 characters, and because of this restriction, its users need to express their social goings-on in as few words as possible.
Thus words are often mispelled, either accidentally or to shorten length, acronyms are substitued for phrases, and non-grammatical scentence structures are used instead of those that are conventional. This often thwarts the identification or labeling of the words in these Twitter posts.
Twitter has grown from 5,000 tweets per day, in the opening year of this social micro-blogging site in 2007, to a starteling 200 million tweets per day in 2011. It is because of this massive growth that Twitter is becoming too large of source of data to be ignored by the research of the day.

IV. Problem Definition

There are many Named Entity Recognition tools already in existence which are available on the web. So why not use one of those? Normal NER tools are very ineffective when used on Twitter posts. NER tools depend greatly on sentence structure and context to determine named entities. However, tweets are short in nature and tend to be wildly grammatically incorrect. Because of the little context in tweets and the sloppy sentence structure, a normal NER tool performs poorly, and a different approach must be taken.
The purpose of this project is to test the effectiveness of using a normalizer as a preproccessor to a NER tool. And, if time permits, to consolidate both the normalizer and Named Entity Recognition tools in to one single NER super-tool. The use of a normalizer should increase the effectiveness of the NER tool because, although the context is still very low, the grammatical changes will increase the usability of the sentence structure in the recognition process.
This project also intends to increase the effectiveness of a NER tool through the use of cross-referencing classifiers. Multiple classifiers which are trained on different data, such as CRF classifiers, should allow for more accurate results. These classifiers would be able to increase the Recall and Percision because each classifier would find named entities where the other had missedand each would overlook falsely catagorized tokens where one had been mistaken. Thus more named entities would be able to be pulled from a given set of tweets, while less tokens would be falsly recognized

V. Related Research

A. Named Entity Recognition

Up untill now, the majority of the study of Named Entity Recognition has been in relation to documents on the web. Lui et al. implemented a classifier based approach to NER [12]. They used a combination of both K-Nearest Neighbors (KNN) and Conditional Random Fields (CRF) based classifiers. However, Downey et al. used a statistical model to extract these named entities from the web. This approach out-performed a semi-supervised CRF by 73 percent [4]. While still other methods of functional relations were implemented by Hasegawa et al. by tagging named entities and learning the context behind named entities which occur in a similar phrase [7].

There are three significant measurements which are used to evaluate the effectiveness of a NER program.

- Precision (P): Precision is the proportion of the number of correctly identified named entities to the total number of entities identified (the sum of the number of correctly identified and incorrectly identified named entities).

$$P = \frac{N_{correct}}{N_{correct} + N_{incorrect}} \quad (1)$$

- Recall (R): The recall is the proportion of correctly identified named entities to the total number of named entities (the Key count).

$$R = \frac{N_{correct}}{N_{key}} \quad (2)$$

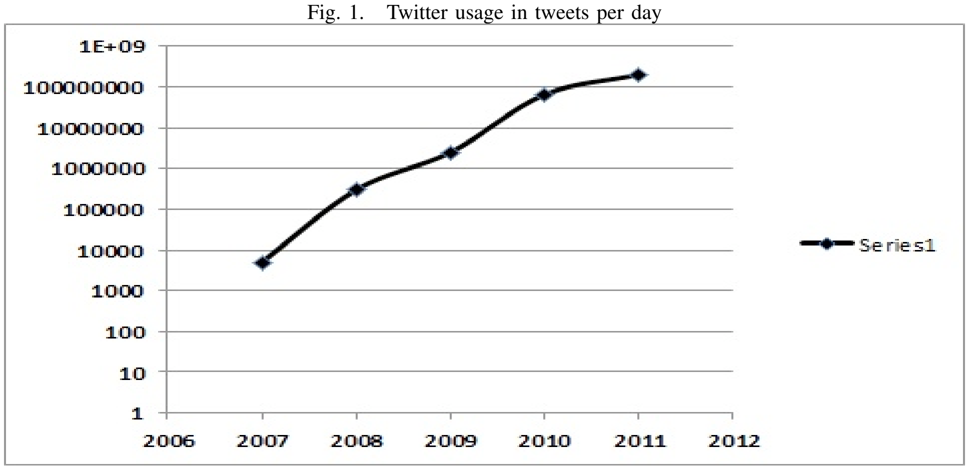
- F-Measure (F): The F-Measure is a measurement involving both R and P as combined in the following equation.

$$F = \frac{2RP}{R+P} \quad (3)$$

TABLE III
Named entity strategies

Strategy	Precision	Recall	F-Measure
CRF on standard texts	91.7	92.0	91.8
CRF on Twitter text	46.3	45.3	45.8
Combined CRF and KNN on Twitter texts	81.6	77.8	80.2

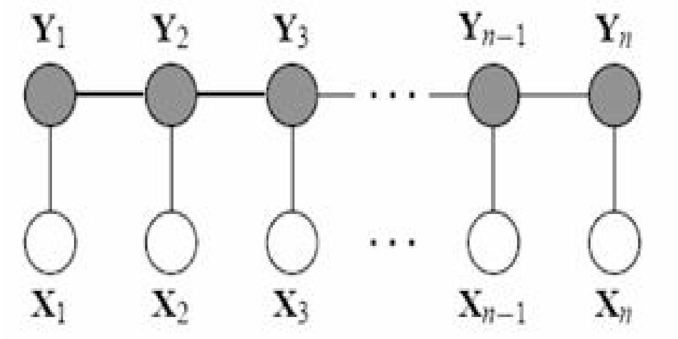
Fig. 1. Twitter usage in tweets per day



B. Conditional Random Fields

Conditional Random Fields are a type of classifier which involves a probabalistic graphical model and is often used in Natural Language Proccessing. This model is used most often for assigning labels to data. It is implemented in place of Hidden Markov Models.
This classifier uses a large amount of training data to draw from as its knowlage base, and then, based on that training data a probabalistic model is formed. Thus, when text is fed into this Condidtional Random Field model, the probabalistic model created by the training data can be used to determine how a spesific word should be tagged or how a sentence should be parsed.

Fig. 2. Y and X of Conditional Random Fieldsl



A Conditional Random Field is similar to a Markov Random Field contained on a random variable $\mathbf{X}$ which represents the observation sequences. We define $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ to be a undirected graphical model such that there exists a node $\mathbf{n}$ in $\mathbf{V}$ which in turn corresponds to another random variable $\mathbf{Y_n}$ in the set of $\mathbf{Y}$ [**?**].

C. Text Normalization

Because Tweets are often written in colloquial English with shortened and altered words, normal NER tools are very unsucessful at determining named entities. This difficulty is also due to the short nature of these Twitter posts, because they have very little context. Liu et al. states that the aim of text normalization is to substitue meaning-consistent standard English for non-standard tokens [11]. There are several ways in which non-standard tokens are used in Tweets according to Kaufmann and Kalita [10].

- The shortening of words
- Spelling errors
- Repetition for emotion

The meaning of shortened words can be found by consulting tables of text acronyms constructed during the research of Choudhury et al. on the structure of texting language [3]. Spelling errors are checked by looking at near letters for reverse order. While repetition of lettedsr is addressed by reducing to 3 repeated letters and checking against other sources [10].
The process of the normalization tool which is used in this project can be seen in figure 2 [10].

Text Normalization is not necessary for NER when the source is simply documents on the web, but when our source is moved to colloquial language sites such as Twitter, conventional machine learning doesn't perform well [6]. Because of this, text normalization is necessary for successful results.

VI. Approach

In this project, a normalizer which translated the text into standard English was used in partnership with a voting CRF classifyer system. These voting classifiers are classifiers which all come up with results on the same selection of text and then, based on the summed results, one classification is reached.

This type of classification is used in the research of Ekbal and Bandyopadhyay [9]. By using this voting system they were able to gain fantastic results of 92.03 in F-measure when

Fig. 4. The process path of a tweet once it has been introduced to the system

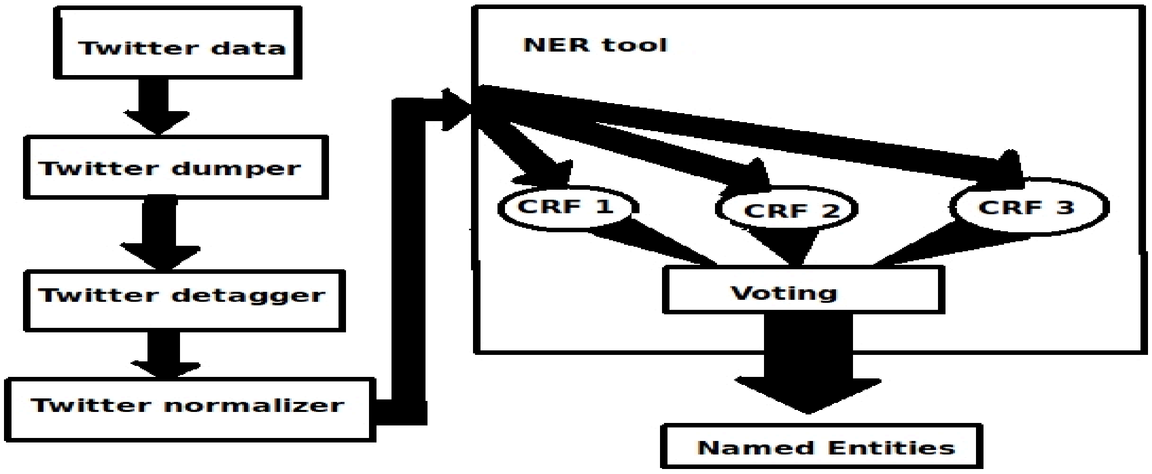
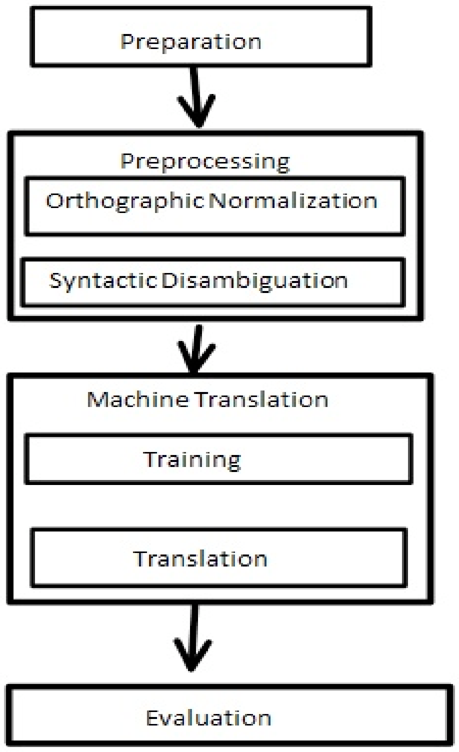


Fig. 3. The process path of the normalization tool



evaluating Named Entity Recognition for Indic laguages. Thus, due to the success of Named Entity Recognition on Twitter posts in our project, we had decided to also experiment with the use of voting classifiers in addition to normalized Twitter posts.

Cross referenced Conditional Random Fields classifiers were used; however, each classifier had been trained on a different set of training data which gave this system more information to draw from. This did increase the effectivness of the given system by .3 percent and will allow for even better results in the future.
During this process a tweet goes through many changes. Table IV shows an example of an actual tweet's changes as it runs through the system.

In previous research by Liu et al CRF and KNN classifiers are combined to create a better NER system, because the f-measure of NER tools drops from 90.8% to 45.8% when used on tweets. In their project they were able to increase the f-measure on tweets to 80.2% when combining the previously mentioned classifiers.

However, The results of this project show that the use of a normalizer with a simple CRF classified NER tool raises the f-measure on tweets to 83.6%. This shows that not only does a normalizer increase the effectiveness of a NER tool, but it can increase it so that it outpreforms a NER tool developed specifically for tweets.

This is a great improvement, and, through the voting tecniques we used with three differently trained Conditional Random Fields classifyers, we were able to increase the F-measure by .3 %. Overall, in this paper we were able to increase the the total F-measure of a NER tool used on twitter posts to 83.9%. This should greatly improve the studies of twitter in the field of Natural Language Proccessing in the future.

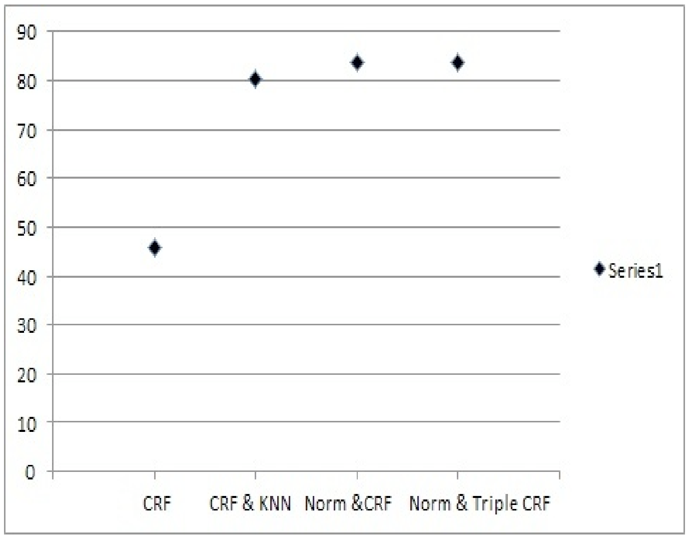
TABLE IV
TWEET NORMALIZATION AND NAMED ENTITY RECOGNITION

Wowwwwwwwwww!!! U guys R done with all the Harry Potter books? lol I hardly finished one!!!!
wow You guys R done with all the Harry Potter books? lol I hardly finished one!!!!
Wow you guys are done with all the Harry Potter books? i hardly finished one!!!!
Wow you guys are done with all the Harry Potter books? I hardly finished one!
Person: Harry Potter

TABLE V
NAMED ENTITY STRATEGIES INCLUDING THIS STATE OF THE ART SYSTEM

Strategy	Precision	Recall	F-Measure
CRF on standard texts	91.7	92.0	91.8
CRF on Twitter text	46.3	45.3	45.8
Combined CRF and KNN on Twitter texts	81.6	77.8	80.2
Triple Voting CRF on Normalized Twitter texts	84.4	83.4	83.9

Fig. 5. The F-measure of Compared NER Systems Having Only a CRF Classifier, Having Dual Classifiers, Having a Normalizer with a CRF Classifier, and Having a Noramlizer with a Triple Voting CRF Classifier Respectively



VII. IMPROVMENTS

A. This project

In this project we set out to increase the effectiveness of a Named Entity Recognition tool on tweets. In order to do this we first created a program which stripped the raw tweets from the tags that are included with them when they are dumped into a file.
This program considered the common characteristics of these tweets and removed the unnessisary peices, the tags. After this, these tweets, which were then just raw tweet text, were placed in the file which input to the Twitter Normalizer.
After this program was completed it was time to set up the Twitter Normalizer, provided by a previous paper written by Kaufmann and Kalita [10]. It took some work to transfer the program to this system, but soon it was in place. A method was then added which would send the normalized text into the input file for the NER tool and run that code.
After this the tweet file would run through the normalizer three times, each with a different CRF classifier which had been traind on different training data. Finally, the classifiers would vote, and any token on which at least two of the classifiers had aggreed as a named entity was considered as such. All those with one or less votes were disregarded.

B. Future projects

There are many future impovements which would be greatly benificial for Named Entity Recognition as it applies to Twitter. Future Improvements include the following:

- Implementing a CRF KNN voting NER which could be combined with the NER tool, (possibly with the addition of a third classifier which would help in the case of ties.)
- Add into the Normalization tool a checker which would take into account common slang and common shortend words which are not included in the program at present.
- Package all of the system into an exicutable .jar file to allow for easier protability and access.
- Create a GUI from which the entire system can be run allowing for easier use by those who do not have an in depth knowlage of programming.

VIII. CONCLUSION

This project has greatly improved the productivity of NER tools on non-standard text such as posts from the micro-blogging site of Twitter. The addition of a normalization tool for tweets allowed for these posts to be translated into standard text which made analysis easier. The voting of differently trained classifiers allowed for higher recall and precition, resulting in an overal higher F-measure on this programs ability to recognise named entities in tweets.

Named Entity Recognition is a large part of information extraction and these advances in relation to NER from Twitter posts will help with many future research problems. Being able to determine if a word that is mentioned in a text is a named person, location, or organization is a huge step forward in Information Extraction and will greatly help analysis of any posts on Twitter. This will be useful in future research such as that involving Natural Language Proccessing and Information Extraction.

Because of this improvement many more research oppertunities in this field will be able to be executed in a much more accurate fashion than in previous such projects. Overall we would judge this project to be a success, and hope that it will bring help to many future projects in this field

References

[1] M. Banko, M Cafarella, S. Soderland, M. Broadhead, O. Etzioni. 2007. *Open Information Extraction from the Web.* In Procs. of IJCAI 2007.

[2] M. Cafarella, D. Downey, S. Soderland, O. Etzioni. 2005. *KnowItNow: Fast, Scalable Information Extraction from the Web.* In Procs. of the Human Language and Technology Conference 2005.

[3] M.Choudhury, R. Saraf, V. Jain, A. Mukherjee, S. Sarkar, A. Basu. 2007. *Investigation and modeling of the structure of texting language.* Int. J. Dpc. Ama;. REcognit., 10(3):157-174, 2007.

[4] D. Downey, M. Broadhead, O. Etzioni. 2007. *Locating Complex Named Entities in Web Text.* In Procs. of IJCAI 2007.

[5] O. Etzioni, M. Cafarella, D. Downey, A. Popescu, T. Shaked, S. Soderland, D. Weld, A. Yates. 2005. *Unsupervised Named-Enitiy Exstraction from the Web: An Expirimental Study.* Artificial Intelligence, 165(1):91134, 2005.

[6] B. Han, T. Baldwin. 2011. *Lexical Normalizations of Short Text Messages: Makn Sens a #twitter.* In Proc. of ACL-HLT 2011.

[7] T. Hasegawa, S. Sekine, R. Grishman. 2004. *Discovering Relations among Named Entities from Large Corpora.* In Proc. of the 42nd Annual Meeting on Association for Computational Linguistics 2004.

[8] H.M. Wallach. 2004. *Conditional Random Fields: An Introduction.* University of Pennsylvania. CIS Technical Report MS-CIS-04-21.

[9] A. Ekbal, S. Bandyopadhyay. 2009. *Voted NER System using Appropriate Unlabeled Data.* In Proc. of the 2009 Named Entities Workshop: Shared Task on Translaiteration (NEWS 2009), ACL-IJCNLP, pp. 210 (2009)

[10] M. Kaufmann, J. Kalita. 2010. *Synatactic Normalization of Twitter Messaqges.* International Conference on Natural Language Proccessing (ICON 2011), Kharagpur, India, December, pp. 149-158.

[11] F. Lui, F. Weng, B. Wang, Y. Lui. 2011. *Insertion, Deletion, or Substitution? Normalizing Text Messages without Pre-categorization nor Supervision.* In the Proc. of The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies

[12] X. Liu, F. Wei, S. Zhang, M. Zhou. 2011 *Recognising Named Entities in Tweets.* Harbin Institute of Technology, Harbin, China.

Automatically Generating Large Freely Available Image Datasets From the Web

Spencer Fonte
University of Colorado Colorado Springs
1420 Austin Bluffs Pkwy, Colorado Springs, CO USA 80918
spencer.fonte@knights.ucf.edu

Abstract

Although there are a few standard datasets in the computer vision community, there are several issues with creating new more challenging datasets. Most of these issues stem from privacy and copyright concerns. This project extends on work done by Mears [1] to develop a new paradigm for collecting and sharing image datasets. In this paradigm, only links to online images are shared using image feeds. Filters can be created and used to produce a new feed that is a subset of an already existing feed, allowing for the easy creation of a specific dataset by using an existing broader dataset feed or the cleaning up of a feed generated by a web crawler. The system consists of three main parts: a dataset feed generator, a feed subscriber, and a contest engine which will allow computer vision contests to be participated in in real time. Architectures for all three parts are provided in this paper and the first two have been implemented. The framework presented in this paper aids in the creation of new computer vision datasets that contain a large number of images, are more representative of the real world, and are less subject to copyright and privacy issues.

1. Introduction

Computer Vision experiments require a large number of images for training and testing algorithms. Creating large datasets that are publicly available can be challenging due to privacy and copyright issues. Most current public datasets are staged photos taken for the purpose of creating a dataset [2, 3, 4, 5, 6]. The images in these datasets do not reflect most of the images people encounter in the digital world today. Efforts have been made to use images found on the web to construct datasets, an example being Labeled Faces in the Wild [7] which is a dataset of facial images of famous people collected from the web. This type of dataset provides images for facial recognition tasks that are more analogous with average facial images taken in the real world. An example image of Tim Allen from Labeled Faces in the Wild is shown in Figure 1. However, there is only an average of around 2.3 images for each person in the dataset. This can be an issue because machine learning algorithms require a lot of data to train. This project seeks to create a paradigm and tool set that allows for the easy creation of large datasets from the web.

Figure 1. Example Image From Labeled Faces in the Wild

Recent work exploring dataset bias [8] highlights that in dataset competitions, algorithms often over-adapt to the peculiarities of a dataset and lose their generality. This work experiments with training and testing on different datasets for object recognition and shows a significant drop in performance. They also discuss the problem with computer vision datasets not being representative of the real world. Our work solves these problems by generating living and breathing datasets from the web. Not only will this prevent over-adapting to datasets, but these images will be representative of the real world.

This project extends work done by Mears [1] and aims to create a system that generates large datasets of images from the web while avoiding privacy and copyright issues. To avoid copyright and privacy issues, the datasets will not be composed of image data but instead of links that point to freely available images on the internet.

A large mass of images without any organization would not be very useful to researchers, so the system must provide a way to clean up the data. This will be done by allowing anyone to be able to create a filter. A filter will take in any existing stream and will output a subset of it as a new stream. Streams will be able to be filtered based on characteristics of the meta data, the image itself, or both the image and meta data.

The system will consist of three main parts; a feed generator, a feed subscriber, and a contest engine. To create these three main parts, a web crawler, downloader, filter creator, feed generator, database, and interface between Matlab, Python, or any other language that a researcher writes code in and the system are necessary. Some of these parts have already been created by Mears [1]. This includes the web crawler, downloader, and some filters. Mears never got a fully functional system working, and we rewrite some of the things he has created. In subsequent sections we will discuss what we have modified and have added to create a functioning system and also what extra features and ideas will be explored once the project in complete. Figure 2 shows a use case diagram for the system.

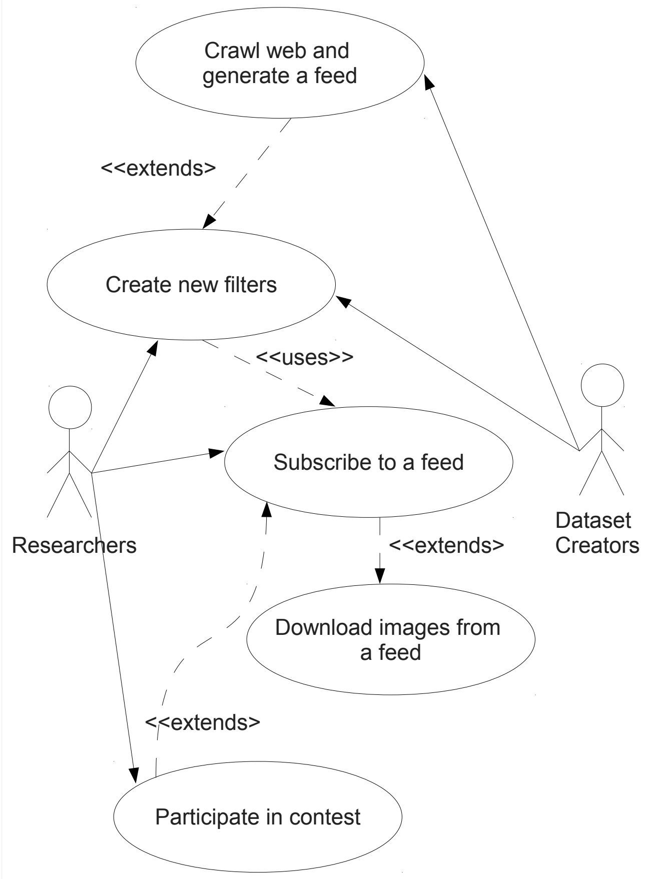


Figure 2. Use case diagram of the system

2. Previous Work and Differences

Although this is a continuation of Mears' work there will be significant differences. The system we will create will be more general and modular and will be more of a framework that will allow researchers to create any type of dataset they would like.

2.1. Web Crawler and Downloader

Mears [1] had modified the Heritrix3 web crawler [9] to gather links and alt text from images on the web along with the website title. He had also created a database to store the information in and then has written a downloader to download the images in order to analyze them. We will keep the project in one language, Python. We wrote a webcrawler from scratch instead of modifying and existing web crawler. This allows our web crawler to be easily customized and used for creating any time of dataset.

2.2. Filters

Mears had implemented a way to detect and remove duplications using a hashing method from [10] [11]. He also used the OpenCV [12] version of the Viola-Jones face detector [13]. He also had found and interesting way to detect logos an such from [14] [15]. Instead of creating filters for specific tasks, we create a filter template that allows for the easy generation of new filters. We also provide examples of filters.

3. Architecture

As stated previously the system is composed of three main parts. Below they will be discussed in detail.

3.1. Dataset Generator

The architecture for a typical dataset generator is shown in Figure 3. It is composed of two parts, a crawler and a filter. It takes as input a list of seed websites for the web crawler, with this list the web crawler will crawl the web and output the URL and meta data for every image it finds. The format of this output is a custom comma separated value feed.

It is unlikely that a dataset creator will want a dataset of all images found on the web. Thus in the typical case a filter will be used to prune the feed generated by the crawler. A filter could prune results based on any criterion on the image or meta data. For example a dataset creator may want to only include images

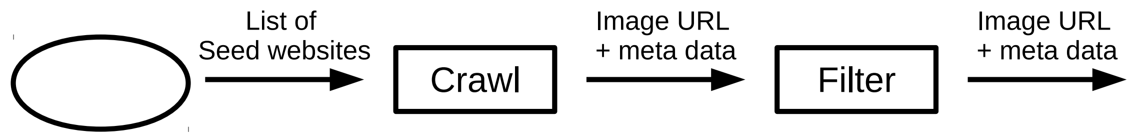


Figure 3. Architecture of Dataset Generator

in which the alt text from the image included the word 'rabbit'.

A more complicated example could be that a dataset creator only wants images of faces. They may choose to accomplish this by creating a filter that downloads images from the web crawler feed then running a face detector on those images with a low confidence threshold. The feed outputted from this filter would then contain images of faces and also a lot of false positives. The dataset creators could then create another filter that would take in the feed outputted by the previously described filter and would then create a new filter that utilizes Amazon Mechanical Turk and only outputs a feed of image URLs and meta data of verified images of faces.

As one can see, the architecture of the system is designed in a way that it is very flexible. In the typical case a web crawler will provide the first feed, but this does not have to be the case. As long as there is a URL to access the image a dataset creator could generate a feed from any source. Also it should be observed that filters can be chained together and be complex. We provide a template to generate simple filters but researchers may choose to create very complex filters on their own, like the Amazon Mechanical Turk example discussed above. Remember a filter just takes in a feed and outputs a feed that is a subset of its input.

3.2. Feed Subscriber

The architecture for the Feed Subscriber is shown in Figure 4. It takes in a URL of a feed that is generated by a Dataset Generator. Feeds are just a standardized file format the first step is to parse the feed file. Once it is parsed any new image information in the feed will then be sent to a module that checks a local database. If the image and its meta data are not in the database, then the image will be downloaded and inserted into the database.

We plan on supporting common database management systems (DBMS). If the user does not want to use a DBMS, then SQLite will be used. Since the system will be very modular if the user wants to use some other non-supported storage system they can just modify the modules that check the database and insert into the database.

Once a user subscribes to a feed they will start downloading the dataset. There could potentially be an issue if multiple research groups want to compare their algorithms and they all subscribed at different times. It is our hope that since disk space is cheap, researchers will subscribe early to feeds that they have a potential interest in. If it is the case that research groups wishing to compare algorithms have subscribed at different times a more recent subset of the feed can be used.

3.3. Contest Engine

Figure 5 shows the architecture of the user side of the Contest Engine. The Contest Engine allows computer vision contests to be preformed in real time. Previous to the contest, the contest host can provide a training feed which would include ground truth. Then during the contest, the contest host would provide a test feed. The research groups participating in the contest would use the Contest Engine which will first subscribe to the test feed then check their local database for the image, download it if necessary, perform their algorithm on the image, and finally output their results as a feed.

The motivation behind the contest engine is to prevent participants in a contest from over adapting to the dataset. Sometimes participants in contests will have algorithms that specifically adapted to the dataset being used. This does not help push the area of the contest to be better as the winning algorithm may be very good on the dataset being used in the contest but perform poorly in general. The contest engine will allow contest to be carried out and since the dataset is live from the web and constantly changing participants must solve the general problem at hand.

We describe the architecture for the contest engine above but we did not implement it. This would be great future work.

4. Implementation

This project aims to be cross platform and to be easy to modify. All of the components are written in Python. The project uses some open source libraries

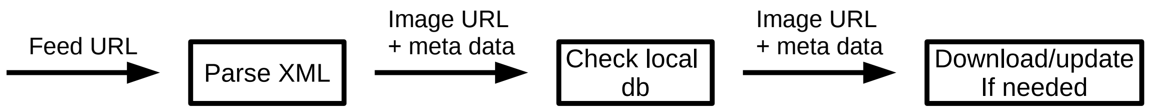


Figure 4. Architecture of Feed Subscriber

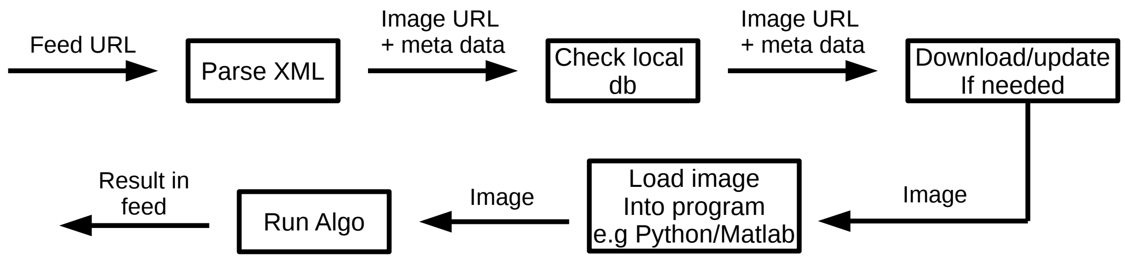


Figure 5. Architecture of Contest Engine

and these will be bundled with the project.

4.1. Dataset Generator

4.1.1 Web Crawler

The dataset generator is organized into a Python script for the web crawler and another script for the filter. The web crawler script utilizes a feed class. The web crawler begins by putting one or more "seed" URLs into a pool. Then a URL is selected from the pool and its content is downloaded and parsed for links. The found links are then added to the URL pool. Beautiful Soup is used to find all the images that the web page refers to. Relative links and image references are both made absolute. Every image link found along with the URL of the web page it was found on, its alt text, and the current time are formed into a tuple which is then added to a list in the feed class. After visiting a web page the crawler calls a publish method on the feed class. This outputs the contents of the feed to a file as comma separated values.

The web crawler does not strive to be extremely fast. It is not multi-threaded, it even pauses for a few seconds after it visits each site. We do not strive for speed in the web crawler because it generates a feed which subscribers must read, parse, download images, and process images from. If the web crawler performs at a significantly quicker rate than the subscribers the amount of "new" unchecked items in the feed will increase. A result of this is that the time between the web crawler placing an image URL in the feed and the time that URL is checked by the subscriber can be a significantly long time, this can result in links no longer be active or accurate which is undesirable.

4.1.2 Feed Format

The feed is contained in a custom comma separated value file. The first line of the feed file contains the Unix time for the last update to the feed. This allows the file to be quickly checked for updates as only one line needs to be read. The second line contains the name of the feed. The third line contains the URL where the feed is located. The fourth line contains a short description of the feed. Then there is a blank line and the sixth line contains the names of the fields for each entry separated by commas, the first entry is always the date of the entry. Each line after that contains a line for each entry in the feeds. There is an entry for every image reference. Bellow is a sample of a feed.

```
1311712160
Bikes vs. Mobiles
http://www.anexample.com
Bikes and mobile Phone pictures from Craigslist

date pub,img url,site linked from,alt text
```

Then the feed would contain all the entries on separate lines. These are too long with all the URLs to show an example for in a sensible way.

4.1.3 Filters

Filters must subscribe to an existing feed in order to filter its results. This may be a feed being used for another dataset or the feed being generated by the web crawler. To subscribe to a feed the filter framework periodically opens the feed file of the feed it is subscribing to. It checks the first line of the feed file which contains the time the file was last updated. If this time is greater than the time of the last check the filter framework subscriber will parse and process the feed file line by line until it reaches the entries it has already processed.

The filters themselves are implemented in a functional manner. Each filter is a function. These functions can be chained together by having one filter call another one. Every time a new entry is parsed from the feed, the first function in the filter chain is called and is passed the data for the entry. To create filters easily Open CV or the Python Imaging Library can be imported and their functionality can be used within the filter functions.

4.2. Subscriber

The subscriber periodically checks a feed for new image URLs and if there are new image URLs it will download them and store their meta data in a database. The subscriber is implemented in Python like the rest of the project. In the same way the filter subscribes to feeds the subscriber only needs to check the first line of the feed file in order to determine if any changes have been made. If there have been updates the subscriber only reads the file until it has read all the new entries it has not yet read before. The database being used currently in a Sqlite3 database. The Python library is used to create and interact with the database file. The images are downloaded into a standard directory and the paths to the images are stored in the database with the other meta data related to the image and the website it was found on. This allows quick querying on the image meta data and then a simply using the path to retrieve the image data if it is desired.

5. Observations

To test the current system, which consists of a web crawler that generates feeds and a filter which subscribes to a feed and filters the results, we inserted code into the filter code to display the output of the filter. We first used a filter that simply outputs everything from the feed it reads. When running this filter on the output from the web crawler it is clear how messy the images on the web are. A significant proportion of the images found are styling elements of websites, this includes logos, images for tool bars, and one-by-one pixel images used to make shapes of one color. Some examples are show in Figure 6.

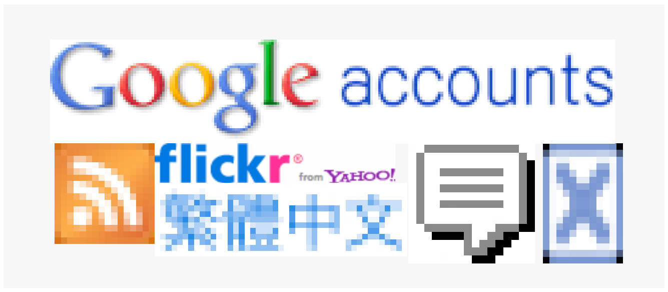


Figure 6. Examples of logos found when crawling the web

We begin experimenting with other simple filters. We observed that a filter which only keeps images in which their size in each dimension is over a certain threshold is very effective at eliminating logos and other images for styling websites. However we want to reiterate that the system does and will not filter out these images by default. Although we will include this filter to be used, some people may want to create datasets that include these images. Our system is made to be flexible and easily modifiable.

On a small set of 434 websites 20418 images are referred to. This averages to 47 images per site. However most of these images are just logos and other features used to style web pages, and also avatars. The histogram in Figure 7 shows the minimum dimension size for each image found.

6. Testing and Applications

In order to test the system and demonstrate its potential uses we have crawled the web to create feeds, filtered these feeds, and subscribed to these feeds. We started out with choosing a seed and generating a feed based on the web crawler unrestrictedly crawling, and then using a basic filter to filter our small images as discussed in the previous section.

6.1. Craiglist: Bikes vs. Mobile Phones Dataset

To demonstrate how customizable the system is and to show a potential application we limited the web crawler to Craigslist classifieds web pages for bikes and mobile phones. Then we subscribed to this feed and downloaded the images and meta data. The result of this is a simple example dataset that could be used for object detection of mobile phones and bicycles. We let the subscriber download images for a few hours and the number of images acquired is shown in the following table.

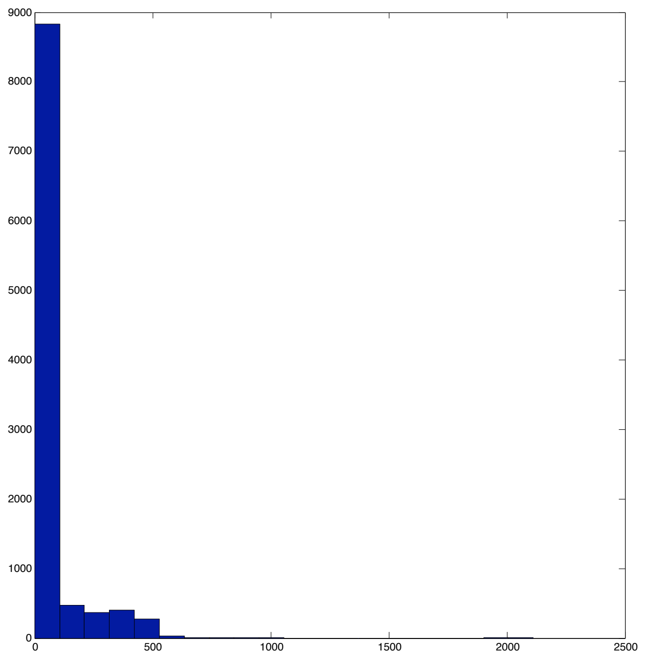


Figure 7. Histogram showing the minimum dimension size for each image

	Bikes	Mobile Phones
Unique images with files	15717	11137
All unique images	16040	11356
Total entries in database	31578	22493

In the table above the first row shows the number of actual unique image files downloaded that we also gathered meta data for. The second row shows the number of unique images that were entered into the feed but that the subscriber failed to download, we end up with the meta data but no image file for these images. The last row in the table is the number of entries of meta data that were entered into the database from the subscriber.

The reason for such a large discrepancy is that the feed generator has an entry for each unique pair of images URL and the URL of the web page it was referred to from. It is not uncommon for the same image to be used on multiple web pages. This is especially true of logos that appear on every web page within a web site.

Figure 8 shows some images that were collected as part of this example. It is clear that they are more representative of images of mobile phones and bicycles that people generally encounter.:

6.2. Future Work

In the future this project can be extended in several ways. Although a basic architecture was described in this paper the contest engine has not yet been implemented. This would be a good first extension to this work.

In this paper we show that the described and implemented framework can be used to create datasets from Craigslist. In the future more example datasets should be generated from the framework, this would help test the framework and also make researchers more likely to use it for dataset generation and acquisition in the future. This is important since multiple research groups must be convinced to subscribe to a dataset feed for the feed to be useful. Ebay would be a good contender for testing out the framework, it has very well defined categories, is always being updates, and has a large number of images. We believe it would be a good resource for creating a object recognition dataset.

After the contest engine it should be used to hold a basic example contest. This will provide more information on the properties of datasets created using the system and will allow the differences between live datasets and old standard datasets to be explored.

Another extension of this project would be to add a graphical user interface. As of now the interface is command line based. This extension could make using the system more intuitive. A web based interface would also be worth exploring as it would be cross platform and allow for people to monitor and control the system off site with ease.

As discussed previously the system is not multi-threaded. Even though we do not view this as a large disadvantage, it could be explored. If the web crawler and feed generator were made mutli-threaded it would be necessary to make the filter framework and the subscriber to also be multi-threaded to avoid the problem of the subscriber being unable to keep up with the rate at which a feed was being added to.

We believe our feed format can handle hundreds of thousands of entries in a feed. To make the reading of feeds more efficient one could split the feed up into multiple files and create a directory file. Then when the feed is read first the directory file would be read and then the appropriate feed file would be read. This would prevent having files that are enormous.

References

[1] B. Mears, “Generating a Large, Freely-Available Dataset for Face-Related Algorithms.” [Online]. Available: http://www.cs.uccs.edu/~kalita/work/reu/REUFinalPapers2010/Mears.pdf

[2] L. Fei-Fei, R. Fergus, and P. Perona, “Learning generative visual models from few training examples: An incremental bayesian approach tested

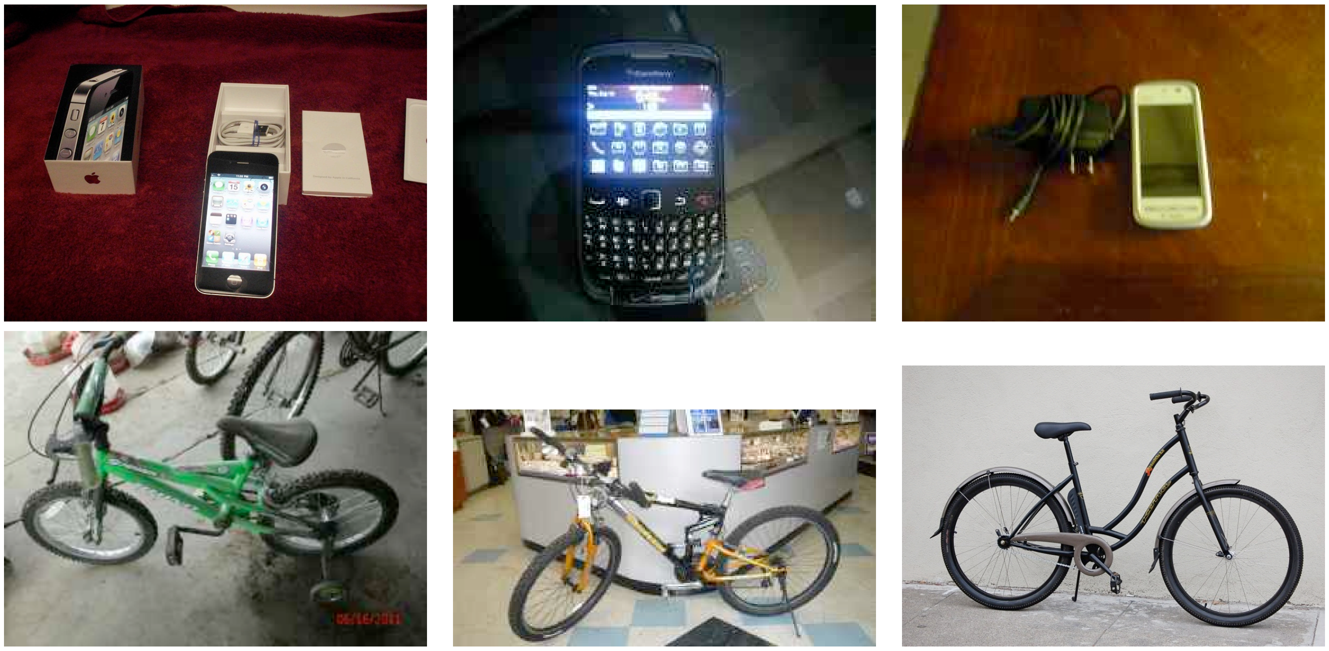


Figure 8. Example images from our bikes vs. mobile phones dataset

on 101 object categories," *Computer Vision and Image Understanding*, vol. 106, no. 1, pp. 59–70, 2007.

[3] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007.

[4] R. Gross, I. Matthews, J. Cohn, T. Kanade, and S. Baker, "The CMU multi-pose, illumination, and expression (Multi-PIE) face database," Technical report, Robotics Institute, Carnegie Mellon University, 2007. TR-07-08, Tech. Rep.

[5] T. Sim, S. Baker, and M. Bsat, "The CMU pose, illumination, and expression database," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1615–1618, 2003.

[6] P. Phillips, H. Moon, P. Rauss, and S. Rizvi, "The FERET evaluation methodology for face-recognition algorithms," in *1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1997. Proceedings.*, 1997, pp. 137–143.

[7] G. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," *University of Massachusetts, Amherst, Technical Report*, vol. 57, no. 2, pp. 07–49, 2007.

[8] A. Torralba and A. Efros, "Unbiased Look at Dataset Bias," in *Proc. IEEE CVPR 2011.*

[9] K. Sigurdwwon, M. Stack, and I. Ranitovic. Heritrix user manual. Internet Archive. [Online]. Available: http://crawler.archive.org/articles/user_manual/index.html

[10] S. Xiang, H. Kim, and J. Huang, "Histogram-based image hashing scheme robust against geometric deformations," in *Proceedings of the 9th workshop on Multimedia & security.* ACM, 2007, p. 128.

[11] M. Mıhçak and R. Venkatesan, "New iterative geometric methods for robust perceptual image hashing," *Security and Privacy in Digital Rights Management*, pp. 13–21.

[12] G. Bradski, "The OpenCV Library–An open-source library for processing image data," *Dr. Dobbs Journal*, pp. 120–125, 2000.

[13] P. Viola and M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple," in *Proc. IEEE CVPR 2001.* Citeseer.

[14] A. Hartmann, "Classifying images on the web automatically," *Journal of Electronic Imaging*, vol. 11, no. 4, pp. 1–0, 2002.

[15] V. Athitsos, M. Swain, and C. Frankel, "Distinguishing photographs and graphics on the world wide web," in *IEEE Workshop on Content-Based Access of Image and Video Libraries, 1997. Proceedings*, 1997, pp. 10–17.

Can Summarization Improve Question Quality?

Bethany Griswold, University of Colorado Colorado Springs

***Abstract*— This project focuses on the generation of questions based on Wikipedia articles, as well as the ranking of the questions based on the content therein. A Wikipedia article was first processed through a question generator, which ranked questions generated based on grammatical correctness and comprehensibility of the question. Named entities throughout the Wikipedia article were counted, consolidated, and ranked based on the number of times the entities occurred in the article. Then each question was ranked based on the number and significance of named entities within the sentence. The score given to a question for content relevance was averaged with the score for grammar and comprehensibility, and then a final ranking was done. Another system was also done where the Page Rank algorithm was used on the questions with presence of named entities used as the feature vector, and then the score given to a sentence for page rank was averaged with the grammar score and ranking was done based on the resulting scores. The resulting set of ranked questions were compared against each other and against the results of the Question Generator itself.**

I. Introduction

Wikipedia, a website devoted to the collection and maintenance of a vast amount of knowledge, has in recent years increased in its reliability as an on-line reference. Many other web-based document sites have also emerged as potential educational resources for teachers. However, unlike traditional textbooks, most of these do not have practice and content questions that can be used to test the knowledge and understanding of students who have read the document [1]. Automatic generation of such questions is an interesting and useful problem. Questions will be formulated using a pre-existing system by Heilman [1]. This is how the algorithm is described:

"Each of the sentences from the source text is expanded into a set of derived declarative sentences (which also includes the original sentence) by altering lexical items, syntactic structure, and semantics. [...] a set of transformations derive a simpler form of the source sentence by removing phrase types such as leading conjunctions, sentence-level modifying phrases, and appositives. [...] [The] implementation also extracts a set of declarative sentences from any finite clauses, relative clauses, appositives, and participial phrases that appear in the source sentence. [...] In the second step, the declarative sentences derived in step 1 are transformed into sets of questions by a sequence of well-defined syntactic and lexical transformations (subject-auxiliary inversion, WH -movement, etc.). It identifies the answer phrases which may be targets for WH-movement and converts them into question phrases. [...] The transformation from answer to question is achieved by applying a series of general-purpose rules. [...] Eight Tregex expressions mark phrases that cannot be answer phrases due to WH-movement constraints. [...] We iteratively remove each possible answer phrase and generate possible question phrases from it. [...] Each question is scored according to features of the source sentence, the input sentence, the question, and the transformations used in its generation."

This process is shown in Figure 1.

Ranking of content will be done by calculating significance of named entities and their density within a sentence and averaging that score with the score given to each sentence by Heilman's algorithm. We will also use a Page Rank algorithm with feature vectors of Named Entities within the article as an additional scoring method for relevance. These entities will be taken from data extracted using a pre-existing system which has succeeded in extracting temporal and geospatial information [2].

II. Motivation

Recent methods of automatically generating questions do so by means of direct sentence manipulation to produce natural language questions in English [3]. Sentences in a document are manipulated to replace known entities with question words (who, what, when). The sentence is then rearranged by the system so that it is more likely to be grammatically correct, and the most likely to be grammatically correct are displayed to a user to choose from. By contrast, our method attempts first to obtain the sentences most likely to be dense with content most central to the article, and focuses on ranking by content importance and relevance. The question generator will still rank for grammatical correctness, but then the questions will be re-ranked to account for relevance.

III. Prior Work

There has been a variety of work previously on question generation from textual content in the past. One method is based on manipulating the structure of each sentence in an article to generate one or more questions, then ranking the questions based on grammatical correctness and importance. This is the "over-generate and rank" method, on which this research will be partially based [3]. Another method is question generation for the particular domain of vocabulary assessment [4]. There are also some tools and research on which this project relies. The first is the Geografikos system, which extracts temporal and geospatial data and then tags sentences to indicate when and where they happened [2] [5]. We are also following prior work of using summarization for feature selection as a first step to a further goal. Where our goal is question generation sorted by importance, it has previously been used for text categorization [6]. We also used the Page Rank algorithm as an additional content importance ranking system [7], [8].

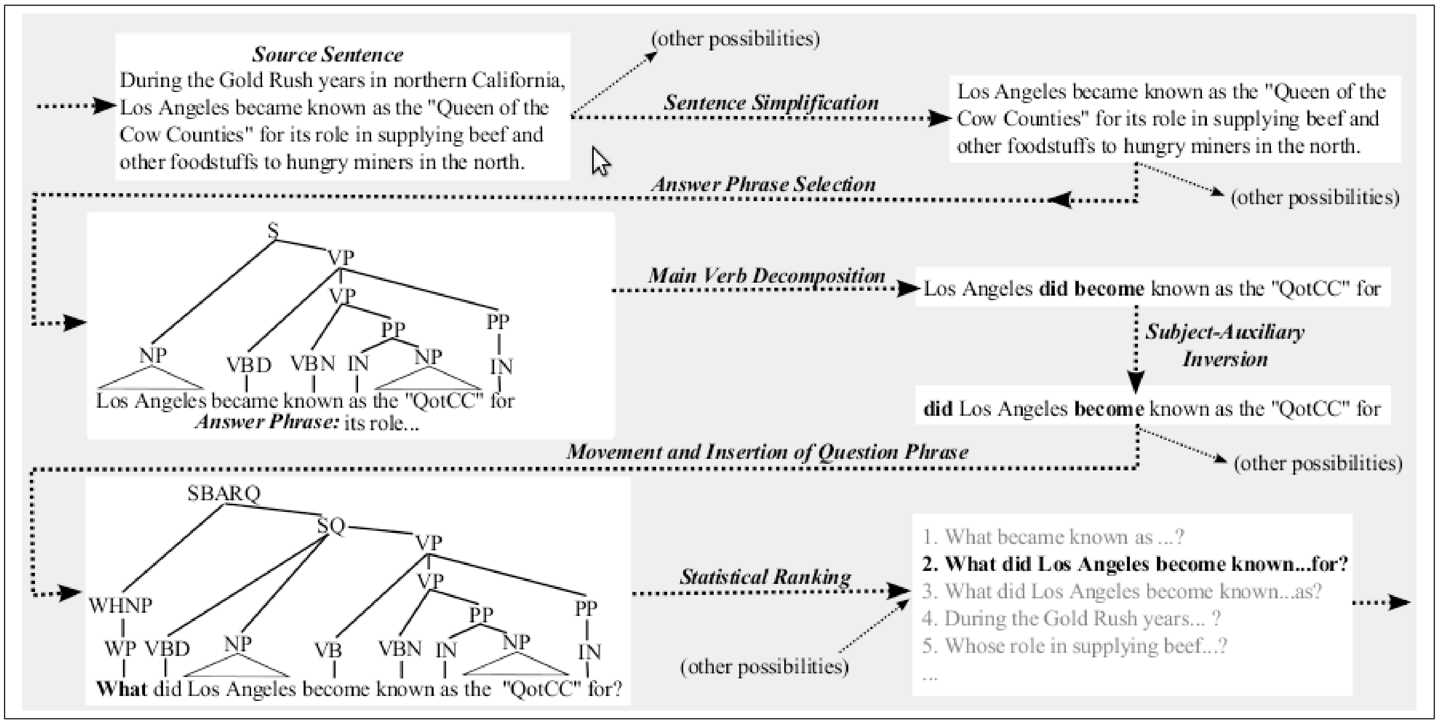

Source Sentence
During the Gold Rush years in northern California, Los Angeles became known as the "Queen of the Cow Counties" for its role in supplying beef and other foodstuffs to hungry miners in the north.
(other possibilities)
Sentence Simplification
Los Angeles became known as the "Queen of the Cow Counties" for its role in supplying beef and other foodstuffs to hungry miners in the north.
Answer Phrase Selection
(other possibilities)
Los Angeles became known as the "QotCC" for
Answer Phrase: its role...
Main Verb Decomposition
Los Angeles did become known as the "QotCC" for
Subject-Auxiliary Inversion
did Los Angeles become known as the "QotCC" for
Movement and Insertion of Question Phrase
(other possibilities)
What did Los Angeles become known as the "QotCC" for?
Statistical Ranking
(other possibilities)
1. What became known as ...?
2. What did Los Angeles become known...for?
3. What did Los Angeles become known...as?
4. During the Gold Rush years... ?
5. Whose role in supplying beef...?
...

Fig. 1. Heilman's Overgenerate and Rank Process

IV. Project Goals

Our ultimate goal is to generate concise questions about historical articles which are ranked based on the strength of their relation to the core ideas of an article.

At the start, we strip a Wikipedia article down to plain text. As an example, use an article on the Battle of Fredericksburg. Here is an excerpt from that article:

"*The battle was the result of an effort by the Union Army to regain the initiative in its struggle against Lee's smaller but more aggressive army. Burnside was appointed commander of the Army of the Potomac in November, replacing Maj. Gen. George B. McClellan. Although McClellan had stopped Lee at the Battle of Antietam in September, President of the United States—President Abraham Lincoln believed he lacked decisiveness, did not pursue and destroy Lee's army in Maryland, and wasted excessive time reorganizing and re-equipping his army following major battles.*"

The question generation tool created by Heilman generates questions and ranks them based on the structure of the question. Following are the top four ranked questions from Heilman's question generator, given the article on the Battle of Fredericksburg:

- When was Burnside appointed commander of the Army of the Potomac?
- Who was Burnside appointed in November?
- What was Burnside appointed commander of of the Potomac in November?
- What was Burnside appointed commander of the Army in November?

All of these questions are generated from the same sentence, but only the first one is really a grammatically good question. None of them are questions that address a very central concept of what, when, and where the battle was or why it happened. From a summarizer, we would hope to preserve as important a sentence such as the first in the given paragraph:

"*The battle was the result of an effort by the Union Army to regain the initiative in its struggle against Lee's smaller but more aggressive army.*"

Such a question could be, "*What was the battle a result of?*"

We have multiple potential methods to try and accomplish this. We used two methods of ranking for relevance based on named entities. For each of these, the article is process through Heilman's question generator first, in order to get the grammar score. The first analyzed the text for named entities, ranked them, and used their presence within a given sentence to determine the importance of that sentence to the text overall. This is done by counting up the number of times a named entity is mentioned within an article and giving that named entity a score of that number. Then each sentence is given a score determined by all the scores of the named entities within that sentence divided by the length of the sentence overall. This score is averaged with the grammar score determined by Heilman's algorithm and the resulting questions are ranked, with the highest scored question first and the lowest scored question last. The second method uses the Page Rank algorithm, which determines relevant and similar sentences based on the similarity of what named entities each has. These processes are shown in Figure 2. The diagrams show that following each process, the score for relevance is averaged with Heilman's score. We also used a fairly simple method for comparison, which consists of processing the entire text through a summarizer and then processing the summary through a question generator, as in Figure 3.

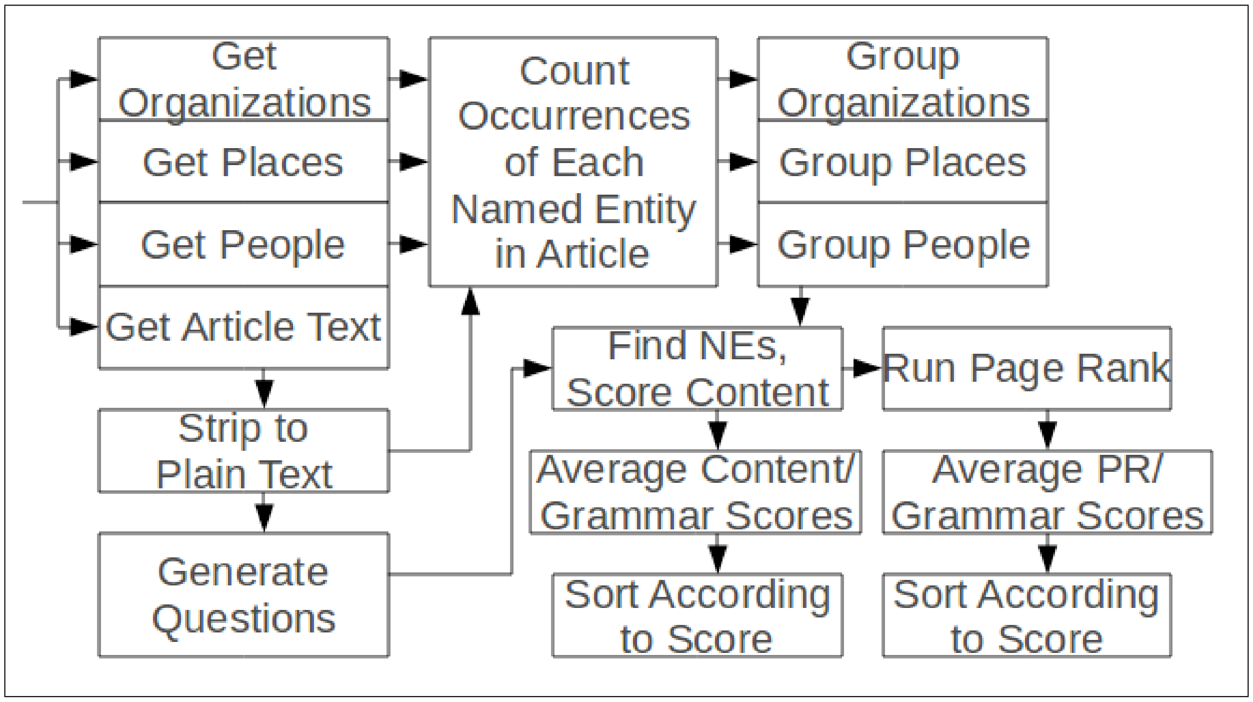


Fig. 2. Ranking Process for NE Based Content Ranking and Page Rank Based Ranking. Any summarization algorithm can be run on a body of text before sending through this process in attempt to acheive better results.

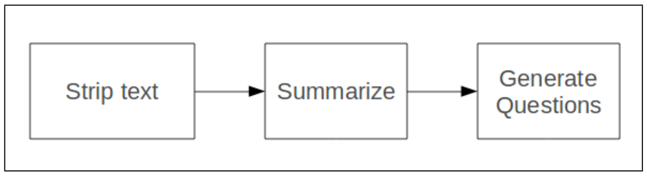


Fig. 3. Summary Based Ranking Process

V. Evaluation

The top 50, 25, and 10 questions, fair numbers of questions for a quiz or test, produced by our six current methods of ranking were rated by one person each on four different features. The first feature is difficulty, which we are testing to see if there is a trend between method of question generation and difficulty of questions. We hope we can pick out questions out of a corpus for a particular grade level. The second feature is relevance, which tests whether or not understanding of the particular question is important or vital to understanding of the article as a whole. We hypothesize that most questions rated highly for content by our algorithm should on average be rated highly for the relevance feature by our evaluators. The third feature is precision, which is the degree to which the evaluator understands what the question is asking and feels that the question is comprehensible and grammatically. We feel that high ratings for precision will closely correspond to a high ranking in Heilman's algorithm.

All evaluators were given the full text of the article on the Battle of Fredericksburg, along with the top 50 questions produced by one of the methods of ranking. Each ranking method was given to one person only. We decided not to allow one person to evaluate more than one set of questions, as we were concerned that a previous set of questions they had read would influence their opinion of the current set of questions. We also did not tell them the ranking method we used, in case assumption of one set having a better ranking method would influence their evaluation of the questions.

There is a bit of a conflict in doing these evaluations, as strong readers may be best suited to assess vagueness, relevance, and completeness, but their high reading level may cause their judgments of difficulty to be very low. We do not have the resources, however, to poll a wide variety of reading levels, so while this point must still be evaluated, the bias of the evaluators must be taken into account. We may also be able to get better results if we are able to have each question set evaluated by a greater number of people, so that the ratings are less influenced by individual skill level.

The methods of question ranking which we decided to give out for evaluation are as follows:

- Question generation from the full text and no further ranking.
- Question generation from the first and last paragraph summary of the full text and no further ranking.
- Question generation from the full text, followed by Named Entity based content ranking.
- Question generation from the first and last paragraph summary of the full text, followed by Named Entity based ranking.
- Question generation from the full text, followed by Page Rank based content ranking.
- Question generation from the first and last paragraph summary of the full text, followed by Page Rank based content ranking.

Different Summarizers could be used in place of the very simple reduction to the first and last paragraph of the text, if a more intelligent summarization system was desired.

VI. Results

One observation given to us by some of the evaluators was that questions generated that required simply yes or no answers were almost too easy to be good questions. Since this is not accounted for in the rankings, a binary of whether something is an acceptable or good question, we may want to do some evaluations in the future that do address this, or otherwise use the feature of Heilman's algorithm that limits output to questions that cannot be answered by true or false before doing any more ranking and evaluation.

The average scores on each quality for the top 50, 25, and 10 generated questions are shown in the graph in Figure 4. Set 1 is the question generation from the full text followed by Named Entity ranking. Set 2 is the set produced by summarizing the text by using the first and last paragraph before generating questions from the summary and then processing the questions through NE based raking. Set 3 was obtained by processing the full text through the question generator, but no other ranking was done. Set 4 is similar, but the text undergoes first and last paragraph summarization first. Set 5 is the full text processed through the question generator and then ranked using the Page Rank algorithm. Set 6 also uses the Page Rank algorithm, but starts out with a first and last paragraph summary.

There were some interesting features of the rankings in comparison to each other. We anticipated that when summarizing and ranking based on named entities, overall relevance of the top 50 ranked questions to the central points of the article would be improved from algorithms that focused primarily on comprehensible grammar. However, as you can see in Figure 4, this is not always the case. The questions in set 3 were ranked most highly for relevance to the article. However, set 3 was the set produced by Heilman's algorithm only, and nothing was actively done to promote more significant questions in the ranks. In fact, it seems that on average, no feature of the top questions was significantly improved from the base question generation by counting named entities and altering the ranking of questions based on density and significance of named entities or by summarizing the text before question generation. However, compared to a simple summarization before question generation, summarization followed by question generation and NE based content ranking shows a marked improvement in precision, relevance, and completeness, as you can see from set 2 and set 4 in Figure 4.

Seeing that the overall average rating of features did not improve with our added content ranking systems, we also did some analysis of the evaluations on a case-by-case basis, with regards to the trends of how well certain features were ranked as an evaluator goes down the list of questions. We expected that the questions near the top of the list would be higher rated in all features except perhaps difficulty than those near the bottom of the list. However, as you can see in Figures 5 and 6, this is not the case in any generalizable way. Set 3 has only a slight tendency toward both better precision ratings and better

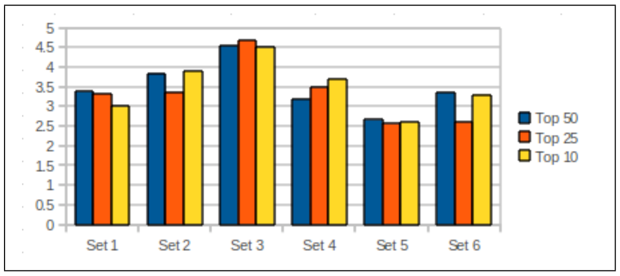


Fig. 5. Relevance scores for all sets of questions generated for the top 50, 25, and 10 questions. Only the questions in set 4 shows improvement of question relevance ratings with increased selectivity of questions.

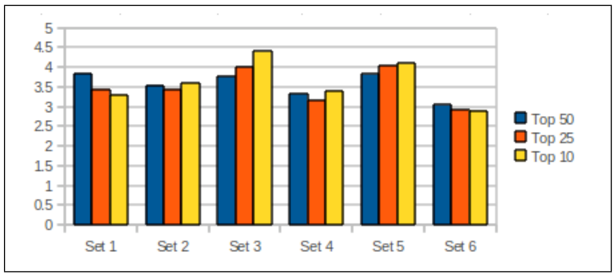


Fig. 6. Precision scores for all sets of questions generated for the top 50, 25, and 10 questions. Sets 3 and 5 are the only sets of questions that show improvement of question precision ratings with increased selectivity of questions.

relevance ratings in earlier questions, and this is, once again, the set that we did not perform any additional rankings on. The failure of all the additionally content ranked sets we believed was likely due to content scores being averaged with grammar scores, so that many of the questions may be very strong in one sense, which compensates enough statistically for weakness in another feature to allow it to be placed high in the rankings. To test this hypothesis, we superimposed the rating for precision, which should be most closely tied to ratings for grammar and comprehensibility, on top of the ratings for relevance, which should be closely tied to algorithmic ranking of content. This is shown in Figure 7. If it was the case that our algorithmic content ranking matched up with human evaluation of content importance or centrality, Heliman's algorithmic grammar and comprehensibility ranking matched up with human evaluation of precision, and the averaging of these features was causing promotion of bad questions, we would see a nearly polar opposite ranking on the other feature for questions rated very high in a single feature late in the rankings or very low on a single feature earlier in the rankings. We once again do not see this in a definite enough trend to make a conclusive statement about whether this is the case. However, it is once again possible that given a high degree of variability in ratings due to variability in skill levels, more evaluators could prove useful in giving a more accurate assessment of each corpus of questions.

Another feature we wanted to consider looking into was whether we could use certain summarization methods to automatically extract questions with a certain level of difficulty from a large corpus of questions. On this task, we got fairly reasonable results. The questions with the highest difficulty came from set 1, which generated questions from the full text

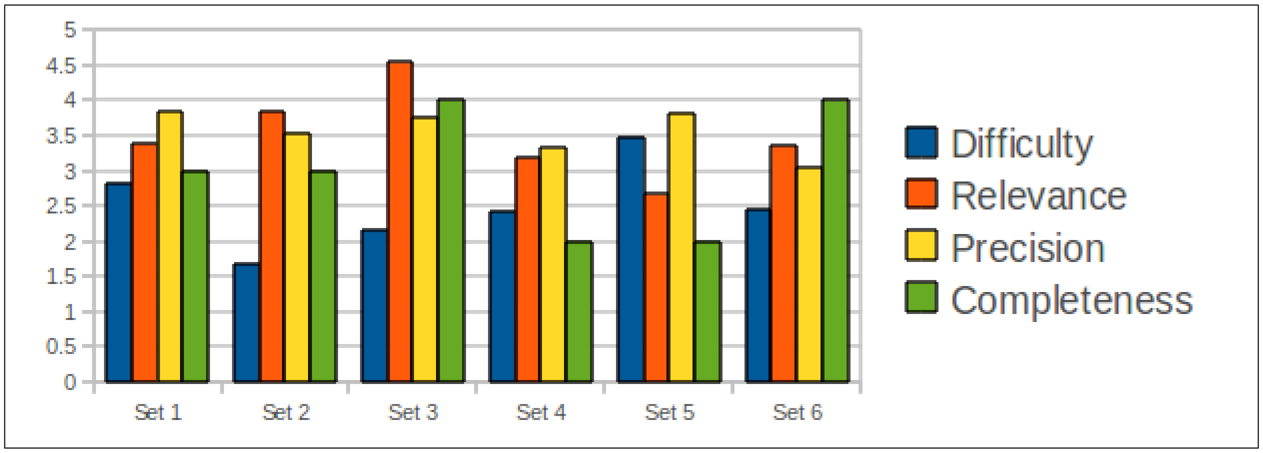


Fig. 4. Average ratings for features of top 50 questions generated by rating methods

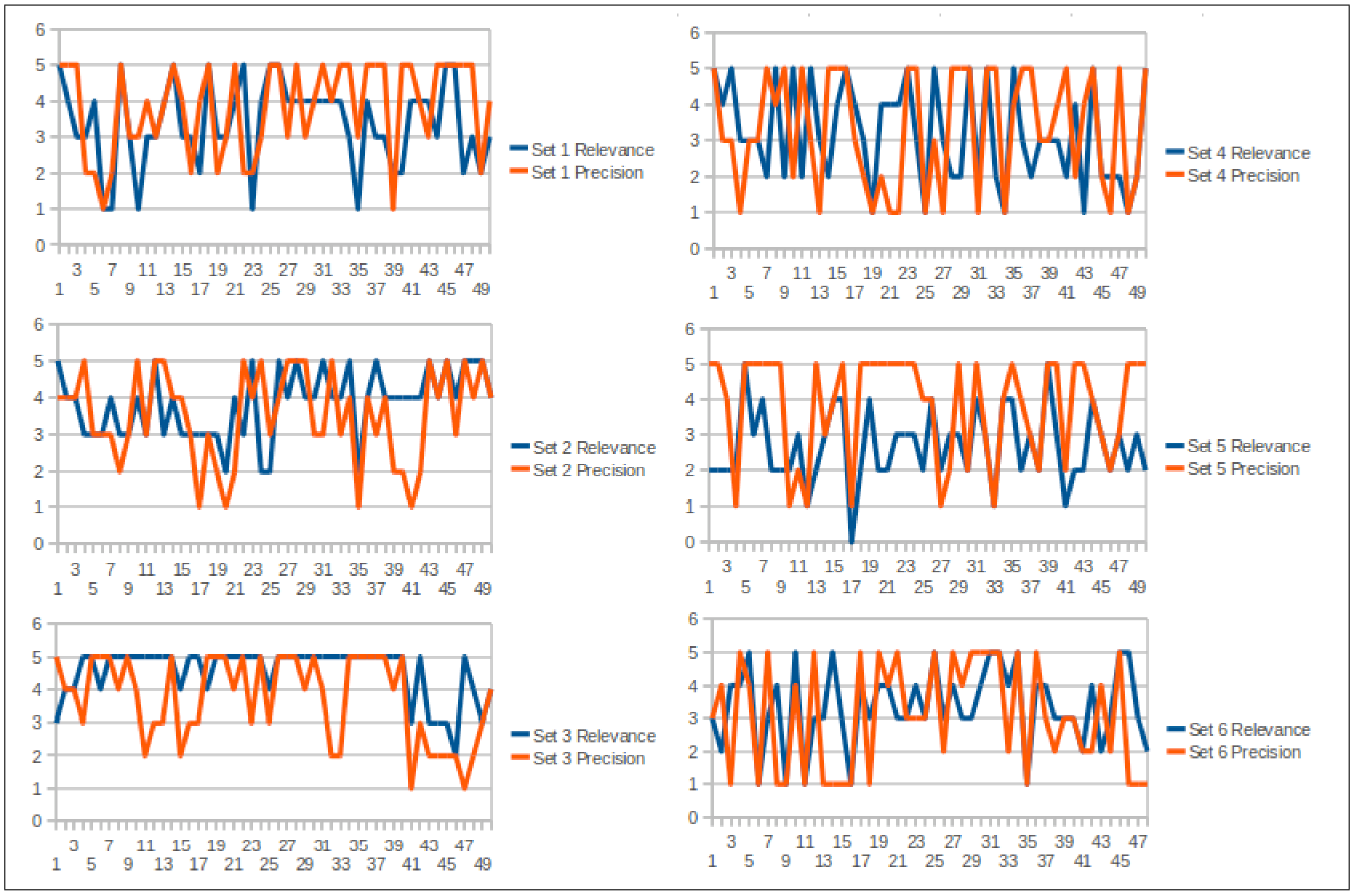


Fig. 7. Precision scores superimposed on top of relevance scores. We supposed that we could be getting low ratings on questions ranked in the top 50 due to a polarity between the grammatical correctness and article relevance giving a question a decent score and rating. While some cases show that low precision scores are counterbalanced by high relevance scores and vice versa, this is by no means an overarching theme in our current data and we cannot assume this is the cause of a lack of downward trend in acceptability of questions in lower rankings of questions.

and then re-ranked based on NE counts. The questions with the lowest difficulty came from set 2, which underwent the same process as set 1, but first was summarized to content only in the first and last paragraph. Set 1 likely has a high degree of difficulty in part because questions were generated from the entire body of the text, where there are many minute details that could be difficult to retain throughout the course of reading the document. Yet this may not be entirely true, since the questions from Set 3 were deemed slightly less difficult than those from set 4, despite that Set 4 was summarized while Set 3 was not and the process for generating and ranking the questions was otherwise identical.

VII. Conclusion

Neither simple summarization before nor content ranking based on Named Entity counts after question generation shows any obvious improvement in question quality, article relevance, or completeness of a question set in evaluating understanding of an article. Only a very slight improvement in precision was seen from the sets with named entity counting and no summarization and page rank ranking and no summarization. This could be due to the averaging of scores for content and grammar quality resulting in promotion of questions that are very good in one feature and very poor in another, but analysis of the ranking data does not give conclusive support for this hypothesis. It is also possible that more sophisticated summarizers could show an improvement, or that using a voting method of rating rather than an average content and grammar score method could eliminate promotion of questions with poor scores in one feature or the other. Further evaluation may also need to be done on our current set of data in order to determine how heavily individual ability played a part in initial rankings of questions, and we may also want to have a binary evaluation of questions being acceptable or unacceptable. Until further evaluation proves otherwise, we must conclude that Named Entity significance and density evaluation on questions does not help to improve the content quality and overall acceptability of generated questions.

VIII. Future Work

To optimize on the different strengths and weaknesses of each method, we hope to combine various methods in a voting system in order to rank key questions highly and attempting to maintain good, comprehensible grammatical structure. It is possible that using a voting system between the top ranked questions using Heilman's algorithm and those ranked by any number of ranking systems based on relevance will be more effective than averaging scores in bringing questions to the top which are both fairly relevant and fairly grammatical. It could avoid retrieving questions that are highly grammatical but completely irrelevant or highly relevant but incomprehensible. Currently, we have a voting algorithm implemented, but several result sets must still be generated from it and evaluations must be gathered from a group of people.

It would also be beneficial to attempt to improve results by using a wider variety of summarizers before question generation. One such summarizer is the MEAD summarizer [9]. Summarization algorithms are meant to compress a document down to the essential and central information, and so more sophisticated summarization than first and last paragraph summarization could potentially result in a drastic improvement in result relevance. However, it's also possible that summarization could cause an increase in overall ambiguity of the questions. This is partially due to there being simply fewer sentences which Heilman's algorithm can rank for grammar and comprehensibility, and partially because summarization inherently removes information, and lack of enough information is what leads to ambiguity.

It is possible that the quality, utility, and relevance of the results and ratings we obtained from our evaluators was substantially decreased by the low number of evaluators we had altogether. In order to gain better results for analysis and ensure that the ability of the individuals rating sets produced by various methods is not disproportionately skewing the data to make it appear that certain methods are better than others, a few things need to be done. The first is that it is essential to have a greater number of people do further ratings on our current results, so that we can do a more accurate evaluation on the usefulness of various methods for articles like that on the Battle of Fredericksburg. Then, we must produce evaluation forms for other articles, which should not be a difficult task, since we have all algorithms in place, and also distribute those to a number of people. It could be that while the basic grammar-only ranking is best for very complex articles like that on the Battle of Fredericksburg for producing acceptable questions, articles that are shorter or longer, simpler or more complex, or more or less technical would have significantly better questions resulting from a different ranking process. If this is the case, these features could be analyzed in the documents and incorporated into the voting algorithm.

Finally, we should attempt to determine what features in a sentence lead to a high level of difficulty. Since the purpose of generating questions is to assist in developing content for education, it would be helpful if it were possible to develop an algorithm that would attempt to sift through a set of questions to find those appropriate for a particular grade level. This may require deeper analysis of not only grammatical structure and Named Entity significance, but also analysis of vocabulary to determine if the general language of a question, or perhaps a full article, is appropriate for a certain age, grade level or reading level.

Acknowledgement

The research reported in this document has been funded partially by NSF grants CNS-0958576 and CNS-0851783.

References

[1] M. Heilman, "Automatic factual question generation from text," Ph.D. dissertation, Carnegie Mellon University, 2011.
[2] R. Chasin, D. Woodward, and J. Kalita, *Machine Intelligence*. Narosa Publishing, 2011, ch. Extracting and Displaying Temporal Entities from Historical Articles, pp. 1–13.
[3] M. Heilman and N. Smith, "Good question! statistical ranking for question generation." Los Angeles: Citeseer, 2010, pp. 609–617.

[4] J. Brown, G. Frishkoff, and M. Eskenazi, “Automatic question generation for vocabulary assessment,” in *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2005, pp. 819–826.

[5] J. Witmer and J. Kalita, “Extracting geospatial entities from wikipedia,” in *Third IEEE on Semantic Computing*. Berkely, CA: ICSC 2009, September 2009, pp. 450–457.

[6] A. Kolcz, V. Prabakarmurthi, and J. Kalita, “Summarization as feature selection for text categorization,” in *Proceedings of the tenth international conference on Information and knowledge management*. ACM, 2001, pp. 365–370.

[7] G. Erkan and D. Radev, “Lexrank: Graph-based lexical centrality as salience in text summarization,” *Journal of Artificial Intelligence Research*, vol. 22, no. 1, pp. 457–479, 2004.

[8] R. Mihalcea and P. Tarau, “Textrank: Bringing order into texts,” in *Proceedings of EMNLP*, vol. 4. Barcelona: ACL, 2004, pp. 404–411.

[9] D. Radev, T. Allison, S. Blair-Goldensohn, J. Blitzer, A. Çelebi, S. Dimitrov, E. Drabek, A. Hakim, W. Lam, D. Liu, J. Otterbacher, H. Qi, H. Saggion, S. Teufel, M. Topper, A. Winkel, and Z. Zhang, “MEAD - a platform for multidocument multilingual text summarization,” in *LREC 2004*, Lisbon, Portugal, May 2004.

Summarization of Historical Articles Using Temporal Event Clustering

James Gung
Department of Computer Science
University of Colorado at Colorado Springs
Colorado Springs, Colorado 80918

***Abstract*—In this paper, we investigate the use of temporal information for improving extractive summarization of historical articles. Our method timestamps every sentence in an article, then clusters the sentences based on their temporal similarity. Each resulting cluster is assigned an importance score which can then be used as a weight in traditional sentence ranking techniques. We cluster thirteen Wikipedia articles describing historical events. Twelve out of thirteen of the clusterings correctly identify the main events of the articles. Temporal importance weighting offers consistent improvements over baseline systems.**

I. Introduction

As the sheer quantity of available information grows, the ability to rapidly locate the salient points in documents becomes increasingly valuable. Manually filtering through large documents for relevant information is a difficult and time-consuming task. The automatizing of text summarization has therefore become an extremely important area of research.

Extractive summarization involves selecting the *k* sentences that best summarize a document. Extensive research has gone into determining which features of text documents are useful for calculating the importance of sentences, as well as how to use these features [1]. Little work, however, has considered the importance of temporal information towards single document summarization. This is likely because many text documents have very few explicit time features and do not necessarily describe topics in chronological order, thus making reliable time feature interpolation for each sentence an impractical expectation.

Historical articles, such as Wikipedia articles describing wars, battles, or other major events, possess characteristics that lend themselves towards time interpolation for each sentence. Historical articles tend to contain many explicit time features relative to other kinds of articles. Additionally, historical articles tend to describe events in chronological order. This further increases the reliability of time feature interpolation.

The motivation of our investigation is based on two basic assumptions pertaining to the structure of historical articles. First, historical articles tend to focus on a single central event, despite mentioning numerous other events of lesser importance. The importance of other events can then be judged by their temporal distance from this central event. Second, important events in an article will be described in greater detail, employing more sentences than less important events. We propose a method that exploits these assumptions.

Given an article where every sentence is assigned an explicit timestamp, we cluster the sentences based on their temporal similarity. That is, each cluster should contain sentences describing events that occurred around the same general timespan. We cluster the sentences by creating a hierarchical set of event boundaries using novelty scores as discussed in [2]. Based on the previous two assumptions, the largest cluster pertains to the central event of the article. Each cluster is assigned an importance score based on cluster size, spread, and distance from the cluster describing the central event of the article.

This paper investigates the value of this temporal-based score towards automatic summarization, specifically focusing on historical articles. We investigate whether or not the score can be used as a weight in traditional sentence ranking techniques to improve summarization quality. For testing purposes, we implement TextRank as a baseline system.

II. Related Work

Considerable work has gone into designing and improving extractive summarization techniques. These techniques look at various features of the text, such as word or phrase frequency, sentence position, or sentence to sentence cohesion.

Event-based summarization is a more recent approach to summary generation. Filatova et al. [3] introduced atomic events as a useful extractable feature for extractive summarization. Atomic events are defined as named entities connected by a relation such as a verb or action noun. Events are then selected for summary by applying a maximum coverage algorithm to minimize redundancy while maintaining coverage of the major concepts of the document. Vanderwende et al. [4] identity events as triples (consisting of two nodes and a relation) similarly to [3]. PageRank is then used to determine the relative importance of these triples represented in a graph. Sentence generation techniques are applied towards summarization, achieving results competitive with extractive summarization. We identify events in sentences for temporal extraction, but consider only one time interval per sentence.

Limited work has explored the use of temporal information for summarization. Lim et al. take advantage of the explicit time information given in multi-document summarization (MDS) for sentence extraction and detection of redundant sentence, ordering input documents by time [6]. They base their technique on the observation that important sentences tend to occur in in time slots containing more documents and time slots occurring at the end and beginning of the documents set. Using traditional methods for extraction of important sentences, they select topic sentences for each time slot, giving higher weights based on the above observation.

Wu et al. use time features towards extractive summarization [7]. They extract events from the text that consist of event elements, the arguments in an event, and event terms, the actions. Each event is then placed on a timeline divided into intervals consistent with the timespan of the article. Each element and event term receives a weight corresponding to the total number of elements and event terms located in each time interval the event element or term occupies. Each sentence is then scored by the total weight of event elements and terms it contains. Encouraging results are reported.

Clustering of events based on time has also received little attention. Cooper et al. investigate clustering towards organizing timestamped digital photographs [5]. They present a method that first calculates the temporal similarity between all pairs of photographs at multiple time scales. These values are stored in a chronologically ordered matrix. Cluster boundaries are determined by calculating novelty scores for each set of similarity matrices. These are then used to form the final clusters. We adopt this clustering method for clustering our timestamped sentences.

III. Approach

A. Overview

The goal of our method is to give each sentence in an article a temporal importance score that can be used as a weight in traditional sentence ranking techniques. To do this, we need to gain an idea of the temporal structure of events in an article. In other words, we want to identify groups of sentences describing events that occurred in the same general timespan. A score must then be assigned to each group corresponding to the importance of the group's timespan to the article as a whole. Each sentence in a particular group will be assigned the same temporal importance score, necessitating the use of a sentence ranking technique to find a complete summary.

B. Temporal Information Extraction

Relatively accurate timestamps for events in an article are needed for this method to be applicable. Timestamp interpolation accuracy depends on the temporal linearity and number of explicit time features in a particular article. Thus, this method's usefulness is dependent on these factors.

For the purposes of this article, we use a temporal expression normalizer to extract the explicit time features. Heideltime is a rule-based system that uses sets of regular expressions to extract time features [8]. Events that occur between each Heideltime-extracted timestamp are naively assigned timestamps consisting of when the prior timestamp ends and the subsequent timestamp begins. This method of temporal extraction is not reliable, but serves the purposes of testing as a reasonable baseline for temporal extraction systems. As the precision increases, the performance of our system should also improve.

C. Temporal Clustering

Clustering is a method for discovering structure in unstructured datasets. To cluster our sentences into temporally-related groups, we adopt a clustering method proposed by Cooper et al. for grouping digital photograph collections based on time.

$$S_K(i,j) = exp\left(-\frac{|t_i - t_j|}{K}\right) \tag{1}$$

Inter-sentence similarity is calculated between every pair of sentences. The similarity measure is based inversely upon the distance between the central time of the sentences (shown in 1). The similarity scores are calculated at varying granularities of time. If the article focuses on a central event that occurs over only a few hours, such as the assassination of John F. Kennedy, the best clustering will generally be found from similarities calculated using a smaller time granularity. Conversely, articles with central events spanning several years, such as the American Civil War, will generally be clustered using similarities calculated at larger time granularities.

The similarities are placed in a matrix and organized chronologically in order of event occurrence time. The resulting matrix is structured such that entries close to the diagonal of the matrix are among the most similar and the actual diagonal entries are maximally similar (diagonal entries correspond to similarities between the same sentences).

To calculate temporal event boundaries, Cooper et al. describe a method for calculating novelty scores [5]. A checkerboard kernel in which diagonal regions contain all positive weights and off-diagonal regions contain all negative weights is correlated along the diagonal of the similarity matrix. The weights of each entry in the kernel are calculated from a Gaussian function such that the most central entries have the highest (or lowest in the off-diagonal regions) values. The result is maximized when the kernel is located on temporal event boundaries. In relatively uniform regions, the positive and negative weights will cancel each other out, resulting in small novelty scores. Where there is a gap in similarity, presumably at an event boundary, off diagonal squares will be dissimilar, thus increasing the novelty score [2]. In calculating novelty scores with each set of similarity scores, we obtain a hierarchical set of boundaries. With each time granularity, we have a potential clustering option.

In order to choose the best clustering, we calculate a confidence score for each boundary set, then choose the clustering with the highest score, as suggested in [5]. This score is the sum of intercluster similarities between adjacent clusters subtracted from the sum of intracluster similarities as seen in Equation 4. A high confidence score then suggests low intercluster similarity and high intracluster similarity.

$$IntraS(B_K)_S = \sum_{l=1}^{|B_k|-1} \sum_{i,j=b_l}^{b_l+1} \frac{S_K(i,j)}{(b_{l+1} - b_l)^2} \tag{2}$$

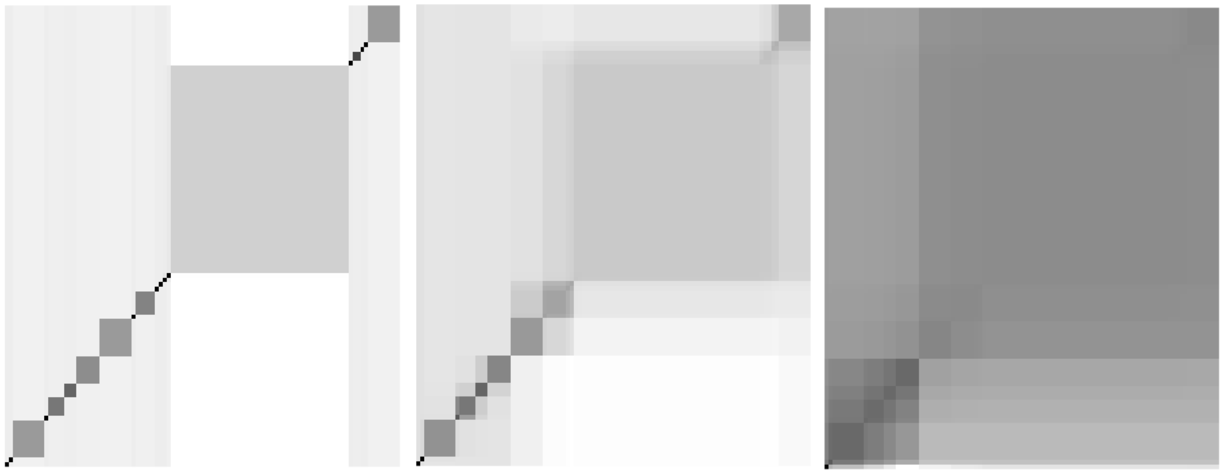


Fig. 1. Similarity matrices at varying *K* displayed as heat maps, darker representing more similar entries. Similarities scores calculated with higher values of *K* correspond to broader time scales (months vs. days). Left to right, *K* is increased by a factor of 10 at each iteration.

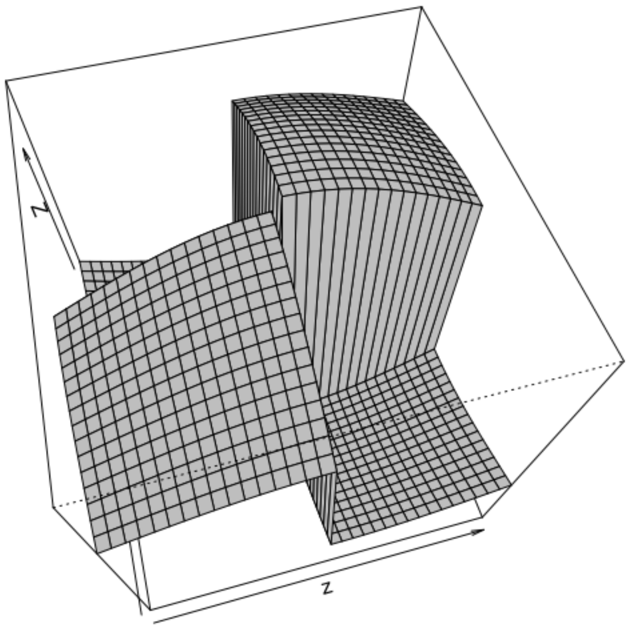


Fig. 2. A gaussian-tapered kernel used to calculate novelty scores. This is slid along the diagonal of each similarity matrix, calculating a novelty score for each sentence. Positive diagonal regions correlate with high intracluster similarity entries. Negative off-diagonal regions multiply by the low intercluster similarity entries, resulting in higher total novelty scores at temporal event boundaries.

$$InterS(B_K)_S = \sum_{l=1}^{|B_k|-2} \sum_{i=b_l}^{b_l+1} \sum_{j=b_l+1}^{b_l+2} \frac{S_K(i,j)}{(b_{l+1}-b_l)(b_{l+2}-b_{l+1})} \tag{3}$$

$$Confidence_S(B_K) = IntraS(B_K)_S - InterSB_K)_S \tag{4}$$

D. Estimating Clustering Paramaters

There are several parameters we must consider before clustering the sentences. Historical articles describing wars will generally have much larger timespans than articles describing battles. Looking at battles at a broad time granularity applicable to wars may not produce a meaningful clustering. Thus, it is worthwhile for us to estimate the temporal structure of each article before clustering. The time granularity for each clustering is controlled by the *K* parameter in the similarity function between sentences. To find multiple clusterings, we start at a base *K*, then increment *K* by a multiplier for each new clustering. We calculate the base *K* using the standard deviation for event times in the article. Measuring the spread of events in the article gives us an estimate of what time scale we should use for measuring similarity.

E. Calculating Temporal Importance

We use three metrics to calculate the importance of a cluster towards a summary. The first metric is based on the size of the cluster (5). This is partially motivated by the assumption that more important events will be described in greater detail, thus producing larger clusters. The second metric (6) is based on the distance from the cluster's centroid to the centroid of the largest cluster, corresponding to the central event of the article. This metric is motivated by the assumption that historical articles have a central event which is described in the greatest detail. The third metric is based on the spread of the cluster (7). Clusters with large spreads are unlikely to pertain to the same event, and should therefore be penalized.

$$Size(C_i) = \frac{|C_i|}{|C_{max}|} \tag{5}$$

$$Sim(C_i) = exp\left(-\frac{|t_{C_i Centroid} - t_{MaxClusterCentroid}|}{m}\right) \tag{6}$$

$$Spread(C_i) = exp\left(-\frac{\sigma_{C_i}}{n * (t_{max} - t_{min})}\right) \tag{7}$$

The parameters *m* and *n* serve to weight the importance of these measures and are assigned based on the spread of events in an article.

The three measures are weighted and multiplied together to obtain a final importance score, working in tandem to ensure that the importance measure will still be valid even if the largest cluster does not correspond to the central event of the article. If the central event is broken up into multiple clusters, they will likely be located nearby each other. If the largest cluster does not correspond to the central event, following our assumption that the central event is described in the greatest detail, it will likely consist of many spread out smaller events, resulting in a greater spread and a decreased importance score. Conversely, clusters corresponding to the central event will be more concentrated relative to the length of the article, and be rated as more important.

F. Final Sentence Ranking

Each sentence is assigned a temporal importance score equal to the importance score of the cluster to which it belongs. To find a complete ranking of the sentences, we need to apply a traditional sentence ranking technique. Any automatic summarization technique that ranks its sentences with specific numerical scores can potentially be augmented with our temporal importance weight.

$$WS(V_i) = (1-d) + d * \sum_{V_j \in In(V_i)} \frac{w_{j,i}}{\sum_{v_k \in Out(V_j)} w_{j,k}} WS(V_j) \tag{8}$$

Existing graph-based methods for sentence ranking apply Google's PageRank algorithm to rank sentence importance [9]. Sentences are represented as nodes on a graph. Some measure of similarity between each of the sentences is calculated, such as cosine similarity or the number of shared open-class words between each pair of sentences [10]. Edges are placed between sentence pairs with similarities above a particular threshold. After giving each node an arbitrary score, the algorithm iteratively calculates the scores of each node based on the scores of neighboring nodes. This score is weighted by the ratio of the weight of the edge between a neighboring node and the current node divided by the total weight of all edges leaving the neighboring node. These weighted scores are then summed to determine the score of the current node (Equation 8). We set a damping factor *d* to 0.85, which in the context of PageRank is used to model the probability of randomly jumping to another page. The scores of each node after convergence indicate the relative importance of each sentence.

$$Similarity(S_i, S_j) = \frac{|\{w_k | w_k \in S_i \& w_k \in S_j\}|}{log(|S_i|) + log(|S_j|)} \tag{9}$$

We choose to use TextRank in our experiments. Our similarity measure is calculated using the number of shared named entities and nouns between sentences as seen in Equation 9. For identification of named entities, we use Stanford NER [11]. It is straightforward to weight the resulting TextRank scores for each sentence using their cluster's temporal importance.

IV. Experimental Results

Evaluation of summaries is traditionally accomplished using ROUGE, Recall-Oriented Understudy for Gisting Evaluation [12]. ROUGE automatically determines the quality of summaries by comparing them to human-created ideal summaries based on measures such as number of overlapping n-grams, word sequences or word pairs. To apply ROUGE, we use human-annotated summaries of the articles we wish to evaluate. These were obtained by asking volunteers to choose what they consider to be the most important sentences from each article.

Using these human-annotated summaries as gold standards, we compare the performance of sentence ranking systems with and without temporal weighting. ROUGE-N computes the number of co-occurring N-grams in the system summary and set of gold standard reference summaries. This is divided by the total number of N-grams in the set of reference summaries. Thus this is a recall-oriented measure. We evaluate using ROUGE-2 bigram matching.

A. Clustering

The Wikipedia articles we test each contain a topic sentence stating the timespan of the main event described by the article. This provides an easy way to determine whether or not a clustering is successful. If the largest cluster contains the timespan of the main event described by the topic sentence, we consider the clustering to be successful (so long as the clustering isn't trivial). The articles vary greatly in length. Also, the ratio of sentences with time features to sentences without is considerably varied. We would expect the temporal-weighted summaries of articles with larger ratios to have more reliable clusterings than those of articles with smaller ratios as they contain more temporal information to interpolate from.

Out of thirteen articles, twelve were successful clusterings by the above criterion. Only Nickel Grass was clustered poorly, the clustering algorithm dividing the main event into two clusters. It is of interest to note that Nickel Grass had one of lowest ratios of sentences containing time features to sentences without, which possibly explains the poor clustering.

B. Temporal Importance Weighting

We test our TextRank implementation on thirteen Wikipedia articles, with and without temporal importance weighting. The articles vary widely in length and ratio of sentences containing time features to sentences without. Each article has at least two human-annotated gold standard summaries for use in ROUGE.

We observe consistent improvements for the articles using the TextRank system with temporal importance weighting over the bare TextRank implementation. In general, articles containing sentences that TextRank ranked highly, but that contain sentences occurring at significantly different times than the central events of the articles observe significant improvements. Although the content of these sentences is highly related to the rest of the article, they should not be included in the summary since the events they contain occur nowhere near the main event temporally.

In 1904, the United Spanish War Veterans was created from smaller groups of the veterans of the Spanish American War.

Fig. 3. An initially highly ranked sentence excluded from the final summary due to low temporal importance

Similarly to the TextRank system, our random ranking system observes small improvements when augmented with temporal importance weighting. The results, however, are more mixed. It is likely that additional human-annotated summaries are necessary for conclusive results. The gold standard summaries widely vary in length and content, displaying the inherent subjectivity and difficulty involved in evaluation.

V. Conclusions and Future Work

The novelty-based clustering method worked extremely well for our purposes. Out of thirteen articles, twelve were clustered such that the temporal bounds of the main events composed or belonged to the largest clusters. These results can likely be improved upon using more advanced temporal extraction and interpolation methods, as our baseline method used a very naive heuristic for interpolating between time features prone to error.

The temporal importance weighting had mixed results with both TextRank and random ranking. The average ROUGE score between all articles was modestly increased, but not significantly. Several single articles showed significant improvement when the ranked sentences were weighted with temporal importance measures. However, improvement was not uniform across all thirteen articles. We attribute decreases in ROUGE scores to poor clusterings. This demonstrates the importance to this method of finding good clusterings, and consequently correctly extracting and interpolating temporal

TABLE I
Results of clustering on *the Battle of Fredericksburg*, only explicit time features.

Centroid: 09/01/1862	
09/01/1862	Although McClellan had stopped Lee at the Battle of Antietam in September, President Abraham Lincoln believed...
Centroid: 11/14/1862	
11/01/1862	Burnside was appointed commander of the Army of the Potomac in November, replacing Maj. Gen. George...
11/09/1862	Burnside, in response to prodding from Lincoln and General-in-Chief Maj. Gen. Henry W. Halleck, planned...
11/16/1862	The Union Army began marching on November 15, and the first elements arrived in Falmouth on November 17.
11/21/1862	By November 21, Lt. Gen. James Longstreet's Corps had arrived near Fredericksburg, and Jackson's was....
11/25/1862	The first pontoon bridges arrived at Falmouth on November 25, much too late to enable the Army of the Potomac...
Centroid: 12/13/1862	
12/13/1862	The Battle of Fredericksburg, fought in and around Fredericksburg, Virginia, from December 11 to December 15...
12/13/1862	The Union Army suffered terrible casualties in futile frontal assaults on December 13 against entrenched Confederate...
12/09/1862	On December 9, he wrote to Halleck, "I think now the enemy will be more surprised by a crossing immediately...
12/11/1862	Union engineers began to assemble six pontoon bridges on the morning of December 11, two just north of the town...
12/11/1862	Eventually his subordinates convinced Burnside to send landing parties over in the boats that evening to secure...
12/12/1862	Over the course of December 11 to December 12, Burnside's men deployed outside the city and prepared to attack...
12/13/1862	The battle opened south of the city at 8:30 a.m. on December 13, when Franklin sent two divisions from the Left...
12/13/1862	By 10 a.m., a thick fog began to lift, and the initially sluggish movements picked up speed.
12/13/1862	The initial assaults west of Fredericksburg began at 11 a.m. as French's division moved along the Plank Road...
12/13/1862	Griffin's division renewed the attack at 3:30 p.m., followed by Humphrey's division at 4 p.m.
12/13/1862	At dusk, Getty's division assaulted from the east and was also repulsed.
12/13/1862	Thousands of Union soldiers spent the cold December night on the fields leading to the Heights, unable to move...
12/14/1862	The armies remained in position throughout the day on December 14, when Burnside briefly considered leading...
12/14/1862	That afternoon, Burnside asked Lee for a truce to attend to his wounded, which Lee graciously granted.
12/15/1862	The next day the Federal forces retreated across the river, and the campaign came to an end.
12/13/1862	Stationed at the stone wall by the sunken road below Marye's Heights, Kirkland had a close up view to the suffering...
12/13/1862	The Cincinnati "Commercial" wrote, "It can hardly be in human nature for men to show more valor...

TABLE II
ROUGE SCORES FOR AN IMPLEMENTATION OF TEXTRANK WITH AND WITHOUT TEMPORAL WEIGHTING.

ROUGE-2		
System	TextRank Weighted	TextRank
Chancellorsville	**0.26495**	0.26305
Chickamauga	**0.23206**	0.22856
Coral Sea	**0.34436**	0.29591
First Barbary	**0.17499**	0.14087
Fredericksburg	**0.12713**	0.05555
Gulf War	**0.33408**	0.32225
Hampton Roads	0.21486	0.21486
Korean War	**0.26084**	0.23666
Nickel Grass	**0.38962**	0.33268
Spanish American	**0.32889**	0.32373
Vicksburg	**0.25000**	0.23118
War of 1812	**0.20970**	0.20960
Whiskey Rebellion	0.21573	0.21573

TABLE III
ROUGE SCORES FOR RANDOMLY SCORED SUMMARIES WITH AND WITHOUT TEMPORAL WEIGHTING.

ROUGE-2		
System	Random Weighted	Random
Chancellorsville	**0.25159**	0.23051
Chickamauga	0.13787	0.16213
Coral Sea	0.17349	0.25397
First Barbary	**0.16882**	0.12637
Fredericksburg	**0.10565**	0.09929
Gulf War	**0.20082**	0.16227
Hampton Roads	**0.29714**	0.19302
Korean War	**0.23441**	0.21803
Nickel Grass	0.14003	0.14003
Spanish American	**0.27194**	0.23872
Vicksburg	0.15585	0.19626
War of 1812	**0.28849**	0.27814
Whiskey Rebellion	0.10308	0.14556

information. Further testing and additional human-annotated summaries are necessary for conclusive results with regard to temporal importance weighting.

It may also be fairly easy to predict the success of using this temporal weight a priori to summarization of an article. A small ratio of explicit time features to sentences (less than 0.15) indicates that the temporal interpolation process may not be very accurate. Many other measures can be considered. The linearity of time features is also a good indication of the success of temporal extraction. More chronological event description will reduce the risk of errors in temporal interpolation. The spread of time features in an article is also a clue to the success of our weighting method. A greater spread indicates that more events will occur farther from the main event of the article necessitating the use of our weighting scheme to filter out unimportant sentences from the summary. Prediction of temporal weighting success would allow for the potential of improving summarization without a great risk of reducing the quality of the summaries by assigning incorrect importance weights.

We have naively assigned a time interval to each sentence. Individual events within sentences are not considered separately. Future work might individually extract events from each sentence, assigning time intervals to each event. For summarization purposes, the most representative event should be chosen for clustering.

Given the success of clustering major temporal events in historical articles, many directions in future work can be taken. It would be useful to augment timeline generation techniques using the hierarchical set of temporal event boundaries produced by the clustering algorithm. Timelines might be constructed at multiple scales, selecting important events representative of each cluster to display at each granularity, allowing the user to progressively zoom in on temporal regions and be provided with more detailed information representative of the region.

Additional sentence importance measures might be explored using the temporal clusterings. Summarization in the vein of maximum coverage, based upon maximally covering topics using a minimal number of sentences, might be explored using temporal boundaries to designate topics.

An alternative approach to clustering the sentences would be to incorporate a content-based similarity measure in the distance measure for the clustering algorithm. This additional dimension would allow for identification of major events that occurred simultaneously but in different clusters. Such an approach would be useful in articles describing events that occurred in parallel.

We presented a method for weighting sentences based on their temporal importance. Sentences are clustered based on the times of events occurring within them. The largest cluster is designated the major event of the article, and other clusters are scored based upon their distance from this cluster, their size, and their spread. Sentences are weighted based on the score of the cluster to which they belong. This weight is used to augment a traditional sentence ranking method, TextRank. We test summarization systems with and without this temporal importance weight, observing modest improvements.

ACKNOWLEDGEMENTS

The research reported in this document has been funded partially by NSF grants CNS-0958576 and CNS-0851783.

REFERENCES

[1] V. Gupta and G. S. Lehal. A survey of text summarization extractive techniques. *Journal of Emerging Technologies in Web Intelligence*, 2(3):258–266, August 2010.
[2] J. Foote and M. Cooper. Media segmentation using self-similarity decomposition. *Proceedings of SPIE*, 5021:167–175, 2003.
[3] E. Filatova and V. Hatzivassiloglou. Event-based extractive summarization. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 104–111, Barcelona, Spain, July 2004. Association for Computational Linguistics.
[4] L. Vanderwende. Event-centric summary generation. In *DUC 2004*, 2004.
[5] M. Cooper, J. Foote, A. Girgensohn, and L. Wilcox. Temporal event clustering for digitial photo collections. *ACM Transactions on Multimedia Computing, Communications and Applications*, 1(3):269–288, August 2005.
[6] J.-M. Lim, I.-S. Kang, J.-H. J. Bae, and J.-H. Lee. Sentence extraction using time features in multi-document summarization. *Information Retrievel Technology*, pages 82–93, 2005.
[7] M. Wu, W. Li, Q. Lu, and K.-F. Wong. Event-based summarization using time features. In *CICLing 2007*, pages 563–574, 2007.

[8] J. Strötgen and M. Gertz. Heideltime: High quality rule-based extraction and normalization of temporal expressions. In *Proceedings of the 5th International Workshop on Semantic Evaluation, ACL 2010*, pages 321–324, Uppsala, Sweden, July 2010.

[9] D. R. Radev and G. Erkan. Lexrank: graph-based centrality as salience in text summarization. In *Journal of Artificial Intelligence Research*, volume 22, pages 457–479, 2004.

[10] R. Mihalcea and P. Tarau. Textrank: Bringing order into texts. In *Proceedings of EMNLP*, volume 4, pages 404–411. Barcelona: ACL, 2004.

[11] J. R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics*, pages 363–370, 2005.

[12] C.-Y. Lin. Rouge: A package for automatic evaluation of summaries. In *Workshop On Text Summarization Branches Out*, 2004.

Pareto Optimal User Interface Design

Sujay Jayakar

***Abstract*—User interface design is more an art than a science, yet developers do not have the luxury of designing unique interfaces for every user's needs. Therefore, in situations where software designers cannot accommodate the entirety of user diversity, optimization based approaches can fill in the gaps. We present a method of designing interfaces using multi-objective optimization. When performing our static optimization computation, we defer forcing trade offs between different objectives, yielding a set of Pareto optimal solutions. Once the user has declared his or her preferences, a secondary ordering on the Pareto set can then find a unique optimal interface for that individual. Since we perform most of the optimization beforehand, we can incorporate global characteristics that real time implementations struggle to capture. The benefit of our approach is largely practical. The computation we perform, the optimization with respect to multiple objectives, could be performed similarly in theory by constraining all of the objectives save one and using the current approaches to single-objective based user interface optimization. This alternative approach, however, is unfeasibly expensive in the face of many, opposing objectives, and exploiting the multi-objective nature of the problem cuts down on the time needed substantially. Furthermore, our approach intelligently distributes the computational burden of optimization between design time and run time, which stands in sharp contrast with the monolithic run time only and design time only approaches.**

I. Introduction

COMPUTER user interfaces are typically designed by hand by software designers, who aim to maximize the utility of their software to their user base. Artificial intelligence researchers have long proposed to leave this optimization to the computer. This suggestion has been met with skepticism from the HCI community. We acknowledge the difficulty of procuring acceptable machine designed interfaces, but there remain situations where such automatically generated interfaces are appropriate. Interface designers typically target a single class of user and platform, with no variance allowed. We propose using the technique of multi-objective optimization to introduce a new level of flexibility in a program's interface. The quality of interfaces is governed by a plethora of objectives, which are often orthogonal and hard to compare. Traditional approaches force a trade off between these objectives during the optimization phase, which may lead to a solution that does not suit the user's preferences. Trade offs may take the form of some linear combination of the different objectives with varying weights: The problem remains of intelligently choosing the weights, and even with all possible weights, single-objective optimization may not identify all optimal solutions. Our solution, on the other hand, defers this trade-off to the end user, whose choices can single out a particular solution for his or her needs.

II. Related Research

Multi-objective optimization is a rich field of research with many novel approaches. The first algorithm we will employ is a multi-objective genetic algorithm entitled *NSGA-II* [1]. Given additional time, we will explore the effects of using alternate algorithms, such as evolutionary covariance matrix adaptation and the Metropolis algorithm [2] [3]. We will use the ECsPy toolkit for multi-objective optimization as a common basis for evaluating these different solvers.

On a more general level, the idea of creating machine-designed interfaces through optimization has an extensive research history. Automated design has been attempted since the 1980s with projects such as Cousin, Mickey, and Humanoid [4] [5] [6]. Zhai designed a keyboard using a sophisticated physically based method derived from the Metropolis algorithm [7]. Hinkle employed genetic algorithms to design keyboards for Brahmic scripts, a difficult problem that involved the allocation of over 3,000 character combinations [8]. Gajos and Weld took a different approach with their Supple software, which designs interfaces in real-time with respect to a user's history [9].

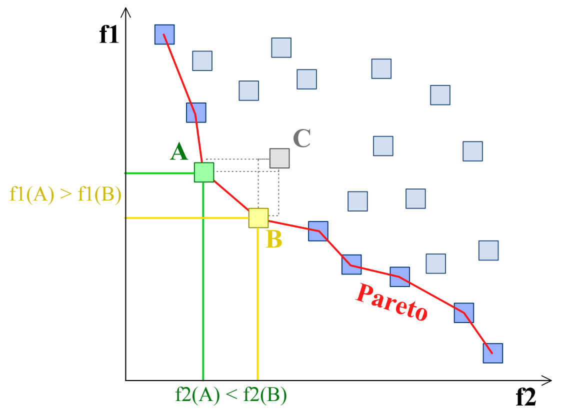


Fig. 1. An illustration of multi-objective optimization with respect to two objectives, f_1 and f_2. The boxes represent feasible solutions, and the darkened solutions form the Pareto optimal set. Note that moving from A to B improves f_2 at the cost of f_1. Source: Wikipedia

III. Problem Description

In this paper, we reduce the task of user interface design to multi-objective optimization. Formally, we have a set of feasible interface designs $\mathcal{S}$ and a sequence of objective functions $f_i : \mathcal{S} \mapsto \mathcal{R}_+$ which we would like to optimize. Using the f_i, we induce a partial order $\succ$ on $\mathcal{S}$ by the typical Pareto criterion. A solution a *Pareto dominates* another solution b if a is at least as good as b with respect to each objective function f_i, and a is strictly better than b for at least one of them. It is straightforward to show that $\succ$ is, in fact, a partial order. From this partial order, we say a solution s is *Pareto optimal* if it is not Pareto dominated by any other solution. Our goal is to find the set of all Pareto optimal solutions $\mathcal{S}^*$, which we will designate as the *Pareto front*. Intuitively, there are no more "free lunches" past the Pareto front: Improving one of the objectives requires worsening at least one of the others.

Once the Pareto optimal set has been identified, the user can supply his or her own ordering $\succ_U$ on the Pareto front, identifying a single optimal solution. The user's preference $\succ_U$ encodes the trade offs the user is willing to make between the different objectives. Since most of the optimization has been done statically ahead of time, identifying the optimal solution with respect to $\succ_U$ is trivial.

For this project, we chose to use evolutionary algorithms to generate the Pareto optimal set. For our results on multi-objective optimization on Indic language keyboards, the set generated was robust to restrictions on individual objectives, meeting our goals of adaptability to user needs. However, our task in general of using multi-objective optimization to design interfaces does not require the use of genetic algorithms, and fairly simple dynamic programming approaches to multi-objective optimization exist [10]. Since our problem representation lends itself well to evolutionary computation, we focused on only the genetic algorithmic approach.

We choose to represent solutions as mappings from abstract interface specifications to concrete widget representations. The core of an abstract interface lies in *primitive types*, denoted by τ, which request integers, strings, floats, and Booleans from users. A designer can add a constraint $\mathcal{C}_\tau$ over the domain of a primitive type to form a *constrained type* $\langle \tau, \mathcal{C}_\tau \rangle$. Types can be composed into *container types* $\{\tau_1, \tau_2, \ldots, \tau_n\}$ that represent some semantic grouping of interface elements.

A potential solution maps an abstract interface representation to a concrete description in terms of widgets. Each platform is a set of GUI widgets that are available to the programmer. An interface description specifies the implementation of each abstract element in terms of widget choice and parameters. Given different objectives, solutions will intelligently choose widgets, preserve container groupings, and tune widget parameters.

IV. Implementation

A. Abstract UI

The user interface design process here begins with the abstract representation of the user interface, provided by the application designer. We provide a simple XML representation along with a parser that creates the Python abstract tree data structure. The allowed container types are `GenContainer` and `IdContainer`. The `GenContainer` type does not enforce any structure on the relationships between its children elements. The `IdContainer` type, on the other hand, imposes the constraint that all of its children must have the same presentation. Both container types take an `ordered` argument as well as a `name`.

The primitive types currently supported are `AbstrInt` for integers, `AbstrFlt` for floating point numbers, `AbstStr` for strings, and `AbstrBool` for Booleans. The abstract integer type takes a `loRange` argument and `hiRange` argument, indicating the lower and upper bounds for the input. If they

```
<?xml version="1.0"?>
<GenContainer name="Classroom Management">
  <IdContainer name="Light Bank" ordered="True">
    <GenContainer name="Left">
      <AbstrBool name="Light"></AbstrBool>
      <AbstrFlt name="Level"></AbstrFlt>
    </GenContainer>
    <GenContainer name="Center">
      <AbstrBool name="Light"> </AbstrBool>
      <AbstrFlt name="Level"></AbstrFlt>
    </GenContainer>
    <GenContainer name="Right">
      <AbstrBool name="Light"></AbstrBool>
      <AbstrFlt name="Level"></AbstrFlt>
    </GenContainer>
  </IdContainer>
  <GenContainer name="A/V Controls">
    <GenContainer name="Projector">
      <AbstrBool name="Power"></AbstrBool>
      <AbstrStr name="Input"
       allowed="Computer 1;Computer 2;Video">
       </AbstrStr>
    </GenContainer>
    <AbstrBool name="Screen"></AbstrBool>
  </GenContainer>
  <AbstrStr name="Vent"
   allowed="Off;Low;Med;High"></AbstrStr>
</GenContainer>
```

Fig. 3. The XML representation of the classroom user interface outlined in Figure 2. Note that the description is entirely abstract: There are no references to size or position other than the semantic groupings provided by the container types.

are not specified, they default to `0` and `10`, respectively. The abstract float type takes the same arguments and defaults to the same values.

The abstract string type serves two purposes. First, if the `allowed` parameter is left empty, the program requests a string from the user in the form of a text input. Second, the developer can indicate a list of allowed choices in the form of a semicolon separated string. Therefore, if the `allowed` parameter is nonempty, the program will present the user with a choice between different strings. The abstract Boolean type does not take any arguments other than its name.

If the developer wishes, he or she can specify a description of the element for internal use by providing a documentation string between the opening and closing tags for a particular element. These do not appear in the final implementation, but may be useful for debugging and development.

Although this approach is limited to static user interfaces, such as control panels and input interfaces, the hierarchical approach derived from combining nested containers with basic action-driven types allows for a great deal of expressive power. Given the flexibility of our abstract framework and the Kivy library[1], adding functionality for dynamic interfaces would not prove to be difficult.

B. Constraints

Fundamental to any genetic algorithm chromosome representation is a set of *constraints* on the legal values of the

[1]http://www.kivy.org

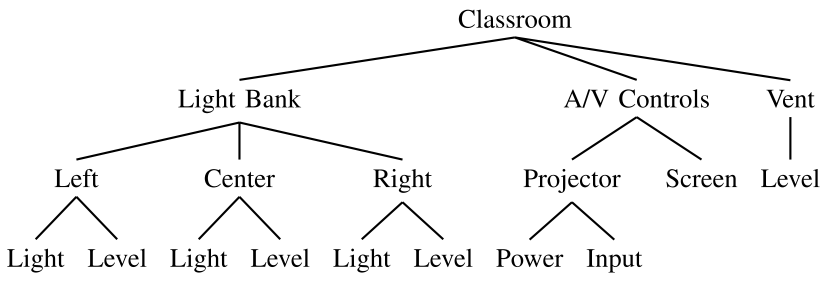


Fig. 2. The abstract representation of the classroom user interface detailed in SUPPLE. Nodes of the tree are container types, represented as sequences of their children. Container types, such as "Light Bank" admit constraints. For example, the presentation of "Left," "Center," and "Right," must all be identical. Leaves of the tree represent primitive types.

alleles. Our set of constraints is especially complicated due to the tree based nature of our problem. Each abstract element can have a set of allowed widgets, and each widget has a set of constraints on its presentation. Therefore, the number of alleles in a given chromosome is not constant, as mutating one widget to another can change the number of alleles. Therefore, we choose to abstract away the details of these constraints to a generic constraint class. This class provides the functionality for storing a constraint's allowed values as well as randomly generating a value for the chromosome.

The constraint class facilitates the choice between both different widget elements and different values for those widgets. Since we have wrapped up the myriad individual constraints within a constraint class, the length of a chromosome remains the same, making the genetic operators much more manageable.

C. Constraint Trees and Arrays

To make the interface between the abstract representation and the chromosomes dictating a particular UI presentation, we need to interpret the abstract tree as a set of constraints on the UI search space. We accomplish this transformation using a collection of rewrite rules on the abstract tree.

We begin by specifying the set of allowed interface widgets and their properties. These choices are largely dictated by KIVY, the user interface library we decided to use for this project. The first is a slider, which has three constraints: orientation, lower bound, and higher bound. The second is a switch, which does not have any constraints. The third is a button, which takes a *group* name as a constraint. Buttons within the same group are presented together, and, much like a radio button, choosing one button disengages any other buttons in its group. The toggle element has a similar group constraint. The final primitive widget is the text input field, which has a constraint indicating whether multi-line input is allowed.

We currently have two layout widgets implemented. The first is the `BoxLayout`, which organizes widgets in a linear fashion. The choice between a vertical or horizontal presentation is encoded as a Boolean constraint. Each child element has a floating point constraint associated that represents the size hint passed to KIVY.

The `GridLayout` layout widget arranges its children widgets in a grid pattern. For a container of n elements, the grid constraint chooses the number of columns c and rows r

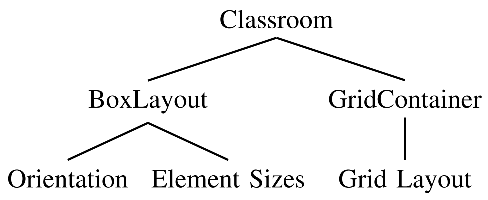


Fig. 4. The UI widgets and constraints associated with a single abstract element. Each element of the constraint tree is paired with sub-tree like this one. Optimizing over trees, much less nested trees, is not straightforward with a genetic algorithm, so we pack the constraint tree into a corresponding constraint array.

such that $r \times c = n$. These factors are passed along as the allowed values for the column and row parameters. Currently, we have chosen to only allow multiples, which may lead to strange results if, for example, the number of children widgets is prime. In this case, the designer should add blank padding elements. At the moment, all columns and rows are required to be the same size, which is obtained by dividing the widget's space evenly.

The abstract tree derived from the XML parse is then traversed using the following rewrite rules.

$$\begin{aligned} GenContainer &\to \{BoxLayout, GridContainer\} \\ IdContainer &\to \{BoxLayout, GridContainer\} \\ AbstrInt &\to \{Slider, TextInput\} \\ AbstrFlt &\to \{Slider, TextInput\} \\ AbstrStr &\to \{TextInput\} \\ AbstrBool &\to \{Toggle, Switch\} \end{aligned}$$

After the rewriting process has completed, each element has a set of associated widgets. Each widget, in turn, has a set of constraints, forming a tree-like structure per abstract element. As noted before, we need to turn this tree into a genetic algorithm friendly array data structure. This process is completed by the `makeArray` method which traverses the tree in preorder, keeping a parent pointer for each element to facilitate tree construction. Since the structure of the tree is preserved under all transformations, the preorder traversal is identical for all chromosomes. This invariant allows us to splice chromosomes together without fear of generating a

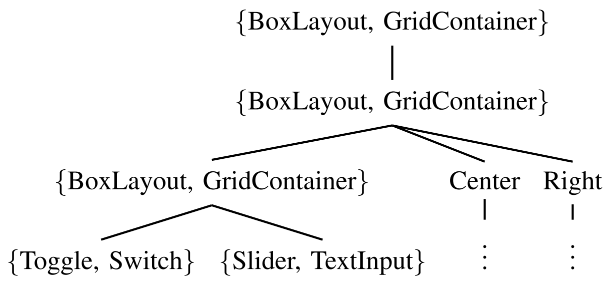

{BoxLayout, GridContainer}
{BoxLayout, GridContainer}
{BoxLayout, GridContainer}
Center
Right
{Toggle, Switch}
{Slider, TextInput}

Fig. 5. A portion of the constraint tree created after rewriting the abstract tree from the XML parse. Each element has a set of associated constraints, as detailed in Figure 4.

malformed tree. This constraint array produced in the tree unwinding is kept throughout the entire algorithm, as generating new nodes under, for example, the mutation operator requires the enumeration of allowed values provided by the constraint array.

D. Genetic Operators and the Algorithm

Once we have this constraint array in hand, we can generate chromosomes randomly using our generator operator, which iterates over the constraints and produces satisfying values. This functionality is implemented by a function class, which internally stores the constraint array as a template for the chromosomes upon initialization. This object is then passed to the ECsPy library[2], which generates the initial population.

The second operator is the combinator, which takes two chromosomes and returns two children chromosomes that are the product of combining its parents. Because of our previous design choices, the implementation of this operator is no more difficult than typical array-based evolutionary operators. Our implementation is a simple implementation of single-point crossover that randomly chooses a crossover point within the chromosome and then splices the two parents in both combinations to produce the two offspring, which are returned to the genetic algorithm.

The final operator is the mutator, which takes a chromosome and alters it with some fixed probability μ, which is set by the user at the beginning of the computation. The mutator operator traverses the chromosome array, marking a node for alteration with probability μ. The nodes marked for alteration are then replaced with a new node instances from their corresponding constraint array node. This approach helps the algorithm explore the search space while ensuring mutated solutions are still legal user interfaces.

The final component remaining for the genetic algorithm is the set of fitness functions, which are the heart of our multi-objective approach. We implemented two fitness functions as examples, but any function can be used, so long as it takes a chromosome and returns a floating point number. The first is the size evaluator, which, roughly, returns the size remaining in the device after the interface has been rendered. More precisely, the evaluator requires that the interface code be in the following format.

[2]http://code.google.com/p/ecspy

```
<GenContainer name = "Device Wrapper">
  <GenContainer name = "Whitespace">
    </GenContainer>
  <GenContainer name = "UI">
    [UI description    ]
    [specified by user ]
  </GenContainer>
</GenContainer>
```

The function then returns the size of the "Whitespace" container, which is to be maximized. This function heavily depends on the size tree implementation in the rendering module, which will be described later. The nature of this objective fits closely with the goal of designing keyboards for mobile devices, which live in a fixed width region on the bottom of a phone's screen. Maximizing the size available to the device for other purposes marks an optimal solution.

The second evaluator is based on Fitts' law, which gives a theoretical estimate of the time needed to navigate the interface [11]. More specifically, it gives us the mean time needed to type two characters using some form of pointing device. The distance between each key is divided by the width of the target key and then added to one. The logarithm, base two, of this result is then weighted by the frequency of this pair of characters and then divided by the "Index of Performance" (IP), which we, in order to maintain consistency with previous research, set at 4.9. Therefore, the mean time to type a character is

$$\bar{t} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{P_{ij}}{IP}\left[\log_2\left(\frac{D_{ij}}{W_j}+1\right)\right].$$

This computation first requires a dictionary of the pairwise distance of the characters. To provide this data, we interface the evaluator again with the rendering module of our library. For each chromosome passed to the evaluator, the function rebuilds the UI tree from the chromosome, traverses the tree, computing the size of each element, and then builds a dictionary mapping element names to their sizes and positions. This dictionary is also used to compute the width data for each element.

The pairwise frequencies are read in from a text file provided by the user. In our implementation for the Assamese language, manipulating Unicode characters proved to be difficult, so we mapped each Unicode character to a unique integer. These integers are then passed around the optimization routine in place of the actual Assamese characters and then substituted in final rendering process.

We used the ECsPy library's implementation of the NSGA II algorithm. The library was flexible enough to accommodate our mutator, combinator, and generator classes described above. Each class was instantiated with the relevant constraint array and then passed to the optimization routine.

Specifically, we used our own mutators, generators, and combinators along with the `generation_termination` terminator and the `file_observer` observer, which intermittently dumped statistics about the evolutionary algorithm as

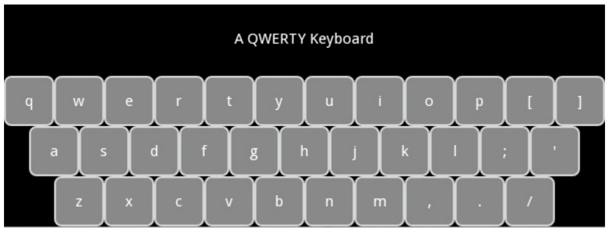


Fig. 6. A simple, abstracted example of a keyboard interface designed in the KIVY UI framework.

well as a list of the individuals in the population to a file. Since we ensured that our operators did not generate any illegitimate solutions, the `nsga2.bounder` was not needed and was set to an identity function.

E. UI Implementation

The KIVY framework for Python provides an excellent cross platform framework for implementing the results from the genetic algorithm. The library's simplicity makes it perfect for our algorithm, which focuses on static interfaces. Another benefit is that KIVY supports both desktop and Android applications, allowing us to perform actual tests on devices with especially relevant size constraints.

There is a direct correspondence between the widgets chosen in the rewrite rules described previously and the widgets available in the KIVY library. From the side of understanding the UI library, we have designed simple demonstrations in an abstract manner that will hopefully interface well with our chromosome specification.

The process of generating a KIVY interface from a chromosome generated by the evolutionary algorithm is involved. As noted before in our discussion on our evaluation functions, the first task is to generate a *size tree* which maps elements to their position and size.

This task is accomplished by first rebuilding the chromosome tree from the linear array specified by the chromosome. This is largely a matter of traversing the array and reconnecting the parent pointers generated from the tree unwinding during the formation of the constraint array. We then proceed from the root node, taking the total interface size from the problem configuration. If the current node is a `BoxLayout` element, the children sizes are determined by their weights w_i.

$$\mathsf{width}_i = \frac{\mathsf{width}_{\mathsf{parent}} * w_i}{\sum w_i}$$
$$\mathsf{height}_i = \frac{\mathsf{height}_{\mathsf{parent}} * w_i}{\sum w_i}$$

Similarly, the permutation information is read from the chromosome. The permutation constraint for a container of n elements is a permutation of the sequence $\{0, 1, \ldots, n-1\}$ that maps the array index of the child to its position. One relevant concern with this approach is that the number of permutations for a container of n children grows as $n!$, which may exceed the period of the random number generator used. If this is the case, the entire solution space may not be explored, which may adversely affect problems sensitive to element permutation (such as keyboards). Fortunately, the Mersenne twister generator used in our implementation has a period that well exceeds $62!$, where 62 is the number of keys in our example.

The case for `GridContainers` is similar. We read in the number of columns and rows from the generated list of factors encoded in the constraint array. For now, the heights and widths of the rows and columns are assumed to be uniform, so determining the sizes and positions of the children is not difficult.

$$\mathsf{width}_i = \frac{\mathsf{width}_{\mathsf{parent}}}{n_{\mathsf{cols}}}$$
$$\mathsf{height}_i = \frac{\mathsf{height}_{\mathsf{parent}}}{n_{\mathsf{cols}}}$$

Finally, if an element is a base type, it simply inherits its size from its immediate parent. Now that the sizes and positions have been determined, we are finally in a position to build the KIVY representation of the interface. The `buildWidgets` method takes the sizes from the size tree previously built and sets the appropriate `size` attributes for the newly generated widget. The method then recursively calls itself on all of the element's children to form the widget hierarchy, which is then passed to KIVY to create an application.

V. RESULTS

We focused our multi-objective approach on the task of designing optimal keyboards for Indic languages to fit in with previous research. More specifically, we aimed to design keyboards for platforms on which space is at a premium. Therefore, our two objectives were speed of use, as determined by Fitts' law, and keyboard size.

As such, we coded an optimized version of our multi-objective algorithm to focus on keyboards alone. We first designed an optimized chromosome that is just an array of integers and floats. The first sixty-two elements are integers that represent the permutation of the keys, as with our abstract approach from before. The next number is a float that represents the normalized height of the device and ranges from zero to one.

A. Keyboard Implementation

Generation of these chromosomes is straightforward, as we simply create an array of integers ranging from 0 to $n-1$, where n is the number of characters in the desired language, and then shuffle the array. Next, we append a random number from zero to one to represent the height of the user interface.

Mutation is similar. With probability μ, we shuffle the first n elements of the array using the built in library function `random.shuffle`. Finally, we choose another random floating point number between zero and one for the height parameter.

We use ECSPY's built in differential crossover combinator to create children chromosomes from parent chromosomes. Since the crossover method does not guarantee that the results will be legitimate permutations of $\{0, 1, \ldots, n-1\}$, we wrote

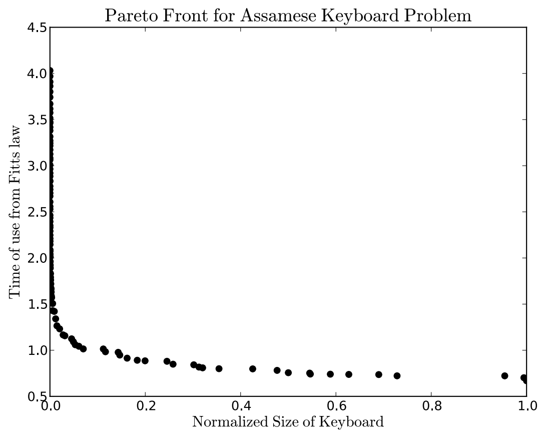


Fig. 7. The Pareto optimal front for an Assamese keyboard design problem on a mobile device. Note that decreasing the mean time per keystroke requires increasing the size of the keyboard, indicating that the two objectives are opposing, as conjectured earlier. Therefore, imposing a maximum size on the keyboard implicitly bounds the maximum efficiency of the keyboard. Now that we have this set in hand, we can, at run time, determine the size constraints for the user and then choose the best solution that satisfies these constraints and maximizes the other objective.

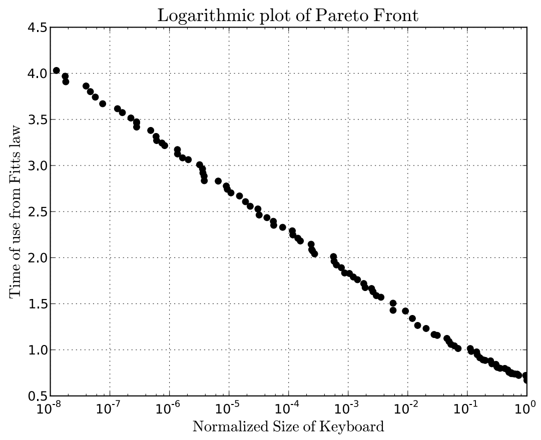


Fig. 8. A logarithmic plot of the Pareto frontier from the previous figure. The trade off between size and efficiency only becomes relevant with smaller sizes, so expanding the horizontal axis renders the plot much more readable.

a *bounder* function that takes an illegitimate chromosome and restores our preferred invariants.

The bounding function first considers the first $n-1$ elements that correspond to the key permutation. It identifies which keys are missing and which keys are repeated. It then randomly assigns missing keys to repeat positions until all keys are present. Finally, it ensures that the final element, the floating point number representing the height of the keyboard, is within the interval $(0, 1]$.

The evaluators used are identical to the Fitts' law evaluator and size evaluation described in the previous section on abstract interfaces. The pairwise frequencies were read in from the Assamese corpus generated from WIKIPEDIA articles.

The Assamese language has 62 characters, all of which must be accommodated on the onscreen keyboard. Each character is associated with a pop up box of conjoints, which were designed by hand and not by the algorithm. This approach was taken from the work by Hinkle, who used a single objective genetic algorithm in a similar spirit [8].

B. Data

We present the Pareto fronts generated by our algorithm running on the Assamese keyboard. There is a clear trade off between size and efficiency, especially once the size becomes especially small. The generation of this set took roughly 12 hours of computation, but once it has been generated, the process of identifying the optimal solution with respect to a particular constraint is instantaneous.

The Pareto frontier was unexpectedly well behaved. Given the complex nature of our optimization problem, we did not expect such a well behaved convex frontier. Furthermore, when viewed on a logarithmic plot, the frontier is linear, which may be related to the log term in Fitts' law and the linear dependence on the height of the chromosome. Although

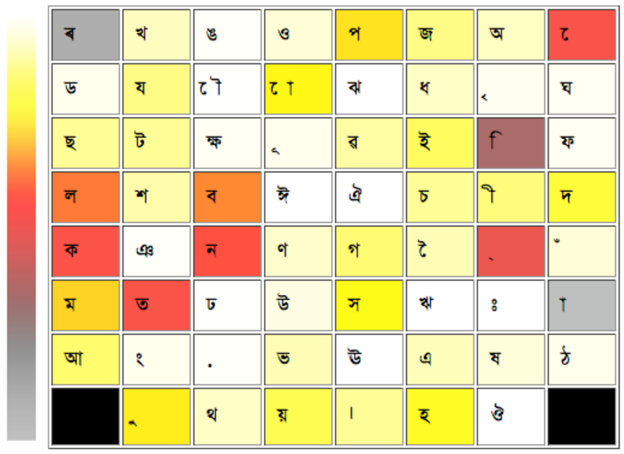


Fig. 9. The keyboard generated when the normalized size constraint of 0.3 was imposed. This keyboard has a fitness of 0.848804 according to Fitts' law, which corresponds to 14.14 words per minute.

we have no proof of convexity, it would be interesting to explore the results of using convex optimization routines, which are much quicker and more robust than our evolutionary algorithms, on this specific optimization problem.

To visualize the results of the keyboard optimization, we developed a special renderer for this problem. We present the keyboard with each key colored according to a heat map that represents the frequency of the character encoded by that key. The heat map legend is presented in the first keyboard. The frequencies of the characters were read in from corpus data and then normalized to one.

After relaxing the size constraint past 0.5, increasing the allowed size of the device did not improve its efficiency, and the optimal solution satisfying the constraint remained the same.

C. Gujarati Keyboard

In addition to running our algorithm on the Assamese language, we developed keyboards for the Gujarati language as well. The Gujarati language has 64 base characters, while

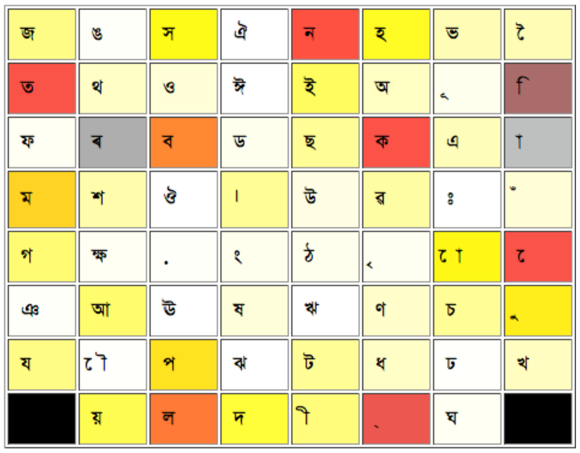


Fig. 10. The keyboard generated when the normalized size constraint of 0.4 was imposed. This keyboard has a fitness of 0.799940 according to Fitts' law, which corresponds to 15.00 words per minute.

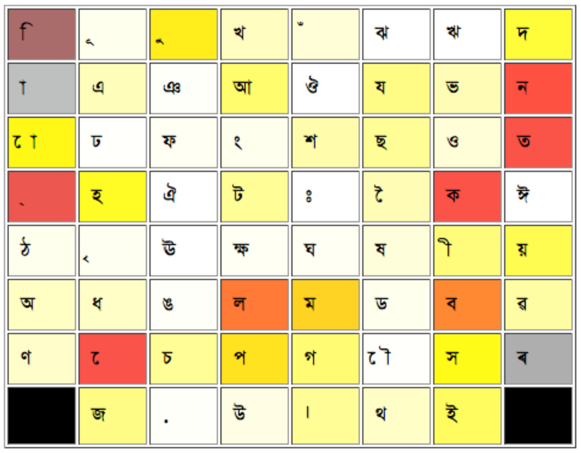


Fig. 11. The keyboard generated when the normalized size constraint of 0.5 was imposed. This keyboard has a fitness of 0.781674 according to Fitts' law, which corresponds to 15.35 words per minute.

Assamese has 62. Therefore, the same 8×8 configuration worked, although the Gujarati keyboard did not have the blank spaces in the corners that the Assamese keyboard did. The mean time per keystroke derived from Fitts' law for the Gujarati keyboards was higher than the solutions for the Assamese keyboards.

We produced similar plots for the Pareto front for the Gujarati keyboard problem. The Pareto front is fairly similar to the Assamese front in that it is convex and exponential. Since the optimization problems use the same chromosome encoding and fitness functions, the similarity is unsurprising.

As with the Assamese problem, relaxing the size constraint past 0.5 did not significantly increase the efficiency of the keyboards. We present three keyboards here in the Gujarati language using the same size constraints from the previous problem.

VI. Conclusions

Casting user interface design as a numerical optimization problem is hardly a new approach, but increasing the richness

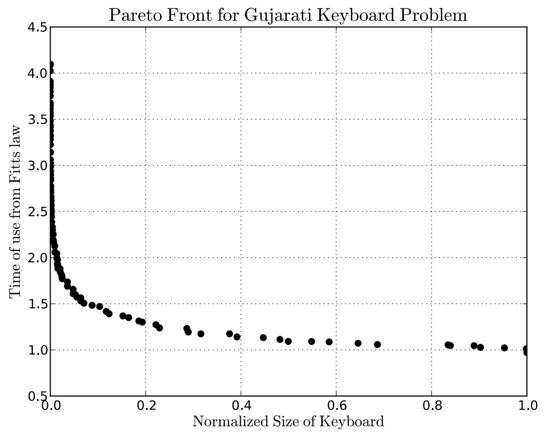


Fig. 12. The Pareto optimal front for the Gujarati keyboard design problem on a mobile device. The front is monotone and convex, just as the Pareto front for the Assamese keyboard design problem.

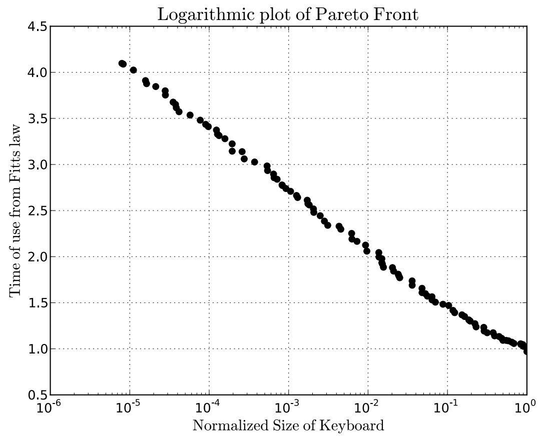


Fig. 13. As before, plotting the Pareto front on a semi-logarithmic plot reveals its simple exponential figure.

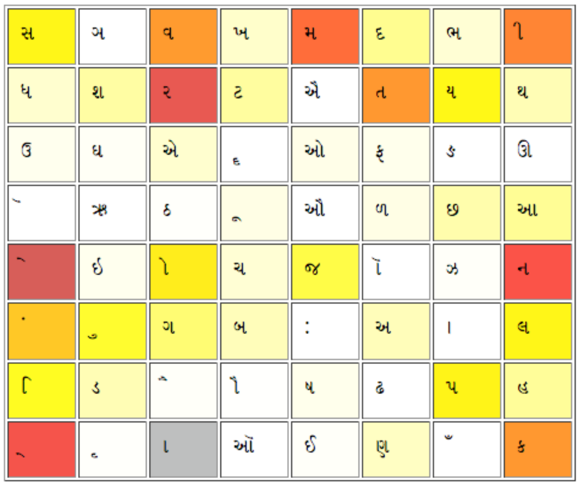


Fig. 14. The Gujarati keyboard generated under the size constraint of 0.3. This keyboard has a mean time per keystroke of 1.193138 seconds, which corresponds to 10.06 words per minute.

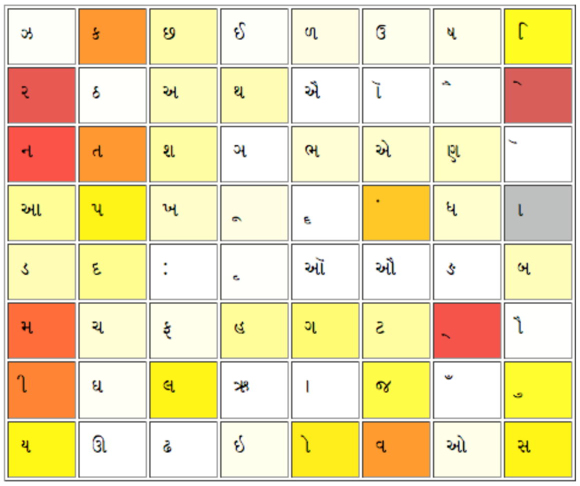


Fig. 15. The Gujarati keyboard generated under the size constraint of 0.4. This keyboard has a mean time per keystroke of 1.140539 seconds, which corresponds to 10.52 words per minute.

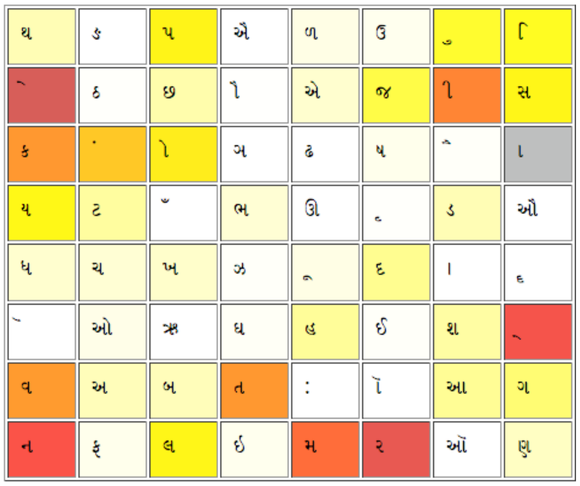


Fig. 16. The Gujarati keyboard generated under the size constraint of 0.5. This keyboard has a mean time per keystroke of 1.091797 seconds, which corresponds to 10.99 words per minute.

of the optimization by simultaneously considering multiple objectives has produced offline algorithms that, in spite of their offline nature, can still be responsive to users' needs. This approach yields a good compromise between entirely static methods that entirely determine the interface ahead of time and fully real-time approaches whose objectives must conform to their heuristics.

Our results with keyboard design confirmed our hypothesis: Multi-objective evolutionary algorithms can identify a Pareto front that represents trade offs between the opposing objectives. Constraining all but one of the objectives then provides an efficient way to optimize relative to the users' needs while benefiting from the flexibility of static offline optimization algorithms.

Acknowledgments

The research reported in this document has been funded partially by NSF grants CNS-0958576 and CNS-0851783.

References

[1] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, April 2002.

[2] C. Igel, N. Hansen, and S. Roth, "Covariance Matrix Adaptation for Multi-objective Optimization," *Evolutionary Computation*, 2007.

[3] J. Vrugt, H. Gupta, W. Bouten, and S. Sorooshian, "A Shuffled Complex Evolution Metropolis Algorithm for Optimization and Uncertainty Assessment of Hydrologic Model Parameters," 2003.

[4] P. J. Hayes, P. A. Szekely, and R. A. Lerner, "Design Alternatives for User Interface Management Systems Based on Experience with Cousin," in *CHI '85: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, 1985, pp. 169 – 175.

[5] D. R. Olsen, "A Programming Language Basis for User Interface," in *CHI '89: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, 1989, pp. 171 – 176.

[6] P. A. Szekely, P. Luo, and R. Reches, "Facilitating the Exploration of Interface Design Alternatives: The Humanoid Model of Interface Design," in *CHI '92: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, 1992, pp. 507 – 515.

[7] S. Zhai, M. Hunter, and B. Smith, "The Metropolis Keyboard–An Exploration of Quantitative Techniques for Virtual Keyboard Design," 2000.

[8] L. Hinkle, M. Lezcano, and J. Kalita, "Designing Soft Keyboards for Brahmic Scripts," in *International Conference on Natural Language Processing*, Kharagpur, India, 2010, pp. 191–200.

[9] K. Gajos, D. Weld, and J. Wobbrock, "Automatically Generating Personalized User Interfaces with SUPPLE," *Artificial Intelligence*, 2010.

[10] R. Kltzler, "Multiobjective dynamic programming," *Optimization*, vol. 9, pp. 423–426, 1978.

[11] P. Fitts, "The information capacity of the human motor system in controlling the amplitude of movement," *Journal of Experimental Psychologyu*, 1954.

An Unsupervised Method for Generating Parallel Corpora for Medium and High Density Languages

Max Kaufmann

Abstract—**Parallel corpora are an extremely useful tool in many natural language processing tasks. However, it is often difficult, impossible, or expensive to obtain parallel corpora for certain language pairs. We propose an unsupervised method that is capable of automatically creating high quality parallel corpora, as well as bilingual lexicons. The outlined method is also capable of automatically augmenting the size of the corpora. The resulting parallel corpora are made freely available for download.**

Index Terms: **Wikipedia, Parallel Corpora, DBpedia**

I. Introduction

The Wikipedia project has resulted in an unprecedented amount of collaboration to create an astounding number of authoritative documents. It is a one of the largest free collections of humans knowledge in existence, and because of this it has received much media attention. But there is another aspect of Wikipedia that receives much less attention. Wikipedia is a large, if not the largest, multilingual repository. Articles in Wikipedia, especially for the more popular topics, are often written in a multitude of languages. However, these articles are not direct translations of each other, so Wikipedia is not a truly parallel corpora. Previous literature has referred to it as a document-aligned corpora or noisy parallel corpora [1] . While the articles are usually semantically similar, they can vary widely in their structure as well as syntax. This makes the task of creating parallel corpora from these articles fairly difficult. However, several projects such as DBpedia and OmegaWiki have organized the knowledge available on Wikipedia in a format that makes it easier to extract aligned sentences. By utilizing these resources, we can build a collection of parallel corpora for a large number of language pairs. In this paper, we will attempt to extract all information that would aid in NLP tasks involving two languages, such as machine translation or relationship extraction. The end result is a set of parallel corpora as well as a bilingual lexicon for each language pair.

II. Motivation

Parallel corpora are used in all types of multilingual research, especially in Machine Translation (MT) tasks. However, creating these corpora is an arduous task for several reasons. Creating parallel corpora requires humans who are proficient in both the source and target language(s). These translators usually are compensated in some form, which means that parallel corpora generation is both expensive and time consuming. Given that the strength of any MT system, particularly statistical MT systems, hinges largely on the quality and size of parallel corpora, we believe that automatic parallel corpora generation could greatly aid the task of machine translation by reducing the dependency on human generated corpora. The costs of creating parallel corpora are further exacerbated by the fact that as time goes on, new corpora have to be generated to account for changes in both languages. A parallel corpora generated 100 years ago would have far less utility in a machine translation task than a corpora made in the past 5 years.

The corpora generated by our approach attempt suffer much less from the previously mentioned flaws. The approach which we will discuss creates parallel corpora and lexicons by extracting and organizing information from two sources: DBpedia and OmegaWiki. Both of these projects are currently active, which suggests that as time goes on they will continue to grow in size and accuracy. The only costs associated with creating parallel corpora from these sources are the initial cost of creating the programs that create the corpora, and the computational cost of running them. These costs are minor compared with the expense invested in paying human translators to generate the parallel corpora.

Our research is further motivated by the fact that, despite the importance of parallel corpora, they are somewhat difficult to obtain. Despite the problems previously discussed, there are many parallel corpora available on the web. However, there are several problems with these corpora. First of all, they are not always free. The Linguistic Data Consortium[1] is one the largest unified providers of parallel corpora. However, membership is required to download these corpora. The cheapest membership is $1,000, while the most expensive is $24,000. The second issue is that the majority of these parallel corpora only exist for more popular language pairs. When attempting to translate from two very popular languages, such as English-German, finding parallel corpora is not difficult. But finding parallel corpora for languages pairs such as Thai-Greek is considerably more difficult. The parallel corpora resulting from this research will include corpora for small languages, and potentially language pairs for which parallel corpora do not exist. The size of the corpora may be small, depending on the size of the Wikipedias of the respective languages, but due to the self-modifying nature of these parallel corpora, they will be capable of automatically growing as their Wikipedias grow.

III. Previous Work

As far as we are aware, there have been no public projects which attempt to extract parallel corpora from this particular combination of resources. However, several projects have tackled the issue of multilingual retrieval from Wikipedia to build parallel corpora. One such example is [2]. They tested two approaches for aligning sentences from Dutch-English Wikipedia. Their first approach used an already existing SMT system to translate the Dutch article into English. They then compared the similarity of the sentences between the English article and the Dutch article (which was translated into English). The second approach create a bilingual lexicon by leveraging the fact that articles across languages translated the titles into English. They then used this lexicon and compared the sentences by computing their lexical similarity. While computing lexical similarity is an extremely effective way to measure multilingual sentence similarity, as evidence by [3], using an already made machine translation system appears rather impractical. The purpose of parallel corpora generation is to create a statistical machine translation, and if one already exists, and is capable of doing a decent job of translating a Wikipedia article, then there seems little utility in generating parallel corpora. This approach also lacks extensibility, because one cannot always assume that a MT system will already exist to aid in the translation. In their study, they found that the bilingual lexicon approach was, unsurprisingly, less accurate then the MT approach. This is potentially due to the fact that lexical similarity, especially when computed with an incomplete lexicon, is not powerful enough to truly capture sentence similarity. One of the main problems with their approach is that too many possible sentences candidates were generated. They compared every sentence in a Wikipedia article to every sentence in the L2 article, resulting in 80 million candidate pairs [2].

[3] used the lexicon methodology outlined in [2], but also attempted to utilize character lengths to compute sentence similarity to align Persian-English Wikipedias. They used the statistical model created by [4] to figure out the approximate correlation between the length of an English sentence and the length of a Persian sentence. Doing this, they were able to discard translation pairs that were too long or too short to actually be translations. This approach shortens the number of possible candidates significantly, which solves the problem of overmatching presented in [2].

Both of the previous approaches have only dealt with aligning one language pair, and it has always been from L1-English or English-L1. This phenomenon is not unique to these papers, the large majority of Wikipedia alignment work has focused on aligning Wikipedia in one language to English Wikipedia. Even projects that have used more than 2 languages, such as [1] have always had English as one of the languages. [1] attempted to align Spanish-English, German-English and Bulgarian-English Wikipedias by building a Maximum-Entropy classifier to determine if sentences from aligned articles were parallel or not. However, their classifier was trained on seed parallel data. Seed data may not always available, especially for small language pairs, and so their approach is not as general as the one we will outline here. We will use techniques similar to theirs for extracting parallel sentences from Wikipedia, but with seed

[1]http://www.ldc.upenn.edu/

data that we have generated ourselves.

IV. Resources

We aim to produce high quality parallel corpora for a wide variety of language pairs, and to achieve that we leveraged two sources of comporable corpora: DBpedia[2] and OmegaWiki[3]. These resources are described in more detail below:

A. OmegaWiki

OmegaWiki describes itself as a c̈ollaborative project to produce a free, multilingual dictionary in every language, with lexicological, terminological and thesaurus information.̈ It has entries for many sense-disambiguated words (each sense-disambiguated words is called an ëxpression)̈. Each entry includes a translation of the expression and its definition in several languages. An entry may also include information for certain grammatical information about an expression such as its POS tag, gender, corresponding Wikipedia article, and hypernms/hyponyms. A database including all of this information is freely downloadable. We have downloaded this database, and have created a program that, given a language pair, can extract all of the expressions in both languages, as well as the corresponding grammatical information available. We also extract the definitions of each expression and if they are available in L1 and L2, add them to our parallel corpora.

B. DBpedia

The DBpedia project is an attempt to take the data in Wikipedia and turn it into a structured representation of knowledge. We exploit the fact that during their attempts to turn Wikipedia into an ontology, DBPedia generates automatically generates short and long summaries, called abstracts for each entry in the ontology. While these abstracts are not entirely parallel, they may contain parallel data. Our approach leverages these abstracts, and attempts to find parallel sentences using the method escribed in the next section. DBpedia contains both long abstracts, which may be 5-10 sentences in length, as well as short abstracts, which are generally 1-2 sentences. The DBpedia ontology also has a l̈abelf̈ields from which we can extract additional vocabulary of L1 and L2. The l̈abelf̈ield which contains the Wikipedia title of the article for every language it is available in. The Wikipedia article titles are direct translations.

[2]http://wiki.dbpedia.org/

[3]http://www.omegawiki.org

V. Parallel Corpora Generation

There are several differences between the two resources we are exploiting (DBpedia and OmegaWiki). The first is that DBpedia contains a large amount of noisy data, while OmegaWiki contains a much smaller amount of clean data. This is due to the fact that DBpedia data is generated automatically, while OmegaWiki data is created by human contributors. Therefore, our approach will first extract the higher quality data from OmegaWiki, and then use that data to aid in gathering data from DBpedia. As previously stated, the two fields that contain helpful information in OmegaWiki are the word translations, and definitions. Therefore, we first extract all parallel words and definitions for a given language pair. We then leverage the parallel words extracted from OmegaWiki to build a bilingual lexicon. However, for smaller languages, this method does not often yield a large lexicon.

Since OmegaWiki does not provide enough data for multilingual experiments, we turn to DBpedia as an additional source of parallel data. DBpedia itself does not contain any parallel data, however there is comparable data that we can exploit to gain parallel data. We posit that since the a pair of abstracts for a given entry in DBpedia are semantically similar, we might find syntactically parallel sentences in these abstracts to add to our corpora. However, parallel data will not necessarily be found in corresponding sentences, so extraction is not a trivial task. If we are attempting to extract English and Chinese parallel sentences, the first sentence in the English abstract may be parallel to the third sentence in the Chinese abstract. To solve this issue, we align each sentence in an L1 DBpedia abstract to every sentence in the corresponding L2 DBpedia abstract. This guarantees that all potential parallel sentence pairs are considered. However, this alignment suffers from two problems. First, it requires that the abstracts, which are in paragraph form, are split into sentences. To solve this problem, we use the Lingua sentence splitter[4], which has support for

[4]http://code.google.com/p/corpus-tools/

splitting Catalan, Dutch, English, French, German, Greek, Italian, Portuguese, and Spanish sentences. The second issue that arises from aligning sentences between abstracts is that it creates a significant amount of non parallel data. If we are aligning two abstracts, one with 4 sentences, and one with 7, we generate 28 sentence pairs, even though there can only be 4 parallel sentences. To solve this, we use HunAlign, a multi lingual open source parallel sentence aligner.

A. *HunAlign*

HunAlign [5] was originally created to detect whether sentences between Hungarian and English are parallel. However, the algorithm has been extended to work for any language pair. HunAlign primarily uses Gale-Church sentence alignment algorithm, which detects parallel sentences by comparing their length. However, if a bilingual dictionary exists, it can be used to compute the lexical similarity of sentences. We construct a bilingual lexicon by first getting all of the parallel words for a given language pair from OmegaWiki. We then concatenate the ïlabelT̈field from DBpedia, which represents the titles of the article. We use this bilingual lexicon as the dictionary for HunAlign. We then run HunAlign on all of the sentence pairs generated in the previous step. HunAlign then generates a probability, p, between 0 and 1 which represents the likelihood that the sentences are parallel pairs. To ensure that our translations are truly parallel, we only accept values of $p > .99$. The end result is a set of parallel sentences between a given language pair.

VI. Results

We tested our method on several language pairs. Our goal was to create to a system which was capable of creating parallel corpora for uncommon language pairs. Since our system did not require any linguistic knowledge of the language pairs, excluding the ability to determine where sentences ended, we expected that it would perform equally well on extracting parallel data from similar and distinct languages. For the purposes of this experiment, we define languages as similar languages as languages from the same family (e.g., romance, germanic, hellenic) and distinct languages as languages from different families. Below are tables showing the data generated for several language pairs from 3 language families. The language pairs are described using their ISO 639-1 codes.

Language Pair	Parallel Words	Parallel Sentences	Family
DE-AF	2,151	5,559	Germanic
DE-SV	31,918	17,482	Germanic
EN-DE	69,210	13,213	Germanic
EN-ES	343,416	51,421	Mixed
EN-HI	17,234	1,550	Mixed
PL-UK	99,758	7,150	Slavic
RU-IT	51,496	11,251	Mixed

VII. Evaluation

From Table 1, we can conclude several things about the effectiveness of our method. Overall, more parallel sentences tend to be produced when the two languages are from similar families. The exception to this is EN-ES, which is probably due to the fact that the English and Spanish are the first and third largest Wikipedia's, respectively. This doesn't reflect on the effectiveness of our method, and is probably just due to the initial sizes of the corpora. It is also interesting to note that RU-IT performed significantly well, despite them being languages from different families. This is probably due to the size of the corpora.

A. *Comparison to Other Corpora*

It is difficult to evaluate the quality of a parallel corpora by any other metric other than looking at its size. Therefore, we will compare our corpora with two existing free parallel corpora.

There are several parallel corpora which are freely available. These corpora were all generated automatically. The smallest of these is Europarl, which contains aligned sentences between 11 European languages and English [6]. This corpora suffers from the same flaw as [2] and [3], which is that it only contains data aligned between English and another language.

The second is JRC-ACQUIS [7]. [7] contains parallel corpora for 22 languages, which were generated by running HunAlign on a series of EU documents. Below is a table comparing the size of the parallel data generated using our algorithm to the size of the two previous parallel corpora.

Despite the fact that we offer smaller parallel corpora than the two previous sources, there are several features that our corpora has that the previous corpora lack. First, our corpora are self-updating, meaning that its size grows automatically. Both DBpedia and OmegaWiki are constantly increasing the size of their databases, and making that data publicly available. Our algorithm can use this updated data to generate bigger parallel corpora. Secondly, both of these corpora were generated from European Union legal documents. This means that, while they offer substantial amounts of data, it does not cover a wide variety of domains. Our corpora are based on Wikipedia, which covers many domains.

VIII. Future Work

The method used to generate parallel corpora in this paper, while only applied to several language pairs, is highly extensible. DBPedia has 97 languages, which means that there are a massive number of parallel corpora that could be generated using this algorithm. The approach we outlined here performs fairly well in extracting data from similar language pairs, but future work could improve its robustness on languages from distinct languages. One way to do that involves translating through three languages to create artificial parallel data. For example, if we have a large volume of English-Hindi sentences, and a large volume of English-Bengali parallel sentences, it would be possible to create new Bengali-Hindi aligned sentences by translating through English as an intermediary language.

Another way to improve this system would be to actually exploit Wiktionary to add grammatical data to the parallel corpora. Wiktionary contains translations, which could augment the size of the generated corpora, but it also contains grammatical information, such as case, part of speech, and conjugations for its entries. This grammatical information could be incorporated into HunAlign to help decide whether sentences are parallel or not, and it could be incorporated into the published parallel corpora to make them more useful to researchers.

IX. Contributions

In this experiment, we have outlined an algorithm for extracting parallel sentence data from DBPedia and OmegaWiki. This algorithm is capable of extracting parallel data from comparable documents with almost no previous linguistic knowledge. We have produced large parallel corpora and lexicons for common language pairs, and medium sized corpora and lexicons for distinct language pairs. We have created several parallel corpora for languages which previously did not have parallel corpora.Furthermore, these corpora are freely available and can be used in many multilingual NLP tasks. This algorithm extracts data from resources which are continually growing, so as these resource grow, so will the size of the parallel corpora and lexicons generated.

References

[1] J. R. Smith, C. Quirk, and K. Toutanova, “Extracting parallel sentences from comparable corpora using document level alignment,” Stroudsburg, PA, USA, pp. 403–411, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?id=1857999.1858062

[2] S. F. Adafre and M. de Rijke, “Finding similar sentences across multiple languages in wikipedia,” in *11 Conference of the European Chapter of the Association for Computational Linguistics*, 2006.

[3] M. Mohammadi and N. GhasemAghaee, “Building bilingual parallel corpora based on wikipedia,” in *Proceedings of the 2010 Second International Conference on Computer Engineering and Applications - Volume 02*, ser. ICCEA ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 264–268. [Online]. Available: http://dx.doi.org/10.1109/ICCEA.2010.203

[4] W. A. Gale and K. W. Church, “A program for aligning sentences in bilingual corpora,” in *Proceedings of the 29th annual meeting on Association for Computational Linguistics*, ser. ACL ’91. Stroudsburg, PA, USA: Association for Computational Linguistics, 1991, pp. 177–184. [Online]. Available: http://dx.doi.org/10.3115/981344.981367

[5] K. Tóth, R. Farkas, and A. Kocsor, “Sentence alignment of hungarian-english parallel corpora using a hybrid algorithm,” Szeged, Hungary, Hungary, pp. 463–478, January 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1466514.1466522

[6] P. Koehn, “Europarl: A multilingual corpus for evaluation of machine translation,” University of Souther california, Tech. Rep., 2002.

[7] R. Steinberger, B. Pouliquen, A. Widiger, C. Ignat, T. Erjavec, and D. Tufiş, “The jrc-acquis: A multilingual aligned parallel corpus with 20+ languages,” pp. 2142–2147, 2006.

Towards Time Inference for Timeline Generation of Historical Wikipedia Articles

Michael Mara
Computer Science Department
Williams College
Williamstown, MA 01267
Email: mtm1@williams.edu

***Abstract*—In this paper, we combine several temporal information extraction techniques to generate timelines from historical Wikipedia articles. We extend HeidelTime to significantly increase precision and recall of temporal expression extraction in our domain, while modestly increasing performance in other domains. We then use a Markov Logic Network acting on time points, and a simple heuristic to assign explicit time intervals to every event in an article, improving performance significantly from prior work.**

I. Introduction

This paper involves temporal information extraction and inference. Information extraction (IE) is the problem of extracting easily processed relational data from unstructured plain text. Research in this area has increased to keep pace with the ever-growing body of information-rich natural language content available on the internet. Temporal information extraction and inference from plain text is a critical and daunting task for natural language processing researchers, but as Ling and Weld [1] note, most information extraction research ignores the temporal information inherently tied to any non-static facts. They go on to acknowledge the literature on temporal expressions in natural language is vast , but mostly focus on specific subproblems, rather than tackling temporal IE as a whole. This paper will attempt to tackle full temporal IE. The motivating application is improving the accuracy of the automated timelines generated by EVOGATE, developed by Chasin and Woodward, which is currently used to create a map and timeline of events from historical wikipedia articles [2]. This involves very accurate temporal expression extraction, and accurately inferring the time intervals of events in plaintext.

II. Related Research

A focal point for temporal information extraction research was the 2007 TempEval challenge [3], which asked participant to focus on three tasks:

Task A	For a restricted set of event terms, identify temporal relations between events and all time expressions appearing in the same sentence. (NOTE: The restricted set of event terms is to be specified by providing a list of root forms. Time expressions will be annotated in the source, in accordance with TIMEX3.)
Task B	For a restricted set of event terms, identify temporal relations between events and the Document Creation Time (DCT). (NOTE: The restricted set of events will be the same as for Task A. DCTs will be explicitly annotated in the source.)
Task C	Identify the temporal relations between contiguous pairs of matrix verbs. (NOTE: matrix verbs, i.e. the main verb of the matrix clause in each sentence, will be explicitly annotated in the source.)

Many approaches have yielded fair success on these tasks or similar tasks, such as Conditional Random Fields, Markov Logic Networks, and rule-based approaches [4]. Yoshikawa et al. [5] succeeded in getting the highest F-score on all three tasks by jointly solving them with Markov Logic Networks. However, solving these tasks is not sufficient for generating a useful timeline from plaintext.

One drastically limiting factor is the relations specified by the TempEval guidelines are a small subset of the Allen interval-algebra relations, which are qualitative relations between two sets of time intervals, such as BEFORE, or OVERLAPS. [6] This is obviously insufficient for timelines, which require exact points in time or time intervals, specified by either a single time or a beginning and end time, for every event displayed.

According to the official Wikipedia style guide, temporal expressions relating to the time of authorship should be avoided except for current events articles [1]. Thus Task B is not highly relevant for our purposes, and our approach this summer disregards document creation time all together. Tasks A and C, however, will certainly be a part of any complete temporal extraction scheme.

[1]http://en.wikipedia.org/

Other promising approaches to temporal information extraction not focused specifically on the TempEval tasks include the hybrid Markov Logic Network and deep semantic parsing approach of Ling and Weld [1] for developing the tightest set of temporal bounds for every event in an article directly implied by the text. One major emphasis of Ling and Weld's work is that they assigned times to events only if the times were directly implied by the text. For many applications, a sufficient criterion is to be reasonably sure of time bounds. Wikipedia in particular, by its nature of being editable by anyone, is subject to factual errors (though not significantly more than standard encyclopedias [7]). Thus direct implication may be a stricter criterion than we would want for developing timelines based on information in Wikipedia.

Another unique approach is aggregating massive amounts of search data from the web combined with shallow parsing to create fuzzy sets representing time intervals for events. [8] This approach is notable for explicitly encoding the uncertainty of time bounds of events.

III. Problem Description

Our problem is to create a program to take a Wikipedia article as input, and output the events in the text along with an explicit temporal interval for each one. This output can then be displayed by the EVOGATE system on a timeline. In order to accomplish this, we will need to extract the sparse temporal information in the text and extrapolate the start and end points for each event.

IV. Temporal Expression Extraction

A. HeidelTime

We use HeidelTime, developed by Strotgen and Gertz[9], to extract all temporal expressions from the wikipedia articles. HeidelTime is an easily extendable rule-based temporal expression extraction system based on regular expressions. Each extracted expression is annotated with its type, value, and the rule it was found with. In the 2010 TempEval-2 challenge[2], HeidelTime was used to tackle task A: Determine the extent of the time expressions in a text as dened by the TimeML timex3 tag. In addition, determine value of the features type and val. The possible values of type are time, date, duration, and set; the value of val is a normalized value as dened by the timex2 and timex3 standard.

Two versions of HeidelTime were tested, one optimized for precision, and one optimized for recall. In the extraction portion, both versions of HeidelTime outperformed their competitors, with an F-score of 86%. In addition, the precision-optimized version had the highest accuracy in assigning the value attribute (85%), as well as a respectable accuracy of (96%) for the type attribute. The publicly available HeidelTime version is a slightly improved version of the precision optimized version submitted for TempEval-2. The results for all three versions can be seen in Table **??**.

[2]http://semeval2.fbk.eu/

HeidelTime uses hand-crafted rules for temporal expression extraction and normalization. The extraction rules are based on regular expressions, but can also take into account part-of-speech (POS) constraints on the tokens extracted. For this reason, HeidelTime is used in a pipeline, with a POS tagger directly before it. Every extraction rule is coupled with a normalization rule, which takes the extracted expression and produces a normalized value as defined by the timex2 and timex3 standard. Extraction rules make use of expression resources, which are simply reusable regular expressions that are categorized and named. For example, the expression resource reSeason (invoked in the extraction rules as "%reSeason") is

$$reSeason = ([Ss]pring|[Ss]ummer|[Ff]all|[Aa]utumn|[Ww]inter).$$

Note that '[]' denotes that exactly one of the expressions inside occurs, and '|' is a boolean OR. Thus "%reSeason" matches the appropriate strings denoting seasons, such as "spring" and "Autumn."

Normalization rules take the extraction rules and create an expression representing the value of the extracted expression. They make use of normalization resources. Normalization resources are just a mapping between extracted expressions and their normalized value. For example, a portion of normSeason is:

$$"Summer", "SU"$$

$$"fall", "FA"$$

which denote a mapping from "Summer" to "SU" and "fall" to "FA". Assuming "%reYear4Digit" stands for four consecutive digits, a full rule could then be

$$EXTRACTION = "\%reSeason\%reYear4Digit",$$

$$NORMALIZATION = group(2){-}normSeason(group(1))$$

$group(x)$ denotes the expression from the xth regular expression resource in the extraction rule. An example of an extraction and normalization of this rule would be "Summer 2011" is extracted and normalized to "2011-SU" in line with the timex2 and timex3 guidelines.

HeidelTime also supports "negative" rules. These rules are identical to others, except the normalization value is simply "REMOVE." This signals to HeidelTime to surpress any positive matches which overlap with an expression captured by the negative rule. An example of how such a rule would be used is to stop "2000 people" from being extracted as the year 2000.

HeidelTime supports underspecified values during the extraction step. An example of an underspecified value is "UNDEF-July." In this case, in a post processing step, HeidelTime uses the last specified time or the document creation time to fill in the missing values. The choice to use the document creation time is specified when running the program. For news articles, using the DCT greatly improves results, but for historical articles, DCT are usually irrelevant.

Chancellorsville		
	Default HeidelTime	Augmented HeidelTime
Precision	0.892857143	0.980392157
Recall	0.980392157	0.980392157
F-score	0.934579439	0.980392157

TABLE I
OUR RULE ADDITIONS IMPROVED PRECISION BY 0.09 WHILE LEAVING RECALL UNCHANGED, RESULTING IN A SIGNIFICANT F-SCORE INCREASE OF 0.05.

Fredericksburg		
	Default HeidelTime	Augmented HeidelTime
Precision	0.933333333	0.948717949
Recall	0.810810811	1.000000000
F-score	0.867768595	0.973684211

TABLE II
IN THIS ARTICLE OUR RULE ADDITIONS RESULTED IN PERFECT RECALL AND A SLIGHT IMPROVEMENT TO PRECISION, RESULTING IN A SIGNIFICANT F-SCORE INCREASE OF 0.11.

B. Improving HeidelTime

We tested HeidelTime in its publicly-available form on our article pool. In our initial tests, we checked the HeidelTime results of two articles, the battle of Chancellorsville and the Battle of Fredericksburg. Our trial runs demonstrated a few shortcomings of the system as-is. It missed some reasonably common time formats (such as ”830 a.m.”), and sometimes failed to extract durations. More significantly, especially for historical articles detailing troop movements, much like Chasin's approach, it mistakenly extracts ”march” when it does not refer to the month, which throws off the accuracy of the values for subsequent underspecified expressions. There were various other minor errors as well.

HeidelTime keeps its code separate from the rules and resources it uses, so additions to the system are easy and involve no compilation. We began by adding a few new rules to the system, to cover for the most obvious shortcomings during our trial run. We then ran a preliminary comparison of the original system and our augmented version on two of our articles. The results (displayed in Tables I and II) show that our augmented version drastically improves both precision and recall.

These results were exciting, however, we wished to make sure that our new rules weren't overly specific to our domain, and we wanted a more rigorous testing environment. Thankfully, the Ruprecht-Karl University Heidelberg Database Systems Research Group's website, maintained by Gertz, contains testing and evaluation scripts for HeidelTime based on a number of corpora, including the TempEval-2 dataset, Timebank 1.2, and the WikiWars corpus.

The TempEval-2 English dataset is actually based on the Timebank 1.2 dataset, and is an order of magnitude smaller. Thus Timebank 1.2 results can provide a more robust evaluation of our changes.

The WikiWars corpus[10] is a collection of 22 war articles from Wikipedia, with temporal expressions annotated with timex2.[3] Both Timebank and the TempEval-2 dataset are comprised of newswire articles. Historical articles are significantly different in style and presentation, and improvements on the results run on a corpus comprised solely of such articles is more likely to translate into improvements in extraction for our Wikipedia articles than improvements specific to newswire articles.

The evaluation scripts[4] provide precision, recall, and F-score results for five categories: Extraction (lenient), Extraction (strict), Normalization (value attribute), Extraction and Normalization (lenient and value attribute), and Extraction and Normalization (strict and value attribute). Strict extraction means for an expression extraction to be a true positive the expression must exactly match that of the gold standard. Lenient extraction relaxes this constraint to be that the extracted expression must overlap with the gold standard expression. The Normalization category requires the value of the extracted expression to exactly match that of the gold standard, and the combined categories require the extracted expression to meet the criteria of both of the component categories. The scripts also provide an easy to analyze document detailing each expression, and its value for both the gold standard and the results of the HeidelTime extraction. This allows us to easily analyze what changes improve the system the most.

Thus we began the iterative process of adding, removing, and modifying rules, expression resources, and normalization resources, and comparing the results on the WikiWars dataset. We continued this process for over 50 iterations. Our results can be seen in Table **??**. The value attribute recall and precision remained largely the same, with massive gains in all other areas. Recall improved the most, with an 8.1% improvement in lenient extent, and a 9.1% improvement in strict extent. However, all other values improved by at least 3%.

These were quite substantial results, but we wanted to make sure our gains were not because of specializing HeidelTime specifically for historical wikipedia articles, at the cost of precision and recall for other datasets. In order to guard against this, we tested our adjusted HeidelTime on the TimeBank 1.2 dataset. Our results, though not showing as dramatic of gains as on the WikiWars dataset, nevertheless show no decrease in accuracy or precision in any category, and, indeed show modest recall gains of over a full percent in both the lenient and strict extents. This gives us some confidence that our modifications are not over specialized.

Finally, we tested our improved HeidelTime on our pool of Wikipedia articles, taking a random sample of 200 sentences. The results will appear in a table in the final version of this paper.

C. Further Improvements

There are plenty of patterns we could still add to HeidelTime to further increase precision and accuracy, and continuing our iterative improvement process would likely yield

[3]Available for download free of charge from http://www.timexportal.info/
[4]Available on http://dbs.ifi.uni-heidelberg.de/

WikiWars				
		Default HeidelTime	Augmented HeidelTime	Improvement
Lenient Extent	Precision	0.940 = 94.0%	0.974 = 97.4%	+0.034 = +3.4%
	Recall	0.821 = 82.1%	0.902 = 90.2%	+0.081 = +8.1%
	F-score	0.877 = 87.7%	0.937 = 93.7%	+0.060 = +6.0%
Strict Extent	Precision	0.851 = 85.1%	0.901 = 90.1%	+0.050 = +5.0%
	Recall	0.743 = 74.3%	0.834 = 83.4%	+0.091 = +9.1%
	F-score	0.793 = 79.3%	0.866 = 86.6%	+0.073 = +7.3%
Value	Precision	0.896 = 89.6%	0.896 = 89.6%	+0.000 = +0.0%
	Recall	0.900 = 90.0%	0.902 = 90.2%	+0.002 = +0.2%
	F-score	0.898 = 89.8%	0.899 = 89.9%	+0.001 = +0.1%
Lenient Extent + Value	Precision	0.842 = 84.2%	0.872 = 87.2%	+0.030 = +3.0%
	Recall	0.736 = 73.6%	0.808 = 80.8%	+0.072 = +7.2%
	F-score	0.785 = 78.5%	0.839 = 83.9%	+0.054 = +5.4%
Strict Extent + Value	Precision	0.787 = 78.7%	0.826 = 82.6%	+0.039 = +3.9%
	Recall	0.688 = 68.8%	0.764 = 76.4%	+0.076 = +7.6%
	F-score	0.734 = 73.4%	0.794 = 79.4%	+0.060 = +6.0%

TABLE III
A COMPARISON OF THE DEFAULT HEIDELTIME AND OUR MODIFIED HEIDELTIME ON THE WIKIWARS DATASET. OUR CHANGES LEFT THE PRECISION AND RECALL OF THE VALUE ATTRIBUTE NEARLY UNTOUCHED, WHILE DRASTICALLY IMPROVING PRECISION AND RECALL IN ALL OTHER AREAS.

TimeBank 1.2				
		Default HeidelTime	Augmented HeidelTime	Improvement
Lenient Extent	Precision	0.905 = 90.5%	0.907 = 90.7%	+0.002 = +0.2%
	Recall	0.914 = 91.4%	0.928 = 92.8%	+0.014 = +1.4%
	F-score	0.909 = 90.9%	0.917 = 91.7%	+0.008 = +0.8%
Strict Extent	Precision	0.835 = 83.5%	0.835 = 83.5%	+0.000 = +0.0%
	Recall	0.843 = 84.3%	0.854 = 85.4%	+0.011 = +1.1%
	F-score	0.839 = 83.9%	0.844 = 84.4%	+0.005 = +0.5%
Value	Precision	0.862 = 86.2%	0.862 = 86.2%	+0.000 = +0.0%
	Recall	0.862 = 86.2%	0.862 = 86.2%	+0.000 = +0.0%
	F-score	0.862 = 86.2%	0.862 = 86.2%	+0.000 = +0.0%
Lenient Extent + Value	Precision	0.780 = 78.0%	0.782 = 78.2%	+0.002 = +0.2%
	Recall	0.788 = 78.8%	0.800 = 80.0%	+0.012 = +1.2%
	F-score	0.784 = 78.4%	0.791 = 79.1%	+0.07 = +0.7%
Strict Extent + Value	Precision	0.732 = 73.2%	0.732 = 73.2%	+0.000 = +0.0%
	Recall	0.740 = 74.0%	0.747 = 74.7%	+0.007 = +0.7%
	F-score	0.736 = 73.6%	0.739 = 73.9%	+0.003 = +0.3%

TABLE IV
A COMPARISON OF THE DEFAULT HEIDELTIME AND OUR MODIFIED HEIDELTIME ON THE TIMEBANK 1.2 DATASET. ONCE AGAIN OUR CHANGES LEFT THE PRECISION AND RECALL OF THE VALUE ATTRIBUTE NEARLY UNTOUCHED, THIS TIME HAVING MODEST PRECISION GAINS AND DECENT RECALL IMPROVEMENTS IN ALL OTHER AREAS.

increasingly accurate results. One change that would immediately increase value precision in the WikiWars dataset is full support for years in BC. We've add the extraction patterns, however, years BC are currently ignored for the purposes of post-processing underspecified values.

However, there are two large areas currently keeping precision and recall. First is inconsistent gold standard labeling. For example, in the WikiWars dataset, ”the same time” is labelled in every occurrence as a temporal expression. In the TimeBank 1.2 dataset, on the other hand, although ”the same time” occurs many times in similar contexts to the appearances in the WikiWars articles, it is never annotated as a temporal expression. Currently we simply disregard this expression, but adding it in would instantly increase extraction recall on the WikiWars dataset a non-negligible amount, while simultaneously decreasing extraction precision in the TimeBank 1.2 dataset.

The other, larger area, can be improved without modifying datasets. Currently the underspecified value post-processing step fills in the underspecified value with data from the last fully specified temporal expression. This causes errors, such as normalizing an expression of ”January” following ”December 2008” to ”2008-01”, when it should be ”2009-01”. Such errors are then propagated until the next temporal expression that specifies the exact year. Inaccurate post-processing is what leads to the greatest number of errors in the value attribute. A slightly more sophisticated heuristic may be able to drastically improve value recall and precision, and a significantly more sophisticated system, perhaps deriving temporal relations, or simply using lexicographical clues could perhaps increase value precision and recall to near-perfect levels.

V. EVENT EXTRACTION

We extract events from the Wikipedia articles by using Evita, developed by Pustejovsky et al. as an event recognizer for question/answer systems. The events extracted by

Evita are mostly action verbs, but alse include some nouns and adjectives that indicate events occurred. Evita, part of the TARSQI toolkit[5], annotates events with polarity, aspect, modality, tense, non-finite morphology, class, which we can use to filter events that are unlikely to be temporally located. We used Evita off-the-shelf; for more information, read the original paper introducing Evita[11].

VI. Temporal Relation Extraction

In order to order events relative to each other, we use a Markov Logic Network to generate the most likely set of temporal relations. Markov Logic Networks, introduced by Domingos et al. [12] are an extension of first order logic with probabilities other than one (true) or zero (false). Formulas hold probabilistically, so its simple to encode soft rules and probabilistic inference. The example given in [5] of a soft formula is

$$futureTense(e) \implies !beforeDCT(e),$$

that an event in the future tense is likely, but not certain, to be temporally located after the document creation time.

The de facto standard for temporally annotated corpora is the TimeBank 1.2 corpus (also used in our temporal expression extraction tests). The TimeBank corpus consists of about 300 newswire articles human-annotated with temporal events and expression and relations between them, given in Allen interval algebra notation. Allen notation consists of 13 qualitative relations between two intervals, which are depicted in Figure 1. As mentioned earlier, the TempEval tasks used a very small subset of these relations, and most temporal extraction research has avoided using the full set of relations. Ideally, our Markov Logic Network should only need to generate one type of relation. Thankfully, as noted by [1], if we treat every temporal event as an interval with a start and end point, then all thirteen relations can be represented by inequalities between the start and end points of the two intervals. This means we can set out to only find $after(a, b)$, which we take to mean a is after or at the same time as b. We can encode equality between two time points a and b as

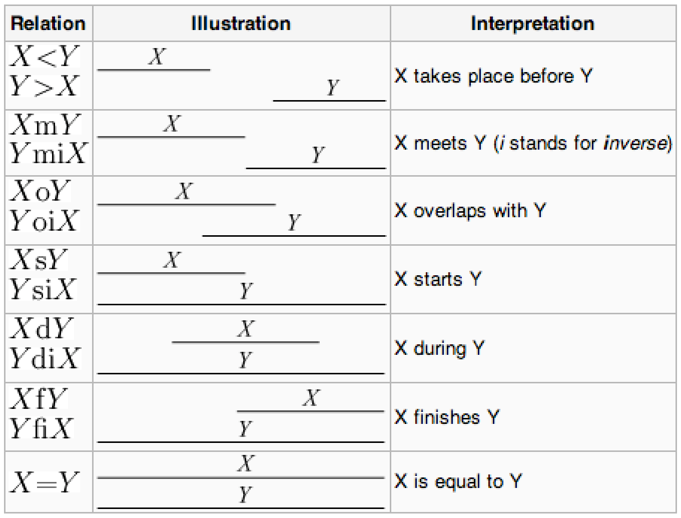
$$after(a, b) \text{ AND } after(b, a).$$

For an example of an Allen relation encoded with these inequalities, consider the two intervals $A = [a_0, a_1]$ and $B = [b_0, b_1]$, where A finishes B. This is equivalent to:

$$after(a_0, b_0) \text{ AND } after(b_1, a_1) \text{ AND } after(a_1, b_1)$$

Similar conversions can be created for all 13 Allen relations. Using these conversions, we can use TimeBank for our training data.

The atoms for our Markov Logic Network are the events from Evita and the time expressions from HeidelTime. There are two main categories of features of our Markov Logic Network. The first is an $apriori\_after$ relation among the atoms from HeidelTime. Since HeidelTime exactly temporally



Relation	Illustration	Interpretation
$X<Y$ $Y>X$	X Y	X takes place before Y
$X\text{m}Y$ $Y\text{mi}X$	X Y	X meets Y (*i* stands for *inverse*)
$X\text{o}Y$ $Y\text{oi}X$	X Y	X overlaps with Y
$X\text{s}Y$ $Y\text{si}X$	X Y	X starts Y
$X\text{d}Y$ $Y\text{di}X$	X Y	X during Y
$X\text{f}Y$ $Y\text{fi}X$	X Y	X finishes Y
$X=Y$	X Y	X is equal to Y

Fig. 1. An illustration of the 13 Allen interval relations. All of the relations depicted except equality have inverses.

locates time expressions, we can generate after relations such as January 1952 is after December 1951 before running the MLN. We code the hard rule

$$apriori\_after(x, y) \implies after(x, y),$$

to ensure that the world outputted by our MLN still contains these relations.

The other features are generated by the Stanford Parser [13]. The relevant soft formula we encode are of the form

$$dep(x, y) \implies after(point(x), point(y)),$$

following [1], where this is a second-order formula, where $point()$ is either the start or endpoint and *dep* stands for any one of the 50+ Stanford dependency. Stanford dependencies are relations between pairs of words in a sentence and are generated by the Stanford parser [14]. Examples of Stanford dependencies are dobj (direct object) and prep_in (related by the preposition ”in”).

We run the Stanford parser on the TimeBank training set, in order to learn the weights on these formulas. The higher the learned weight, the more likely it is that $after(point(x), point(y))$ is a relation in the final world outputted by an MLN. A negative weight corresponds to a formula that is actually likely to not hold. We only consider the dependencies between events, or between temporal expressions, or between one of each.

We also incorporate a global weighted formula softly enforcing transitivity:

$$after(a, b) \text{ AND } after(b, c) \implies after(a, c).$$

This constraint is soft, because there are inconsistencies in our training data that would break this constraint. This is an unfortunate limitation, but one to be expected of a large human-annotated corpus. Like Weld [1], we use a positive prior for this transitivity rule, mostly because the gold dataset

[5]Download details available at http://www.timeml.org/site/tarsqi/toolkit/download.html

often lack transitively closed relations that should be included for complete information.

Unlike [1], we are not using a statistical relational learner to improve results; as we rely on our a priori after relations to pick up the slack. However, an intended extension for future work is a feature relating main verbal events of adjacent sentences. This will allow for more relations between sentences, leading to higher recall and ultimately a more accurate timeline.

Our results in using the MLN to generate temporal expressions will be available in the final draft of this paper.

VII. Synthesis

At this point we have two separate systems, a state-of-the-art temporal expression extractor improving upon HeidelTime, and a temporal relation extractor, following the basic implementation of the TIE system. We still need to integrate the results of these two mostly separate systems, to generate accurate timelines for Wikipedia articles. We show the results of three different approaches.

A. Naive Approach

We first create a baseline by assigning events in a sentence the tightest time-bounds of any date or time expression extracted by HeidelTime in the same sentence. Any events with no bounds after this process acquire bounds through a simple heuristic as follows: determined lower bounds (start points) are propagated forwards, upper bounds (end points) are propagated backwards. If there are still undefined bounds, they are filled in with the closest bound of their type in either direction. Events with identical start bounds are assumed to occur in the order in which they appear in the article. This fully specifies all bounds in an article, so long as at least one upper bound and one lower is specified by a temporal expression. Since each temporal expression has a start and endpoint, this means all bounds are specified so long as there is at least one temporal expression in the entire article. This approach is near identical to the one taken by Chasin.[2]

B. Basic Improvements

Our second approach uses some more data and basic heuristics to improve accuracy. Events are only given the time bounds of temporal expressions if there is a Stanford dependency between the two. If there is a dependency between an event and a duration expression, that event is determined to have taken as long as the expression in question. Otherwise temporal bounds are propagated as in the naive approach.

C. Incorporating the MLN

Our final and most sophisticated approach is largely similar to the last approach. However, instead of propagating bounds based on proximity in the original, bounds are propagated using the after relations generated by the Markov Logic Network.

D. Results

Our results can be seen in a table in the next draft of this paper.

VIII. Future Work

There are several directions to take this work in the future for whoever decides to extend this work. Feature selection for the Markov Logic Network could lead to massive improvements in temporal relation extraction. A simple filter of events unlikely to be temporally relevant could decrease the problem size and difficulty. One could attempt to use fuzzy sets or incorporate explicit uncertainty scores in order to get more accurate timelines. Fuzzy sets, or uncertainty is easily represented visually on a timeline by use of a gradient.

The most useful extension would be the creation of a gold standard annotated corpus based on historical articles. The training data for the Markov Logic Network consisted solely of newswire articles, which are of a decidedly different nature than a historical article, and a more domain-specific corpus would be incredibly useful. WikiWars, though quite useful for improving temporal expression extraction, lacks temporal relation annotation, and thus cannot currently be used to train our MLN. Also, any corpus creation where the articles are annotated with explicit time intervals would open large avenues of research. One such avenue would be trying to learn how long certain events are likely to take using a machine learning approach, and then using this information to estimate the time interval durations in the timeline more accurately.

As mentioned in the temporal expression section of this paper, improving HeidelTime is an area with several low-hanging fruit. More accurate temporal expression extraction will directly lead to more accurate timeline generation. This project's focus was mostly on improving HeidelTime, with some effort going into creating a working Markov Logic Network. Thus the synthesis section represents early first efforts at creating good timelines from temporal expressions and relations, and it is likely there are ways of improving the timelines independently of improving expression or relation extraction.

IX. Conclusion

We have presented significant improvements to HeidelTime, creating what is currently the most accurate system for the extraction of temporal expressions from historical articles. We also reimplemented an MLN, most following in the footsteps of TIE, for temporal relation extraction, and present decent results. Finally, we presented early attempts to combine these two systems in order to generate temporal bounds for all events in a document, and gave suggestions for future improvements.

References

[1] X. Ling and D. S. Weld, "Temporal information extraction," *AAAI*, 2010.

[2] R. Chasin, D. Woodward, and J. Kalita, "Extracting and displaying temporal entities from historical articles," *NCMT*, 2011.

[3] M. Verhagen, R. J. Gaizauskas, F. Schilder, M. Hepple, J. Moszkowicz, and J. Pustejovsky, "The tempeval challenge: identifying temporal relations in text." *Language Resources and Evaluation*, pp. 161–179, 2009.

[4] I. Mani, “Recent developments in temporal information extraction,” *RANLP 2003*, 2003.

[5] K. Yoshikawa, S. Riedel, M. Asahara, and Y. Matsumoto, “Jointly identifying temporal relations with markov logic,” in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ser. ACL ’09. Stroudsburg, PA, USA: Association for Computational Linguistics, 2009, pp. 405–413. [Online]. Available: http://portal.acm.org/citation.cfm?id=1687878.1687936

[6] J. Allen, “Maintaining knowledge about temporal intervals,” *C. ACM*, 1983.

[7] J. Giles, “Internet encyclopedias go head to head,” *Nature*, 2005.

[8] S. Schockaert, M. De Cock, and E. Kerre, “Reasoning about fuzzy temporal information from the web: towards retrieval of historical events,” *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 14, pp. 869–886, 2010, 10.1007/s00500-009-0471-8. [Online]. Available: http://dx.doi.org/10.1007/s00500-009-0471-8

[9] J. Strötgen and M. Gertz, “Heideltime: High quality rule-based extraction and normalization of temporal expressions,” in *Proceedings of the 5th International Workshop on Semantic Evaluation*, ser. SemEval ’10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pp. 321–324. [Online]. Available: http://portal.acm.org/citation.cfm?id=1859664.1859735

[10] P. Mazur and R. Dale, “Wikiwars: a new corpus for research on temporal expressions,” in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, ser. EMNLP ’10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pp. 913–922. [Online]. Available: http://portal.acm.org/citation.cfm?id=1870658.1870747

[11] R. Saurí, R. Knippen, M. Verhagen, and J. Pustejovsky, “Evita: A robust event recognizer for qa systems,” *Proceedings of HLT/EMNLP 2005*, pp. 700–707, 2005.

[12] M. Richardson and P. Domingos, “Markov logic networks,” *Machine Learning*, vol. 62, pp. 107–136, 2006, 10.1007/s10994-006-5833-1. [Online]. Available: http://dx.doi.org/10.1007/s10994-006-5833-1

[13] D. Klein and C. D. Manning, “Fast exact inference with a factored model for natural language parsing,” *Advances in Neural Information Processing Systems*, vol. 15, pp. 3–10, 2003.

[14] M.-C. de Marneffe, B. MacCartney, and C. D. Manning, “Generating typed dependency parses from phrase structure parses,” *LREC 2006*, 2006.

Experiments in Creating Bilingual Dictionaries using Existing Dictionaries

Richard Seliga
Department of Computer Science
University of Colorado at Colorado Springs
Colorado Springs CO 80918

***Abstract*—The purpose of this project is to create a multilingual dictionary that is available to people on the web and can translate from any language in our database to another. There are around 6000 languages in the world. The biggest translation tool out there today is Google translate which supports only around 60 languages. Another problem with translating is that most of the dictionaries on the internet are between two common languages like English, French or German. But what if you would like to translate something from Slovak to Assamese? This project will create a dictionary between these languages using the clique (graph) theory, triangulation and using variables like language family to get good translations.**

I. Introduction

The plan for this project is to create a multilingual dictionary focusing on the Slavic and Indic languages, since dictionaries between these two families of languages are rare at best. We will be doing this by using existing bilingual dictionaries and using the worlds' most common languages like English or German as the common ground. The motivation for this project is the fact that there are not many dictionaries between languages like Slovak and Assamese but there are dictionaries between English and Assamese or English and Slovak. This project will create a dictionary where you can translate from any language in our database to another.

A. Building Database of Bilingual Dictionaries

One of the issues we run into in the beginning is the fact that there are not many dictionaries available on the internet. We decide to build our database of dictionaries by querying websites that offer bilingual dictionaries. Even though this process is slow it is currently the only option. We also used Wiktionary as a resource. The only issue with using Wiktionary is that even though its very detailed the amount of information for smaller languages is insufficient but its still more then we get from querying the previous dictionary. Wiktionary is a good resource because not only does it contain language translation but it includes the senses which are very important in creating our dictionary graph and getting good results.

B. Creating a Dictionary between two Languages

We initially have two or more bilingual dictionaries which are the Slovak-English dictionary and the Czech-English dictionary. These dictionaries were created by querying the same site with the same 58000 English words. The information stored includes the part of speech and the translation. The problem we are foreseeing is the fact that this dictionary does not show the sense of the translating word which is a vital part in creating these dictionaries. We solve this problem by querying Wiktionary.org and getting the senses for some of these translations.

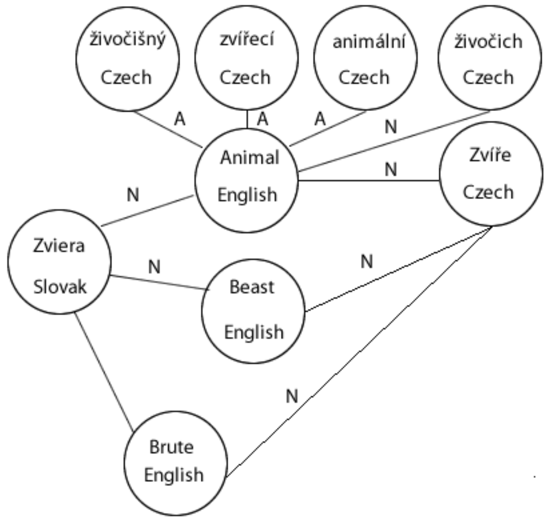


Fig. 1. In this figure we show a very simple example of translating between Slovak and Czech using one common language and parts of speech as lines in between these nodes. N = Noun. A = Adjective

II. The Process using Three Languages

In this graph we see that the word *zviera* in Slovak has three translation in English and they are all nouns, however the word *animal* in English has more translations in Czech which include adjectives as well as nouns. We ignore the adjectives since the word in Slovak is a noun. The English word animal has some other translations including *zvíře* and *živočich.* Beast and Brute also translate to *zvíře* and since this is the most active part of the graph we see that we can translate *zviera* to *zvíře*. This does not seem too bad however in this graph we are using simple words that do not have more then one sense. So for example using the word heart can mean two or more things in both Slovak or English and that is the problem we

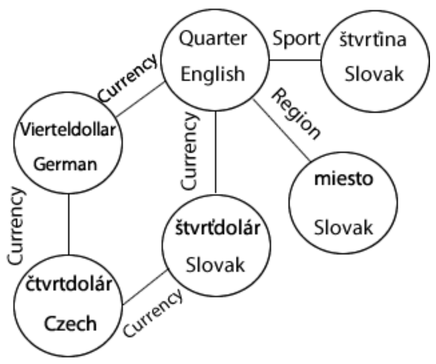


Fig. 2. In this figure we create a clique. This clique includes the nodes *quarter* in English, *vierteldollar* in German, *štvrťdolár* in Slovak and *čtvrtdolár* in Czech. This is one of the simpler cliques we will use to find translations.

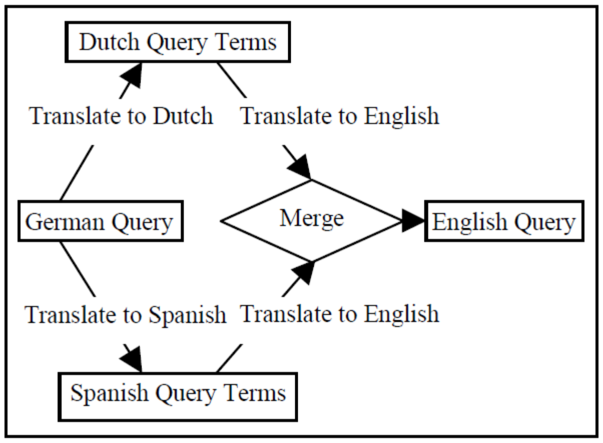


Fig. 3. In this figure we see the research done by Tim Gollins and Mark Sanderson from University Of Sheffield. This was the earliest method of creating dictionaries. However this was inaccurate.

are trying to solve. In this graph it really is a ideal situation as three English words translate to the same word in Czech but this will not always be the case. During this project we will start using 2 bilingual dictionaries and see if the results improve as we enter in more variables which will include more languages and the words senses.

III. The Process using More than Three Languages

When you do not have an ideal situation like in the Figure 1 you will most likely have to use more then one language to get the correct translation. With this we will turn to clique theory. A clique is a set of three or more nodes connected to one another. In our case the words will be nodes which will be connected to each other using translation, sense and part of speech. From this we will create a similar graph to the one in Figure 2.

In the graph above we are looking for a translation from English to Czech. The word *quarter* however has a lot of meanings in English and could be a problem to translate correctly. This is where we cannot rely on part of speech since the meaning of *quarter* could be *miesto* or *štvrťdolár* which are both nouns. So this is where we could use multiple languages to pinpoint the correct translation. Since we have the senses, this is also not a complicated problem to pin point the translation.

IV. Complication of lack of data

The complication becomes when we lack the senses and we are translating a word like quarter. Some of the possible solutions to this problem could include:

- Creating word graphs using language families and sub-families
  - Using the Slovak language as an example we know that it is part of the Slavic family of languages and part of the west Slavic subfamily.
- Looking and the age of the languages is also important, if a language is reasonably young we know that it is more influenced by the more popular languages like English.
  - For example current day Poland, Czech and Slovak republic at one point used the same language. Slovak being the youngest of the three would be more influenced by the worlds languages.

V. The Database and The Translation Graph

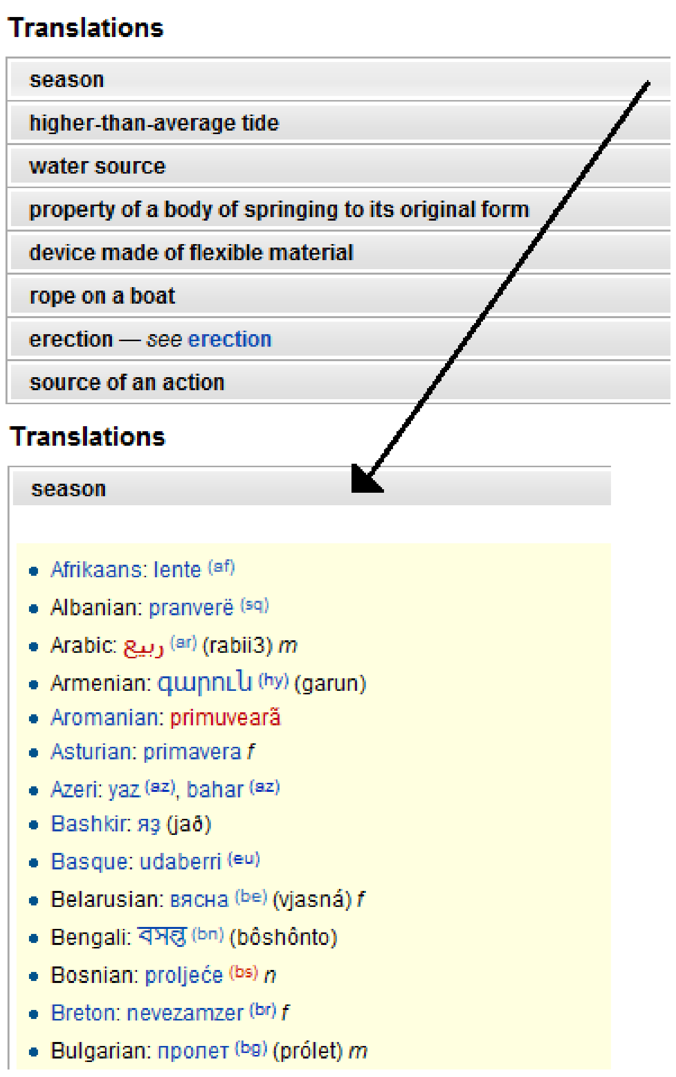
Since the number of nodes is over 300,000 using only 4 languages we needed to find an effective and fast way to store the data. We have 4 tables in our database including a part of speech table, language table, word table and dictionary table. The dictionary table includes 2 words from the word table and an edge which is the sense. This is shown in figure 4. The language table includes the ID and text field which represents the language. The same go's with the part of speech table. The word table are included the indexes of the word since this is how we built our graph which includes 300,000 nodes, 4 bilingual dictionaries and 4 languages. Previous work puts senses and converts them to ID's which results in sense ID inflation however we use the senses from Wiktionary.org and store them as text.

A. Storing the Senses

We also query the 48,000 English words and store the multiple senses of the English word into a database and the language ID's that include a translation. What we mean by this is that Wiktionary.org does not have all the translations but it still has quite a bit. This can be seen in figure 4.

VI. Graph Theory

The way we will find translations is using graph theory which in mathematical and computer science fields is the study of graphs. A graph in our case will be a collection of nodes connected to each other by edges/senses. Graphing these nodes will create a huge graph with over 300,000 nodes using only 4



Translations

season

higher-than-average tide

water source

property of a body of springing to its original form

device made of flexible material

rope on a boat

erection — *see* erection

source of an action

Translations

season

- Afrikaans: lente (af)
- Albanian: pranverë (sq)
- Arabic: ربيع (ar) (rabii3) *m*
- Armenian: գարուն (hy) (garun)
- Aromanian: primuveară
- Asturian: primavera *f*
- Azeri: yaz (az), bahar (az)
- Bashkir: яҙ (jað)
- Basque: udaberri (eu)
- Belarusian: вясна (be) (vjasná) *f*
- Bengali: বসন্ত (bn) (bôshônto)
- Bosnian: proljeće (bs) *n*
- Breton: nevezamzer (br) *f*
- Bulgarian: пролет (bg) (prólet) *m*

Fig. 4. This figure shows the Senses for the word Spring. There are multiple translations for this words which you get by opening one of these tabs up. This results in numerous translations in different languages.

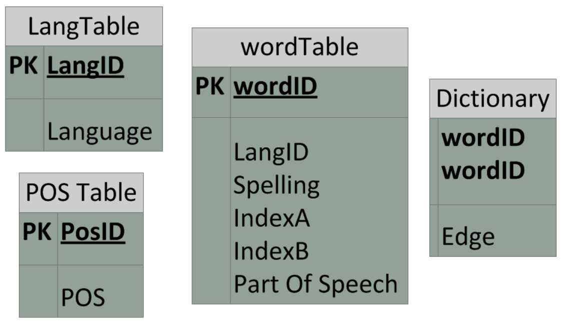


Fig. 5. In this figure we have the database diagram of the tables

languages. Previous papers discussed completing graphs into cliques to create translations of high accuracy. A clique is a subgraph where each node has an edge between every other node. For example if you have node 1,2 and 3 then node 1,2 have to be both connected to 3 and 2,3 have to be both connected to 1 and so forth. This will form a clique. In relation to our problem we can see that completing these kind of graphs is very complicated when you lack data. The problem will be finding the probability if we can complete the clique in a given subgraph which will give us the correct translation most of the time.

VII. Previous Research

This is a relatively new subject in the research world as the earliest paper about this subject was in 2001 by Tim Gollins and Mark Sanderson. There research provided us with a new way to use online resources to create dictionaries between languages. In 2010 however the University of Washington came up with an idea of using graph theory to come up with results that rival human translating. We believe combining these two methods and making improvements will give us even better results and create dictionaries between the worlds rarest languages.

A. *Triangulated Translation*

Figure 3 shows how the University of Sheffield research created dictionaries. However just using triangulation does not produce the best results but at the time this was the best method. [3] This was the very beginning of using other dictionaries to create other dictionaries. However the problem using this technique was the fact that for example the word spring has a lot of different meanings, it can mean the season or a water source. This however does not translate the same way to the other languages and this is where a lot of inaccuracies happen and that's why it was not as effective as the research done by University of Washington.

B. *Probabilistic Inference*

University of Washington has just recently created algorithms who use senses and probabilities to get translations. Since the point is to create a cliques from these nodes, the research is not really about finding translations but about completing the subgraphs into cliques to find the probability of the translation. Their research shows that if you can find these cliques there is a very high chance that the translation is correct. The edges in these graphs use the senses which they have from their dictionaries. Since the amount of resources they have exceeds any previous research done it will be tough to test the same algorithms. To solve this we will have to change these algorithms to better suit a smaller dataset. The dataset of University of Washington includes over 600 dictionaries, 60,000,000 edges and 10,000,000 nodes. They use this data to build a giant graph on which they run their analysis of probability. Algorithm 1 shows the algorithm to obtain the probability.[1]

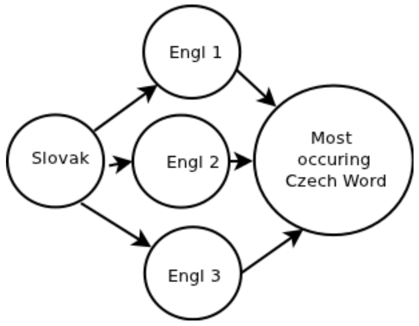


Fig. 6. This figure shows the Senses for the word Spring. There are multiple translations for this words which you get by opening one of these tabs up. This results in numerous translations in different languages.

Algorithm 1 S.P.$(G, v_1^*, v_2^*, \mathcal{A})$

```
1: parameters $N_G$: no. of graph samples, $N_R$: no. of random walks, $p_e$: prob. of sampling an edge
2: create $N_G$ versions of $G$ by sampling each edge independently with probability $p_e$
3: for all $i = 1..N_G$ do
4:     for all vertices $v$ : $rp[v][i] = 0$
5:     perform $N_R$ random walks starting at $v_1^*$ (or $v_2^*$) and pruning any walk that enters (or exits) an ambiguity set in $\mathcal{A}$ twice. All walks that connect to $v_2^*$ (or $v_1^*$) form a translation circuit.
6:     for all vertices $v$ do
7:         if($v$ is on a translation circuit) $rp[v][i] = 1$
8: return $\frac{\sum_i rp[v][i]}{N_G}$ as the prob. that $v$ is a translation
```

VIII. Results

A. One or Two Intermediate Languages Algorithm

In this algorithm we use one intermediate language to translate from language A to language B. Our intermediate language would be language C. To translate from A to B we first translate from A to C and then from B - C. The way we scored the translation from A to B is that we totalled the number of the word with the same spelling at the end of translating from B - C. We can see this in Fig. 6.
For the two intermediate languages we follow the same steps as in the paragraph above, however instead of having one intermediate language we have two. This improved our results by .08. For such a simple algorithm we believe these results are pretty good for something so simple. The languages used for testing were Slovak, Czech, English and German. For the one intermediate language we used Slovak $\rightarrow$ English $\rightarrow$ Czech and for two intermediate languages we used Slovak $\rightarrow$ (English,German) $\rightarrow$ Czech

Results	Accuracy
1 intermediate language	.63
2 intermediate languages	.71

IX. Future Work

Even though the possibility of being able to translate from one language in our database to another is a tall task on its own in the future we would like to make additions to translate documents. Some of the other options after finishing building these dictionaries include creating a multilingual search engine which would accept any language and return results in English.

A. Algorithm 3

This algorithm is more complicated but we believe it would give us substantially better results. This algorithm will use a database of at least ten multilingual dictionaries. This will essentially create a big graph. A small part of this graph can be represented by Fig. 1. What we will do after we generate this graph is that we will take as many paths as possible to our desired language from the original language. This will give us a lot of different paths. What we analyse with this algorithm is these paths. Some of the rules we have generated for these different paths.

1) Consider the path length as a negative effect on the accuracy
2) Consider the branching factor as a negative effect on the accuracy
3) Translating between languages of the same family within the path improves the accuracy.
4) Having the same sense and/or part of speech on the path improves the accuracy.

This is a different approach then the work done at University of Washington however implementing these changes on top of their work could definitely improve already outstanding results.

References

[1] Mausam and S. Soderlang and O. Etzioni and D. S. Weld and K. Reiter and M. Skinner, *Panlingual lexical translation via probabilistic inference*, Essex, UK: Elsevier Science Publishers Ltd., 2010.

[2] Mausam and S. Soderlang and O. Etzioni and D. S. Weld and K. Reiter and M. Skinner and J Bilmes, *Compiling a Massive, Multilingual Dictionary via Probabilistic Inference*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2009.

[3] Tim Gollins and Mark Sanderson. *Improving cross language retrieval with triangulated translation*, New Orleans, Louisiana, United States: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval 2001.

Vaulted Verification: A Scheme for Revocable Face Recognition

Michael Wilber
University of Colorado, Colorado Springs
mwilber@uccs.edu

***Abstract*—As biometric authentication systems become common in everyday use, researchers are beginning to wonder about the ethics of biometric recognition. In particular, this paper outlines the need for a face recognition system that does not compromise the privacy of the subjects being recognized. We present a brief overview of current face verification systems and discuss one such implementation. We also outline several obstacles that must be overcome to protect SVM-based face classifiers. To overcome these obstacles, we present a novel protocol we call "Vaulted Verification" that allows a server to authenticate a client's biometric in a privacy preserving way. Finally, we conclude with a small evaluation of performance, discussion of some security implications, and ideas for future work.**

I. Introduction

FACE recognition systems are ubiquitous. The real-world usefulness of a system that recognizes faces combined with increasing computational power has led to an explosion of research over the past few years. As a result, many different face recognition systems enjoy widespread use in several professional domains such as academia, security, data mining, and information retrieval.

Given their ubiquity, it is important that face recognition systems can both guarantee security and operate without sacrificing individual privacy. This project seeks to build an accurate, secure, and private face verification system. By "accurate", we mean that this system should correctly recognize faces with no false positives. By "secure" and "private", we mean an individual should be able to revoke their identification at any time and an attacker should not be able to reconstruct information about the face if they somehow acquire a copy of the template database. Finally, this system should operate well under the speed and memory constraints imposed by modern hardware.

Unfortunately, there are few existing verification packages available that preserve privacy and security. Corporations are beginning to grapple with the dilemma of ethical face recognition. With increased widespread use comes increased responsibility and increased pressure to preserve customers' privacy. Recent events such as Facebook's controversial widespread deployment of automatic face recognition technology [1] are beginning to raise questions about privacy in customers' minds.

II. Existing face recognition schemes

According to [2], there are three closely related problems in the domain of automated face recognition:

- **Pair matching**. In this scheme, an algorithm decides whether or not two separate photos are of the same individual.
- **Recognition**, also known as identification, closed-set, or "multi-class" recognition. A gallery is constructed containing photos of several different people. The algorithm takes a new photo of a subject and decides who it is, selecting one out of the set of gallery subjects.
- **Verification**, also known as authentication, open-set recognition, or "one-class" recognition, is where the gallery consists of several images of a single subject. The algorithm is given either an image of the subject or an image of an impostor from an unknown universe of impostors. The algorithm then decides whether the probe image is of the subject in the gallery or is an impostor.

We concern ourselves with the last of these problems as it is the issue addressed by most common biometrics systems. Our face verification system is analogous to a fingerprint scanner in the sense that it can be used to allow or deny an individual's access to a resource.

How can one build a face verifier that does not compromise privacy? To answer this question, we will first present an overview of current face verifiers. Then, we will describe the privacy issues in Section III and present the preliminary results of a novel protocol designed to overcome these issues.

Consider the following scenario: User Bob wishes to log in to his bank account using his face. When Bob visits the bank to enroll, his bank takes several pictures of his face. They normalize these pictures and convert them into feature vectors which are stored in the bank's gallery database. To authenticate a potential account holder's photograph (probe), the stored feature vectors are used to train a one-class SVM classifier which then either accepts or rejects the probe.

A. Experimental protocol: Dataset composition

Several commonly used public-accessible datasets exist for the purpose of studying facial recognition problems. In particular, FERET [3] and LFW [2] are among the most recognized, but they answer slightly different questions as FERET is intended for closed-set recognition [3] and LFW is primarily concerned with image pairing [2].

Because our experiments depend on machine learning classifiers that require three or four images to train each subject, we chose the FERET240 set, a subset of FERET that contains the subjects that have at least four images, as described in [4].

FERET240 is intended to test recognition problems rather than verification problems (see Section II), so the following changes were made to the test protocol: First, the test is repeated for each of the 240 subjects. The total scores – true and false positives and rejects – are summed and considered as scores of the overall test. g pictures of the subject (usually three or four depending on test) are considered to be the gallery, and the rest of the subject's pictures are considered the positive-labeled probes. The impostors are all images of the other 239 subjects in the set.

B. Generating the feature vector: GRAB

For every classification, we must convert images into feature vectors. For this project, we will use the GRAB descriptor [4] as it has been shown to yield good accuracy on multi-class recognition problems in the FERET and LFW datasets. GRAB has also been used on open-set recognition problems similar to the ones we face [5].

GRAB is a modification of linear binary patterns (LBP) [4]. LBP works by splitting the image up into 64 sub-windows [6]. A "feature histogram" is computed for each sub-window, where each feature is found by thresholding each pixel's brightness against its eight neighbors, yielding an 8-bit number. This means each pixel may possess one of 256 distinct feature "tags". Each sub-window's feature histogram is then concatenated to form the final feature vector.

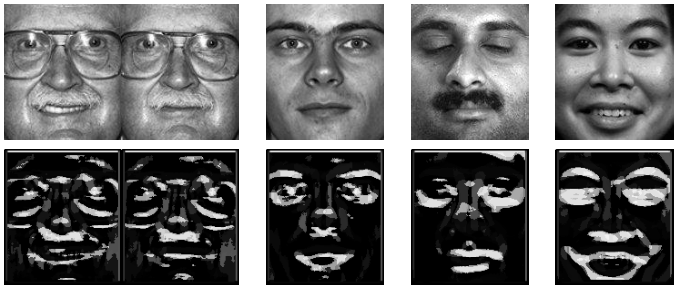


Fig. 1. A demonstration of GRAB neighbor preprocessing before windowed histogram application. The two images on the left are of the same subject and demonstrate intra-class variation; the three images on the right are of different people and show inter-class variation.

GRAB works in much the same way but with some slight differences. First, instead of sampling each pixel's neighbors, GRAB samples different-sized neighborhoods of pixels around the target pixel. This makes GRAB less resistant to noise and resolution changes than LBP [4]. Second, GRAB demands that each pixel's brightness must be at least a given threshold brighter or darker than its neighborhood, whereas LBP only compares the two pixels' values by a simple "greater-than" or "less-than" comparison. This ensures GRAB only finds features important enough to yield high contrast changes.

C. Classification

Unfortunately, GRAB feature vectors may have nonlinear intra-class variations. This means it is often impossible to distinguish between two vectors of the same subject versus two vectors across different subjects with a "nearest neighbor" classifier. To work around this, machine learning techniques such as support vector machines (SVM) are often used because they can better distinguish these relationships [4], [7].

A binary SVM is trained against positive and negative feature vectors. The output is a support vector machine that, when given an unknown feature vector, can distinguish whether it is a member of the positive class or the negative class. Because our problem involves verification rather than recognition, we wish to use an SVM that can distinguish between the intended subject and everyone else. Unfortunately, a binary classifier is unfeasible because we cannot possibly know all of the negative examples – this would require taking pictures of every possible impostor. Thus, we use a one-class SVM, which can distinguish between members and nonmembers of the gallery set.

D. Preliminary evaluation results

We built the above system to establish a good baseline of the kind of results we can expect. This baseline will show us errors in our existing implementation and will reveal how privacy-preserving mechanisms degrade verification scores.

	TAR	FAR
One-class SVM, linear kernel	71.09%	39.38%
Gaussian kernel (γ = 0.5)	78.80%	4.51%
Gaussian (γ = 0.75)	72.38%	3.26%
Gaussian (γ = 1)	65.73%	2.85%
Gaussian (γ = 2)	57.39%	2.41%
Gaussian (γ = 5)	50.96%	2.10%
Open-set SVM (γ=1, η=best rejected score)	80.09%	1.04%
$\eta=\frac{9}{10}$ between rejected and accepted	64.6%	0.017%
η=Optimize recall when precision=.75	81.80%	0.89%

TABLE I
PRELIMINARY RESULTS OF A ONE-CLASS SVM VERIFICATION SCHEME. TAR, FAR = TRUE ACCEPT RATE AND FALSE ACCEPT RATE

In Table II, a "true positive" is defined as correctly accepting an honest subject and a "false positive" is defined as accidentally accepting an impostor. True positive and false positive rates are obtained by taking the corresponding statistic and dividing it by the respective number of classifications that should have been accepted and rejected – for this experiment, 467 classifications should have matched and 251,906 should have been rejected.

At a fundamental level, it is hard to pick the proper parameters for a one-class SVM to achieve a desired level of generality. Should the SVM match against all images? All humans? All humans of a certain ethnicity? Images of a certain subject? Only images of a certain subject with a certain pose? Depending on the problem, all of these may be desired classifiers. Very recent work with "open-set support vector machines" is beginning to emerge as a possible solution. Open-set SVMs build upon one-class SVMs by using "margin morphology" techniques to achieve a desired level of generality by shrinking or expanding the 1-class SVM's decision function to minimize error [5]. This essentially works by training a one-class SVM as usual and then providing canonical negative examples. These do not impact the support vectors or kernel parameters (as they would in a binary SVM);

instead, these canonical negatives provide a reference for the desired generality of the classifier. Three experiments with an open-set SVM classifier are included in Table II.

III. Privacy concerns of SVM-based recognizers

Implementing a face verifier as per the above system may satisfy the "accuracy" requirement in that the system will ideally accept only the subject and reject impostors (please humor us), but this system does not address the "security" or "privacy" requirements. In other words, simply building an SVM-based classifier is not enough to ensure the privacy of the subject being classified. In our case, we want to protect the final classifier itself because if an attacker could obtain the SVM representation, he could conceivably consider local maxima of the kernel function to produce a feature vector that would be accepted by the SVM. Once the attacker has suitable feature vectors, he could conceivably produce an image that has the same GRAB features. Even if this image might not look anything like the original subject, it would still be falsely accepted because it would evaluate to a feature vector that the kernel classified in the same way as the subject.

The literature discusses several ways of "protecting SVM privacy", but none of them are directly suitable to this problem domain.

A. Previous approaches to SVM privacy protection

[8], [9] describe two secure ways of training an SVM where multiple parties have different shards of the training set. Unfortunately, both methods are secure only when there are more than three parties. Also, though these methods protect the training set, they do not address protecting the SVM itself after training; it is assumed to be securely stored by a third party – this is an assumption we cannot make.

[10] describes a way of post-processing a trained SVM to protect privacy of the support vectors, but this approach only works on Gaussian kernels because it discards terms of the Gaussian function. In a sense, the final SVM produced by this method trades privacy for accuracy. The final SVM still leaks information about what types of vectors it classifies.

All three methods focused on problems such as medical classifiers where the goal was to release the final classifier while still protecting the privacy of individual patients' support vectors. The final SVM was left only slightly altered [10] or completely unprotected [8], [9].

B. Fuzzy Vaults: Another approach to biometric privacy

Other previous work involving privacy-preserving biometric matching includes the use of "fuzzy vaults", where a secret can be 'locked' in the coefficients of a polynomial and can only be 'unlocked' if the subject provides a certain number of biometric features. In theory, privacy is preserved by including several 'chaff' features. The security of the scheme rests on an attacker's inability to guess the real features among the chaff.

Unfortunately, the literature contains both practical security problems of fuzzy vaults [11] (outlined in Section III-C) and information-theoretic weaknesses in certain implementations of fuzzy-vaults [12], [13]. Our protocol includes many ideas from fuzzy vaults but used in unconventional ways. As such, we try to sidestep many of these issues.

Fuzzy vaults [14], an improvement of "fuzzy commitment" [15], are cryptographic schemes that allow a user to conceal a secret S using a set of elements A as a key. S can be decoded by obtaining a sufficient number of elements of A. These vaults are "fuzzy" because not all elements of A are required to release S, the elements of A used to lock and unlock S can differ by a small amount, and the elements of A can come in any order. This is useful for biometric systems where noise, reordering, and missing information are obstacles that must be amended.

In brief, fuzzy vaults work by embedding the secret S inside the coefficients of some polynomial F. To lock this secret with set $A = \{A_1, A_2, \ldots, A_k\}$, a list of pairs is built: $(A_1, F(A_1)); (A_1, F(A_2)), \ldots, (A_k, F(A_k))$. Additionally, chaff points are added to this list by concatenating pairs of random numbers $(R_1, R_2), (R_3, R_4), \ldots$ to the list such that $R \cap A = \emptyset$ and $R \cap \{F(A_i) | A_i \in A\} = \emptyset$. The list of pairs is then permuted to remove ordering information.

Because an honest user has a sufficient subset of A, they can easily tell which pairs are chaff and which pairs correspond to $(A_i, F(A_i))$. From this, they can determine the coefficients of F and decode the secret S embedded therein.

C. Shortcomings of Fuzzy Vaults

Unfortunately, using fuzzy vaults alone may not provide adequate security for all applications. [12] explores the insecurity of certain configurations of fuzzy vaults by presenting the feasibility of better-than-brute-force attacks. Further, [11] lists three potential attacks with potentially serious consequences:

- An **attack via record multiplicity** allows an attacker to correlate the subject's records across multiple vaults by comparing the pairs of $(A_i, F(A_i))$ across each vault. Common (or similar) pairs are more likely to be elements of A and pairs that are dissimilar are likely to be chaff. At best, the attacker's search space is greatly reduced. At worst, the attacker can piece together the elements of A and by solving for F, the attacker can trivially decode both secrets.
- A **surreptitious key inversion** attack allows the attacker to recover the elements of A if he knows S. By building a polynomial from the coefficients of S, he can trivially determine which points lie on F and which are chaff. From there, he can recover A. If the elements of A are raw biometrics, the attacker has compromised the subject's privacy.
- An **insidious substitution attack** allows an attacker to construct a fuzzy vault that silently authenticates both him and the subject. This can be done without alerting the subject that anything is wrong. To do this, the attacker edits the stored fuzzy vault and replaces the chaff points $(R_1, R_2), (R_3, R_4), \ldots$ with $(B_1, F(B_1)), (B_2, F(B_2)), \ldots$ for his own set B. This constructs a fuzzy vault where $(B_i, F(B_i))$ look like chaff from the subject's point of view and $(A_i, F(A_i))$

look like chaff to the attacker's point of view. Thus, the vault authenticates both subject and attacker without any warning to the subject. Now the attacker can log in to the subject's account without the subject's or the bank's knowledge.

IV. Evaluation

To find out how well our Vaulted Verification protocol performs, we implemented a preliminary version over the course of the summer.

	TAR	FAR
N=64, Fuzzy search, chaff from random person, required bits = 20	70.491%	6.549%
N=64, Required bits = 35	38.235%	0.508%
N=64, Required bits = 45	36.364%	0.052%
N=64, Required bits = 50	23.529%	0.000%

TABLE II
PRELIMINARY RESULTS OF THE VAULTED VERIFICATION PROTOCOL AS APPLIED TO FACE RECOGNITION. IN THIS TABLE, TAR MEANS TRUE ACCEPT RATE AND FAR MEANS FALSE ACCEPT RATE.

Table II presents preliminary results of several tests at varying levels of sensitivity. Unfortunately, each test is very computationally expensive and thanks to a power outage partway through, these results are representative but incomplete.

Each test follows the experimental protocol established in Section II-A. For these tests, "required bits" depicts how many bits of the challenge are required to authenticate. The total template contained 64 bits. This allowed us to vary the sensitivity.

In the table presented, the highest true accept ratio required 20 bits. This yields only 1,048,576 different possible keys and is feasible for a brute-force search. In the real world, we would require many more than 64 bits for an attacker to guess. The next stage of our work involves trying larger-scale experiments. Other improvements can also be made. For example, several feature vectors of the subject can be used at match time to improve confidence.

These impostor trials assume the attacker successfully subverted SSL encryption along with all keys kept by a client. These tests only test how well the biometric itself protects the template and in a practical sense, a successful authentication requires physical access to the client's device. If the attacker could get this far, they could likely obtain an image of the face and authenticate easily anyway. That said, without working face recognition, we have merely re-implemented two-factor authentication. There are clearly many opportunities for future improvement here.

V. Goals, Status, and Roadmap

This phase of the project is nearing completion. Over the course of the summer, we implemented the vaulted verification protocol on top of the MFFR experiment framework. This allows for easy experimentation and sweeping customizations to the experimental protocol. `mffr` is a modular facial recognition pipeline descended from the "V1" codebase [17], [18]. Our first task was to reproduce the GRAB experiment [4] with the mffr pipeline as a quick sanity check. To do this, we reimplemented GRAB in Python. This created a unified pipeline for the future work of other lab mates. Our implementation of the GRAB feature vectorization was pixel-perfect with `grab-c`, the reference implementation. Though we reimplemented the experiment as described, we achieved inferior (but comparable) results and are working with the author to improve recognition rates. This may yield improvements for other areas of our pipeline as well.

Because mffr was only suited to recognition problems, as the next step, we adapted mffr to handle face verification problems as well. After all, the vaulted verification protocol is a verification problem at heart, not a multiclass recognition problem.

Once this was completed, we ran preliminary experiments of an ordinary GRAB-based face verifier, the results of which are presented in Table II. This provided a baseline of how well a non-privacy-protecting scheme would work.

Date due	Task description	Status
—	Implement GRAB feature vector descriptor in mffr pipeline	**Done**
—	Reproduce GRAB recognition experiment [4] with mffr pipeline (as a sanity check)	**Done** (imperfect)
—	Repurpose mffr pipeline for verification challenges	**Done**
—	Run preliminary experiments, gather baseline with one-class SVM	**Done**
—	Optimize/improve baseline	**Done**
—	Implement Open-set SVM	**Done**
—	Phase 1: Vaulted Verification protocol, first iteration	**Done** (imperfect)
Aug 5	Final presentation and paper	**Done**
Aug 15	Phase 2: Vaulted Verification protocol, refine and harden	To do
Sept 15	Final ICB2012 submission deadline	On track

TABLE III
SCHEDULE AND TARGET DATES

We then implemented the first draft of the vaulted verification protocol, collecting preliminary results as seen in Table II. This was intended to find various ways of defining chaff and their impact on recognition scores.

The final product of this work will be a paper submitted to ICB 2012, the 5th International Conference on Biometrics. The submission deadline is September 15, and we hope to have all work completed by then. This paper will describe the details and implementation of our face verification system. Time permitting, we will also describe possible approaches to fingerprint verification.

Long-term goals of this work include commercialization of a privacy-enhanced face recognition system, its implementation on mobile devices, and searches for further ideas of improving the privacy protection scheme while raising recognition rates.

VI. Conclusion

This paper demonstrated the need for a face recognition system that earns users' trust by allowing faces to be recognized without knowing what those faces look like. To do this, we started out by building a normal, ordinary face verification system by using GRAB features and one-class SVMs. We then implemented the "vaulted verification" protocol as a way of protecting subjects' privacy. We presented preliminary results that demonstrate this protocol's feasibility for face verification and we outlined several possible ideas for improvement.

Acknowledgments

The research reported in this document has been funded partially by NSF grants CNS-0958576 and CNS-0851783.

References

[1] "Facebook 'face recognition' feature draws privacy scrutiny," ser. Bloomberg News. New York Times, 2011.

[2] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," University of Massachusetts, Amherst, Tech. Rep. 07-49, October 2007.

[3] P. Phillips, H. Moon, S. Rizvi, and P. Rauss, "The feret evaluation methodology for face-recognition algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22(10), pp. 1090–1104, 2000.

[4] A. Sapkota, B. Parks, W. Scheirer, and T. Boult, "Face-grab: Face recognition with general region assigned to binary operator," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, june 2010, pp. 82 –89.

[5] Anonymous, "Margin morphology and the open set svm," Currently under review for ICCV, Tech. Rep., 2011.

[6] T. Ahonen, A. Hadid, and M. Pietik?inen, "Face description with local binary patterns: Application to face recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, pp. 2037–2041, 2006.

[7] D. Fradkin and I. Muchnik, "Support vector machines for classification," Series in Discrete Mathematics and Theoretical Computer Science, Tech. Rep., 2006.

[8] H. Yu, X. Jiang, and J. Vaidya, "Privacy-preserving svm using nonlinear kernels on horizontally partitioned data," in *Proceedings of the 2006 ACM symposium on Applied computing*, ser. SAC '06. New York, NY, USA: ACM, 2006, pp. 603–610.

[9] H. Yu, J. Vaidya, and X. Jiang, "Privacy-preserving svm classification on vertically partitioned data," in *Advances in Knowledge Discovery and Data Mining*, ser. Lecture Notes in Computer Science, W.-K. Ng, M. Kitsuregawa, J. Li, and K. Chang, Eds. Springer Berlin / Heidelberg, 2006, vol. 3918, pp. 647–656, 10.1007/11731139_74.

[10] K.-P. Lin and M.-S. Chen, "Releasing the svm classifier with privacy-preservation," in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 899–904.

[11] W. Scheirer and T. Boult, "Cracking fuzzy vaults and biometric encryption," in *Biometrics Symposium, 2007*, sept. 2007, pp. 1 –6.

[12] P. Mihailescu, A. Munk, and B. Tams, "The fuzzy vault for fingerprints is vulnerable to brute force attack." in *BIOSIG*, ser. LNI, A. Brmme, C. Busch, and D. Hhnlein, Eds., vol. 155. GI, 2009, pp. 43–54.

[13] E.-C. Chang, R. Shen, and F. W. Teo, "Finding the original point set hidden among chaff," in *ASIACCS*, 2006, pp. 182–188.

[14] A. Juels and M. Sudan, "A fuzzy vault scheme," *Designs, Codes and Cryptography*, vol. 38, pp. 237–257, 2006, 10.1007/s10623-005-6343-z.

[15] A. Juels and M. Wattenberg, "A fuzzy commitment scheme." ACM Press, 1999, pp. 28–36.

[16] V. Guruswami and M. Sudan, "Improved decoding of reed-solomon and algebraic-geometric codes," in *Foundations of Computer Science, 1998. Proceedings.39th Annual Symposium on*, nov 1998, pp. 28 –37.

[17] B. Parks, "Mffr (multi-feature face recognition) user manual," University of Colorado at Colorado Springs, Tech. Rep., 2011.

[18] N. Pinto, J. J. DiCarlo, and D. D. Cox, "How far can you get on a modern face recognition test set using only simple features?" *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2009.