Stony Brook University

Efficient Skimming of Text

A Thesis Presented

by

Aaswad Anil Satpute

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Master of Science

in

Computer Science

Stony Brook University

May 2016

Stony Brook University

The Graduate School

Aaswad Anil Satpute

We, the thesis committee for the above candidate for the

Master of Science degree, hereby recommend

acceptance of this thesis.

Prof. I.V. Ramakrishnan
Professor, Computer Science department

Prof. Paul Fodor
Professor, Computer Science department

Prof. Yevgen Borodin
Professor, Computer Science department

This thesis is accepted by the Graduate School

Charles Taber
Dean of the Graduate School

Abstract of the Thesis

Efficient Skimming of Text

by

Aaswad Anil Satpute

Master of Science

in

Computer Science

Stony Brook University

2016

This paper is an attempt at describing the priority of skimming in enhancing the lives of the visually impaired and dyslexic. The paper distinguishes summarization from skimming. It speaks of skimming a given blob of text. And with it, arises some unique challenges. So, the paper tries to tackle these challenges by enhancing and speeding up the algorithm. The algorithm breaks down the input into smaller and compact skimmed results, which when used as the input to the algorithm completely fastens the algorithm with a tremendous decrease in run time. The ways of creating the compact skimmed version of the text are described in the paper, at the same time comparing and contrasting the differences between those ways. The paper finally describes the fastest algorithm, its origin and the way to derive the shortest skimmed version to input to the final algorithm.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank the following for their support throughout the project

1. Prof. I. V. Ramakrishnan, advisor (Stony Brook University) for all his guidance and help
2. Prof. Ritwik Banerjee (Stony Brook University) for his inputs to the project and his guidance at every step
3. Dr. Faisal Ahmed (Stony Brook University) for the algorithm to work on, and the code to summarize a given text
4. Vasudev Balasubramanian (Stony Brook University) to help understand Faisal's code and be a part of conversation with Faisal
5. Uma Nagarsekar (University of Southern California) to help me in several parts of the thinking process which helped me get the core algorithm's idea in place
6. Saurabh Netravalkar (Cornell University) to help think out loud my basic algorithm so which helped me get pen the new algorithm making it way more faster
7. Aditi Mhapsekar (University of Illinois at Urbana–Champaign) to help me understand CoreNLP
8. Partha Patil (Cambridge University) to help proofread the report and make it even more concise and to the point

Definitions

Summarization

A summary [4] is the condensed presentation of the body of the material. This is the procedure to manipulate and modify the existing textual material in such a way so that the resulting text meets the following conditions

- The output of the procedure should yield a text that is much minuscule than the input larger text.
- The output text must not lose its meaning in any way. It should syntactically, semantically same as the input text. The outcome of the procedure should be exactly identical to the input text in its meaning.
- There should be absolutely no loss in the meaning and the output would manage to have all the pivotal words that are necessary to preserve the meaning of the entire text.

Precis Writing

A precis is also a summary of the body of the text, but it usually contains one-third of the words present in the original text. A precis is a miniature portrait of the text or the passage. The precis writing is brief and to the point but preserves the meaning of the entire text.

Selecting important words from a paragraph

This is the procedure where we select the vital words in the paragraph or body of the text. These are the words that usually conserve the meaning of the text. The most important word in the paragraph will be chosen every time we try to categorize the important words in the paragraph. The next important word in the course of the text will be the word that will be chosen next to the most prominent word and is always chosen. Thus the first, second and third vital words are always bound to be picked and selected in the sphere of the summary and precis writing of the input text. These words mark the clarity, brevity and the precision of the input text. These words maintain a logical order of the text. These words make the passage well-knit and coherent and maintain the connectivity and flow of the meaning.

Skimming

The skimming is the process that is like the first page of the book. Just like when we attempt to read a book, we read the first page of the book and make a resolution to read the entire book or not. Some of the visually impaired and dyslexic people who are unable to skim through the introduction of the page may have to hear the entire book content or at least the start of each and every sentence before they can decide to continue further or not, by using audio books or a

reader. The screen readers [1][2] read the software from start to finish. But then when they are halfway into the book, they may decide that the material is uninteresting to continue any further and might want to stop reading it. But they have already invested some of their valuable time in processing the information till that part of the book. But if they had an opportunity to skim through the gist of the data, information or the book, then it would have been profitable to them to make an appropriate decision to read through the material now or save it for later reading.

Overview to summarization and skimming

The procedures of summarization, precis writing and selecting the important words from the paragraph are pondered upon and experimented with from times immemorial. These ideas are age old and not new to the researchers. The summarization is one of the most explored areas. It is the need of the ever growing data that necessitates the people who access that data to summarize it because a summary is effortless to read than reading the whole text.

The precis is also a compressed and diminished version of summary. So the ever expanding needs of data storage requirements made the researchers explore ways to make the summary even shorter. To equip this arena of data handling, many research papers and algorithms were developed [4]. These algorithms evolved and got refined in due course of time. So we have an immense number of algorithms, research papers and developments in the area of text summarization. But the paramount and crucial step and pace in both the above processes of summarization and precis writing is to identify and hand-pick the most significant words which span and play the most important meaning of the text under consideration.

Difference between Skimming and Other Techniques

Skimming is more like a human approach to get the gist of the text but summarization is more like a well thought out approach. Summarized text needs to preserve and respect the rules of grammar. Skimming is a rapid reading method that allows to obtain the specific information that is of utmost importance in understanding and grasping the type of content of the data. Summarization contains all the information of the data in a concise way. They are particularly useful in determining a preview of the data and when we do not want to read through each and every word of the text. These methods help us to quickly identify the main ideas of the text. Skimming helps us identify all the essential information to help identify what things might be present in the rest of the content. Precis writing is an intelligent summary of the text and needs a clear understanding of the material to decide what are the most important points in the text that provide an insight on the essential points. Selecting the few pivotal words in the entire text is to pick and select a very few words and phrases. But the precis writing is an accumulation of several phrases or words to convey the importance of the text as a whole.

Selecting the important words of the text is an approach that might always not work as desired because there might be a lot of very important or a lot of many unimportant words in the text. Sometimes, keyword search might not be possible on a text or it might not result in the words we want to understand the entire gist of the content. In these cases, skimming approach is a major rescue tool, since, skimming allows us to look for specific information in the text without

making the user lose the basic understanding of the content. Thus skimming is always effective in saving the users from heavy reading load.

Importance of Skimming

Usually when someone is planning to read a passage, essay or even a book for that matter, the first thing they do most often than not is skimming the preamble of the book, introduction of the passage or abstract of the essay. But without skimming these most people don't confirm their plan to read. Skimming is the primordial step which aids the user to make and adhere to the decision as to read the information ahead or not.

Proposal

As a matter of utmost importance that was sought and most discussed in the world is to come up with more innovative ideas to make available a gist of the information to the visually impaired [5] and dyslexic people. Rich research [3] has been made to make reading an enjoyable task for the visually impaired. As the other users skim through the material and decide to identify the importance of the material they are about to process, even these disabled people must be provided with a way to hear to the gist of the material. So, we now attempt to equip them with a tool that will take the input as the text whose skimming is to be done. The output is the gist of the material that will be the short summary of the text. And we design a text to speech conversion software which will read the output of the tool and empower the disabled to obtain a summary of the material and make a judgment and speculate whether investing time to hear and process the material would benefit and profit them or not.

Previous Work

This is mainly Faisal Ahmed's work [6]. On which I have worked on enhancing it to make it real-time.

The grammar is the basic building backbone of a sentence. So every sentence is first parsed to extract all the grammatical relationship between all of its words. Then a lexical tree is constructed based on these relationships in the sentences, with each node of the tree representing a word in the sentence. Next, for every word in the text, 2 features are identified. They are - its grammatical features (like POS tags) and its structural features (in degree/out degree) are identified. These features are then fed to trained classifier which resolves whether to include the word in the skimming summary. As the final step of the algorithm, the subtree which consists of all the selected words is constructed. And this subtree represents the skimming summary that all the users can obtain when they interact with the system via a user interface. Thus the key elements of the algorithm would be the language parser, classifier and a skimming interface.

One such parser is the Stanford Parser, which is a natural language parser that can analyze the grammatical and syntactical structure of sentences and extract relations. The relations identified by the parser are simple and accessible to all. Stanford parser identifies 48 different binary grammatical relations that exists between governor word and dependent word. Using these relations, a directed graph is constructed where the nodes are the words and the edges are the relations from the governor to the dependent. This is a sentence tree.

We need datasets to train the classifier. The summaries were collected by using trained participants who produced gold-standard summaries of all the chosen newspaper articles and then up-voting them. The purpose of skimming was to pick up most salient information without content rephrasing. The goal was to produce summaries which were ½ to ⅓ the original content satisfying the following rules - each sentence separately summarized and with punctuation, word order and information preserved.

They designed an interface that facilitated the screen-reader users to switch seamlessly back and forth between reading the skimming summary and reading the original web page preserving the current reading position, so that the blind are allowed to scan the text easily and quickly at times, and to slow down and read text slowly at other times, just like the sighted readers. When switching between summary and the regular screen reading, the cursor position is placed at the closest preceding word that was present both in original text and in summary. The interface can skim on any web page. But if a page contains a list of links, then that page will be read entirely. The features that were the output of the Stanford parser that were used to train the classifier are - Number of Outgoing Edges (normalized by no of nodes in the tree), Level in the Tree (normalized by highest node in tree), Number of descendants (normalized by highest number of descendents of a node in tree), Incoming relation type, and POS tag. A feature vector was generated using Stanford parser can and every word in dataset. Then, to classify a feature vector, we check if the corresponding word appears in the gold summary. If it appears, vector is labeled as “YES” class otherwise as “NO” class. “YES” class means this word is selected to be used in summary and “NO” class meant otherwise. These datasets are used to train different Classifiers through Weka. The “Yes” class words are rearranged to preserve meaning. If there are a set of words that are repeated often, then LCM (Lowest Common Subsequence) algorithm was used to chose the set. A MinConnectedTree (Minimum Connected Tree) algorithm was used skim through a given sentence. Another algorithm named SkimSentence was used to select phrases because users skim through a set of words to understand the content better when skimming. The algorithm in paper [6] is described in Figure I.

Definitions:

- **R** is a set of dependencies in the type relations. It represents relations between the words where the mapping is created between Governor depends on Dependent.
- **T** is a tree where the parent is the Governor and the Child is the dependent.
- **S** is the input that consists of the sentence to be skimmed.
- **K** is the output and is the skimmed sentence.

Algorithm:

- `SN = set() #set of the summary nodes`

```
• K = list()  #it is the list of the words
• R = TypedDependencies(S) #from the Stanford Parser
• T = ConstructTree(R) #obtained from the typed dependencies
• For node N in T:
    o O =  number of Outgoing Edges of node N
    o L = LevelInTree(N)
    o C  = number of Descendants of node N
    o I = IncomingRelationType(N)
    o P = POSTags(N)
    o F =  <O,L,C, I, P> #feature vector
    o If InSkimming(F) = True #SVM classify
        ▪ SN.add(N)
• MT = MinConnectedTree(SN)
• K = OrderAndListWords(MT)
• Return K #the skimmed sentence
```

Figure I. Algorithm of paper [6]

Aim

The aim I worked on was to speed up the previously defined skimming algorithm according to paper [6] and to make it more efficient. Also, to create a better Skimmed version. Which means I would have less number of words selected in the final output, thus a better skimmed results. This in turn should help me speed the algorithm as well.

My Approach

Any textual material consists of a large collection of sentences. Some of the sentences are so relevant to the text that their removal is infeasible. They are of monumental and massive importance to preserve the overall meaning of the text. On the other hand, there are some other sentences in the text that we are trying to process that might be redundant to the context of the text. They might convey the same idea again that was conveyed by an earlier important sentence or a group of words. Thus, they might be of less importance. And the key idea here is that the removal of those sentences might not make the text lose any of its meaning. And this idea that the input text to the algorithms can be effectively reduced in size is a major breakthrough idea that will ease our tasks by an enormous amount in the future.

By using the idea just mentioned, the input to the algorithm can be effectively reduced. If the algorithm in the beginning was fed with an input of size, say, a 100 sentences, now, after the preprocessing step, will be fed only 60-70 sentences instead of 100. This is a huge diminishing factor as far as the size of the input is concerned.

The algorithm suggested in the previous section requires a great deal of runtime and is a bottleneck algorithm. If the input to the algorithm is large, the bottleneck algorithm takes a huge amount of time to run and yield results. But with an additional step that will reduce the input fed

to the bottleneck algorithm, the efficiency can be effectively increased by a great amount. So we employ a threshold. This threshold determines how many sentences can be removed from the text without losing its meaning and how many sentences need to be present because of their massive importance. This threshold value can be effectively decided upon by the users of the application. And larger the threshold value, lesser is the number of sentences that are permitted to be removed from the algorithm. On the other hand, if the threshold value is smaller, the users are at the liberty of removing a greater number of sentences without harming the meaning of the original text. This type of preprocessed input when fed to the bottleneck algorithm yields results in a much shorter span of time. And so running the bottleneck algorithm, which earlier utilized more time with the unprocessed input will now run much faster. The effective running time including the preprocessing step and the bottleneck algorithm (with the processed text, reduced in size, as input) combined, is much smaller than using the raw unprocessed text. This accelerates the entire process. And this is a massive and novel idea that polishes and ameliorates the running time and efficiency of the bottleneck algorithm. Now we will discuss in detail how to filter out and pick the sentences that can be removed and those that to be mandatory retained from the original text and how my approach does the filtering of input.

Original Approach

The original text has to be now filtered. The opening step in the approach is to breakdown and identify each and every sentence present in the original text. We thus obtain, a set of all sentences. Then we attempt to separate out words and group them into sets. We first identify all the nouns that are present. We identify the nouns in each and every sentence iteratively. After identifying the nouns in every sentence, we group them into a set. This is a set of all the nouns that are present in the original text material we have considered. Similarly, we then isolate all the verbs that are present in each and every sentence of the text. Then we group them into a set. This is a set of words, where each and every word of the set is a verb. This set contains all the verbs that are present in the original text.

So, after the execution of the introductory step, we now possess a list of sentences, list/set of all the nouns and a set of all the verbs that are present in the original text. Now we proceed to decide which of the nouns are more important than the others in the list. Similarly, we identify and filter which of the verbs are more important than the other verbs in the list of verbs.

So, we have a user defined threshold that can be negotiated. We then compare the meaning of each of the word in the list of nouns with every other word in the set of nouns. And we determine the extent of similarity in the meaning of the noun with the other nouns. If two nouns are similar to each other a lot, then we retain them. If the two nouns are very similar to each other in their meaning and the extent of their similarity is greater than the user defined threshold value, then they are retained in the refined list of nouns. Else if the extent of similarity between the two nouns is lesser than the threshold value, then they can be removed from the refined list. And this is performed for all the nouns in the list. So every noun is compared with every other noun for similarity.

And this step is repeated in the same way for all of the verbs in the list of verbs. Similar to the nouns, we identify and filter which of the verbs are more important than the other verbs in the list of verbs. So, we have a user defined threshold for verbs that can be negotiated. We then compare the meaning of each of the word in the list of verbs with every other word in the set of

verbs. And we determine the extent of similarity in the meaning of the verb with the other verbs. If two verbs are similar to each other a lot, then we retain them. If the two verbs are very similar to each other in their meaning and the extent of their similarity is greater than the user defined threshold value, then they are retained in the refined list of nouns. Otherwise if the extent of similarity between the two verbs is lesser than the threshold value, then they can be removed from the refined list. And this is performed for all the verbs in the list. So every verb is compared with every other verb for similarity.

Now, lets us analyze with a very mundane simple example how this might help refine the important sentences in the text. Suppose we collect textual material of several people expressing their views about the New York City. We can thus have textual opinions of several people discussing and expressing several things above New York, like the major tourist attractions, weather, culture etc. But there might be a sentence that says "My uncle has traveled there", which is not so relevant to the entire context. So by skimming through the other opinions, we might be attempting to obtain useful information and get a bird's eye view of the place. But this sentence is of no relevance to this aim of ours since it does not convey any useful information about the place. So the set of verbs or nouns that might be present in this sentence are not so much in similarity with the other nouns or verbs. And removing them may remove irrelevant words from the set. And that sentence can itself be removed without losing any useful information. Thus, we now can obtain a refined list of sentences, nouns and verbs, all of which are important and relevant to the text and context.
Then, we attempt to form the lexical chains [7] for each and every noun that is present in the list of the nouns. These lexical chains of words from a WordNet encompassing several levels of words with similar meaning woven together as a chain at several levels. So, we take a noun or a verb into consideration, and try to form lexical chains of the word. So, the words that are very similar in meaning to the noun or the verb under consideration are placed at the second level, with the original word itself at the first level. Then the words that are very similar in meaning to the words at the second level are placed at the third and all the consequent levels with the original word (noun or verb) at the head or root of the chain. These lexical chains may span up to several levels. The key idea is to parse through several levels of the lexical chain and determine which two of the nouns or verbs has the words with very same meaning or same words down the lexical chain in the WordNet. These are the words that are very similar to each other in the meaning. Since, we had to limit the number of levels that one would traverse to determine similarity between two nouns or verbs under consideration, we chose to traverse to a depth of 5 levels down the lexical chains of the words. So, we then retained the words which had the words same at the fifth or earlier levels of depth in the lexical chain as the two nouns or words that have similar meaning. Otherwise, those words can be removed from the refined list of the nouns or refined list of verbs. And thus after this step, we have in possession the refined list of nouns and verbs. Then we can run through the list of all the sentences in the text, and pick the sentences that contain the nouns or verbs present in the refined set of nouns and verbs. Rest of the sentences can be removed. So, we conserve the important and relevant sentences from the text. The procedure is summarized in Figure II.

- `Input T is the text that needs to be summarized`
- `S = set() #all the identified sentences`

- N = set() #all the nouns identified in the text
- V = set() #all the verbs identified in the text
- W = set() #all final nouns and verbs of relevance to the text and thus, are retained
- nounMeaningsList = list() #list of the trees of nouns and its meanings (till the depth of 5th level)
- For noun in N:
  - nounMeanings = tree() #tree of a noun and its meanings (till the depth of 5th level)
  - For x in range(5): #number of levels in the lexical chains is limited to be 5
    - nounMeanings.getNextNoun() #get the next noun (Breath First Traversal) from the nounMeanings' tree
    - allMeanings = getAllMeanings(noun) #get all meanings of the noun using WordNet
    - For aMeaning in allMeanings: #loop through all the meanings of the noun
      - nounMeanings.add(noun, aMeaning) #add this meaning of the noun as the tree's next level node.
  - nounMeaningsList.add(nounMeanings)
- For i in range(length(nounMeaningsList)): #loop through all the nouns in the list, nounMeaningsList
  - If treeMatch(nounMeaningsList [i], nounMeaningsList [i + 1]): #if any node in any level of the tree (nounMeaningsList [i]) matches any other node in any level of the next tree (nounMeaningsList [i + 1])
    - W.add(nounMeaningsList [i].root.data()) # retain the root word of nounMeaningsList[i] tree which finds a match in the nounMeaningsList[i+1] tree
    - W.add(nounMeaningsList [i+1].root.data()) # retain the root word of nounMeaningsList[i+1] tree that was matched in the nounMeaningsList[i] tree
- Repeat the above steps for the set of verbs
- Retain the sentences from the input text that contain words in the set W to obtain the Output X
- Return X

Figure II. Algorithm of the Original Approach

Issues with this Approach

Though the above mentioned approach is good, it does not fare well because of the diversity in the meanings of the words of the language. And so the effective time to refine the set by identifying the words that are similar in meaning is quite very large. Suppose, we consider that there are n number of words that need comparison. And we are traversing five levels of depth down the lexical chain. If we assume that each word is similar to 4 other words in its meaning, and those 4 words are similar to 4 more words in their meaning, then we end up having almost 1024 number of words with a depth of 5. Suppose if the number of words is 100, then we would effectively have almost 102,400 number of words that need a comparison with each other, which is a huge number. But there are always more than 100 words in the text. So, the total number of comparisons that might be required is far larger than the just discussed numerical figures. And since the language has a lot of words that are synonyms of each other and have similar meanings to each other, we end up having a greater set of words that might have very similar meanings to each other. And the set is much larger than expected. And the number and scale of the words that might be similar in meaning to the word under comparison might be much more than 4 words for every word. So, we end up having a larger set at hand, and the number of comparisons that result from having to compare every word in the set with every other word in the set is tremendously massive. And thus this approach does not scale well with the increased number of comparisons that we have to perform to fetch the refined set of words. These are the major hindrances with the approach.

Current not optimized Approach

This is novel idea that is much better than the previous approach because it reduces the total number of comparisons that we need to perform to obtain a refined set of nouns and verbs. First we obtain a dimensional array of nouns and verbs. Each and every row of the two dimensional array consists of one noun followed by all the other nouns in the list. Similarly, we get the matrix of all the verbs too. Each and every row of the two dimensional array consists of one verb followed by all the other verbs in the list. Next we need to populate this multi-dimensional array with the similarity quotients. To do that, we use the Princeton WordNet. This is an application that takes in two nouns or verbs as input and tells us the probability of similarity that might exist between these two words. In other words, this tells us how much the two words are similar to each other.

So, we go ahead and fetch the corresponding probabilities of the words. These are the elements that populate the two dimensional arrays that we created earlier. Now these are numerical values and thus they are probabilities. So, they will a value starting anywhere from 0 and between 1. So, we obtain a unique probability number for every set of words. These values are then populated in the two dimensional array of both nouns and verbs. So, we now have the extent of similarity between all words. If we have 100 words in the list like we discussed earlier in the previous approach, we will now have 10,000 matches of words to be fed in WordNet and populated now as matrices. This is much efficient and better than having 102,400 number of comparisons.

Then, as the next step in the approach, we scan through the elements of the two dimensional array and see if each and every element in the array is greater than the user defined and set threshold value. If the value of the element is greater than the threshold value, then we set the

value of that element as 1. If the value of the element is lesser than the threshold value, then we set the value of that element as 0. This step is repeated both the set of nouns and verbs. So, we end up in having two two-dimensional arrays of all nouns and verbs respectively having either values 0 or 1 as its elements. This procedure is carried out for all the elements of the two-dimensional array except for the diagonal elements.

After this step, we then scan through all the rows of the two-dimensional array. If the row contains all zeroes as its elements, then we remove those words (may be nouns or verbs) from the final set. If one or more of the entries in the two-dimensional arrays is one, then we retain those words from the final refined set of nouns and verbs. And following this approach reduces the number of comparisons by a large significant number than the previous one. The procedure is summarized in Figure III.

```
• Input T is the text that needs to be summarized
• S = set() #all the identified sentences
• N = set() #all the nouns identified in the text
• V = set() #all the verbs identified in the text
• W = set() #all final nouns and verbs of relevance to
  the text and thus, are retained
• K = 0.4 #threshold for the probability of two words' to
  be synonyms (which have similar meanings)
• Construct two-dimensional arrays of nouns nounArray and
  verbs verbArray
• For i from 0 to length(N):
    o For j from 0 to length(N):
        ▪ nounArray[i][j] = getSimilarity(N[i], N[j])
          #get the probability of extent of similarity
          between the words N[i] and N[j].
• For i from 0 to length(N): #apply the threshold on
  the whole matrix
    o For j from 0 to length(N):
        ▪ If nounArray[i][j] > k :
            • nounArray[i][j] = 1
        ▪ Else:
            • nounArray[i][j] = 0
• Repeat the above steps for the set of verbs and
  populate verbArray
• For i in length(nounArray): #For each row in nounArray
    o If at least 1 element in nounArray[i] is 1:
        ▪ W.add(N[i]) #populate final set of words
• Repeat the above steps for the set of verbs and add
  them to W
```

- Output X = refined text containing sentences with the final set of verbs and nouns in W (these are the only relevant and important sentences).
- Return X

Figure III. Algorithm of Current non-optimized approach

Completely optimized current Approach

This approach is much better optimized than the earlier approaches, which will be evident to the users from the following discussion. In this approach, we have the input as all the list of sentences, set of all the words (both nouns and verbs) which we have identified in the step 0 of the approach as mentioned earlier. This is the first crude set. Now, we employ a new approach. Here we start from the first word in the set, and begin by comparing it with the other words in the set. Here comparison essentially means that we will have probability by which the two words can be similar (fetched from the Princeton WordNet application) higher than the threshold value. If this is true, then we break through the loop for comparison and add the first word into a new set of words. We then repeat the same step with the second word, compare it till we find the next similar word in the list of words. Then once, the word is found, we add the second word to the new set. This is repeated for all the words in the list. And thus, we will have a new set of words (one new set of nouns and one new set of verbs) that will have all the words that were added during the run of the optimized algorithm.

It is by far the best approach because it reduces the number of comparisons needed to populate the final set of words by a maximum number. In the previous approach, if we had 100 words, we would require almost about 10,000 number of comparisons at any cost. It would be the same for best, average and worst case complexities. We had to compare each and every word with every other word. But with this new optimized approach, we just have to compare a word until a very similar word to it is found. Once a similarity is identified, it can be added to the refined set. So, here the average case complexity is far better and optimized than the average case complexity of the previous algorithm. If the input to the algorithm is a well-written textual material, then it will usually speak of a definitive topic and have firm views about it. So, the chances of a noun or a verb not finding a word with a similar meaning down the list is almost nil. But on the other hand, if the input text is poorly written with no definitive topic, then we might have to do the same number of comparisons that we did previously. But this is the worst case complexity of the optimized algorithm. But on the bright side, the average case complexity is significantly improved by using this optimized algorithm against 10,000 comparisons that were mandatory to be done in the previous algorithm. So, this is a significant improvement in efficiency as it reduces number of comparisons to a great extent. The procedure is summarized in Figure IV.

```
• Input T is the text that needs to be summarized
• S = set() #all the identified sentences
• N = set() #all the nouns identified in the text
• V = set() #all the verbs identified in the text
• W = set() #all final nouns and verbs of relevance to
  the text and thus, are retained
• K = 0.4 #threshold for the probability of two words' to
  be synonyms (which have similar meanings)
• For i in length(N):
    o If getSimilarity(N[i], N[i + 1]) > K: #get the
      probability of extent of similarity between the
      words N[i] and N[i + 1] and compare it to K
        ▪ W.add(N[i]) #add N[i] to W
        ▪ W.add(N[i + 1]) #add N[i + 1] to W
• Repeat the above steps for the set of verbs and add
  them to W
• Output X = refined text containing sentences with the
  final set of verbs and nouns in W (these are the only
  relevant and important sentences).
• Return X
```

Figure IV. Algorithm of Current optimized approach

Results

The skimming algorithm, defined in the paper [6] was run on a set of 28 randomly selected Wikipedia articles for setting a standard against which the results of my algorithm shall be compared to. My algorithm was also executed on the same 28 Wikipedia articles, and its output was fed as input to the above described, skimming algorithm [6]. And these two set of outputs generated with and without my algorithm in the pipeline were noted and compared with each other. These algorithms were run on the exact same machine to avoid any disturbance in difference in time taken due to different processor speeds.

Comparing the speed of both the outputs considered in an end-to-end fashion is that, it takes about 29 minutes for the skimming algorithm [6] to run for the given set of data. The original algorithm when executed on the input, takes a really long time to run, which stretches over 5 hours. After this time period, I had to stop executing it. The current not optimized algorithm takes about 31 minutes to complete its execution. On other hand, the total end-to-end run time of my such a setting takes only 13 minutes. My algorithm when run, before skimming algorithm defined in paper [6], it speeds up by a lot, because the run time changes every once in a few run cycles, these timings are an average of 15 run cycles. The output of execution of all the above mentioned algorithms is the same, resulting in a skimmed version of the input, but with different times and speeds of execution in the algorithms.

Now to quantify, the results, here is a table of 28 different input files and number of words selected by the skimming algorithm [6] on the unedited text and the skimming algorithm [6] on sentence selected text created by my algorithm in place and the percentage of improvement that my algorithm resulted.

File	Number of Words By Skimming Algorithm	Number of Words by Skimming Algorithm on Selected Sentences	Percentage improvement
1	288	123	42.71
2	201	110	54.73
3	234	95	40.60
4	240	77	32.08
5	343	176	51.31
6	206	154	74.76
7	158	40	25.32
8	232	34	14.66
9	152	50	32.89
10	167	70	41.92
11	358	212	59.22
12	232	156	67.24
13	202	92	45.54
14	191	106	55.50
15	213	84	39.44
16	230	150	65.22
17	169	43	25.44
18	289	156	53.98
19	207	143	69.08
20	276	153	55.43

21	186	116	62.37
22	196	57	29.08
23	189	123	65.08
24	145	85	58.62
25	259	67	25.87
26	235	112	47.66
27	283	67	23.67
28	252	141	55.95
Total	6333	2992	47.24

Table I. Comparison of the results

Discussion

Algorithm for text summarization by sentence extraction [8] describes an algorithm which accomplishes an aim closest to my aim. Every experiment in the paper comprises of the following steps as described in the words of the author are summarized in Figure V.

- Preprocessing step: includes removing all the stop words (like the, in) followed by the application of Porter stemming [9].
- Term selection step which comprises deciding the features (size of n-grams) to be used to describe all the sentences in the text.
- Deciding a method to calculate the importance of each feature is the next step termed as term weighting.
- Sentence clustering phase which decides the initial seeds for k-means algorithm as the baseline sentences.
- Selecting the closest sentence to each centroid for constructing the summary is the sentence selection step.

Figure V. Algorithm of text summarization by sentence extraction

The algorithm employed in the paper which was summarized and described in the Figure V, is a testimony to clearly describe the major differences between the approach followed by me in my algorithm and the approach which is deemed as the gold standard approach to perform a similar task. The major difference that exists between the two approaches, is that this approach is performed using Machine learning approach. This approach highly depends on the machine learning techniques. And for a machine learning technique to work, we need to first create a model. And we will need to train the model. And we require a huge amount of training data to create a model. And we gather this data.

Once we have the training data in place, we can create an appropriate model. And then we train this created model by using the above gathered training data. Then the trained model is utilized every time the code is executed, to obtain the required output. To achieve this model, we will require initial training time to create the model. And then we will need some additional time to train the model with the data.

But with the help of my approach, the additional time for training the model is eliminated because we do not have to train the model. I, in my approach, use the WordNet application which is a lot faster than the regular machine learning approach that is employed in the algorithm described on the paper. And using WordNet do not require any sort of training of the model. The most important benefit that can be obtained from using my algorithm is that it does not need any prior training. In my approach, we can transfer the main file of the program into any platform, which can be executed on the fly as and when needed. This does not necessitate the requirement of any prior setup code. This program can be executed to obtain the results without any training data. But the paper was a major inspiration to develop an easier algorithm throughout the development of the Thesis project.

Links

These are the links of all the codes, input files and the output files

- My Code - https://drive.google.com/folderview?id=0B4XC1hBF8VqkejA4bDV2QzIxYWs
- My Final Output - https://drive.google.com/folderview?id=0B4XC1hBF8VqkbGQzbC14Vnh2LUE
- My Intermediate Outputs with Comparisons - https://drive.google.com/folderview?id=0B4XC1hBF8VqkWXItcjFLNFV0YWc
- Faisal's Code - https://drive.google.com/folderview?id=0B4XC1hBF8VqkTjZxN3MyRUFNV2s
- Faisal's Output - https://drive.google.com/folderview?id=0B4XC1hBF8VqkLVZzVmRQOVlkQ2M
- Common Inputs to both algorithms - https://drive.google.com/folderview?id=0B4XC1hBF8VqkR2lxdzduQVRLNWM

If any more information or explanation is needed, please feel free to write to asatpute@cs.stonybrook.edu.

References

1. Ahmed, F., Y. Borodin, Y. Puzis, and I. V. Ramakrishnan, *Why Read if You Can Skim: Towards Enabling Faster Screen Reading, in International Cross-Disciplinary Conference on Web Accessibility*. 2012.
2. Ahmed, F., Y. Borodin, A. Soviak, M. A. Islam, I. V. Ramakrishnan, and T. Hedgpeth, *Accessible Skimming: Faster Screen Reading of Web Pages, to be appeared in UIST*. 2012, ACM: Cambridge, Massachusetts, USA.
3. Ahmed, F., M. A. Islam, Y. Borodin, and I. V. Ramakrishnan, *Assistive web browsing with touch interfaces, in Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility*. 2010, ACM: Orlando, Florida, USA. pp. 235-236.
4. Goldstein, J., M. Kantrowitz, V. O. Mittal, and J. G. Carbonell, *Summarizing Text Documents: Sentence Selection and Evaluation Metrics, in In Research and Development in Information Retrieval*. 1999. pp. 121-128.
5. Grefenstette, G., *Producing Intelligent Telegraphic Text Reduction to Provide an Audio Scanning Service for the blind, in Workshop on Intelligent Text Summarization*. 1998, American Association for Artificial Intelligence Spring Symposium Series: Menlo Park, California. pp. 111-117.
6. Ahmed, F., and I. V. Ramakrishnan, *Algorithms and Interfaces for Automated Non-Visual Skimming*. 2012, Doctoral Thesis, Stony Brook University.
7. Hirst G., and St-Onge D., *Lexical chains as representations of context for the detection and correction of malapropisms*, University of Toronto. Toronto, Canada, 1995.
8. R. A. García-Hernández, R. Montiel, Y. Ledeneva, E. Rendón, A. Gelbukh, and R. Cruz, *Text Summarization by Sentence Extraction Using Unsupervised Learning*, MICAI 2008: Advances in Artificial Intelligence: 7th Mexican International Conference on Artificial Intelligence, AtizapÃ¡n de Zaragoza, Mexico, October 27-31, 2008 Proceedings
9. Spark Jones, K., Willet, P.: *Reading in Information Retrieval*. Morgan Kaufmann, San Francisco (1997)