GRAPH BASED ALGORITHM FOR TEXT SUMMARIZATION

GOAL : To develop a sentence extractive, single document summarizer.

METHOD : Basic idea is to build a semi-automatic extractive summarizer by combining Google's page rank (Brin and Page, 1998) algorithm and weighted graph based representation of document. WG representation offers powerful and effective features the graph theory offers. The rank carries the significance of a vertex (sentence) in the graph (document) by accounting all the global information from entire graph. At the end of each iteration we assign a score/ rank to the vertex . The connection between sentences (edges) can be composed based on similarity between sentences . The similarity measure is calculated on many parameters like content overlap , further to avoid hiccups **(like long sentences obviously have high frequency of words)** with long sentences we can normalize the weights to respective sentence lengths, but in our case we have normalized to the maximum term frequency in a particular sentence. Main parameters would be Term frequency(TF) and Inverse document frequency (IDF). Using vector space model for representing sentences as indexes / vector of identifiers,we go by the classic *tf -idf* weighting system with some adoptions . *tf -isf* weights are computed for each sentence, where *sj* shows the *jth* sentence and *ki* is *i*th index term,

$$tf_{i,j} = \frac{\text{freq}_{i,j}}{\max_l \ \text{freq}_{l,j}} \qquad\qquad isf_i = \log \frac{N}{n_i}$$

tfi,j is said to be 'term frequency' of *i*th index term in the *j* th sentence, and *isfi* is 'inverse sentence frequency' of *i*th index term, where *N* is the number of all sentences and *ni* is the number of sentences which contain *ki* . The corresponding weight is therefore computed as,
$wi,j = tfi,j \times isfi$.
The similarity /edge weight between 2 sentences s_m and s_n is easily calculated based on cosine measure as follows

$$W\ (s_m, s_n) = \frac{\sum_{i=1}^{t} w_{i,m} \times w_{i,n}}{\sqrt{\sum_{i=1}^{t} w_{i,m}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,n}^2}}$$

The sentences are sorted based on ranks of nodes. The 'n' best sentences are chosen based on maximum cut off words/sentences in the summary. The original page rank combines the effect of both incoming and outgoing links.

$$PR(V_i) = (1-d) + d * \sum_{V_j \in In(V_i)} \frac{PR(V_j)}{|Out(V_j)|}$$

where d is a parameter set between 0 and 1.

The above equation has been adapted to include the notion of edge weights in the graph.

$$PR^W(V_i) = (1-d) + d * \sum_{V_j \in In(V_i)} w_{ji} \frac{PR^W(V_j)}{\sum_{V_k \in Out(V_j)} w_{kj}}$$

where
$PR^W(V_i)$ is Page rank of vertex V_i
$In(V_i)$ is all the predecessor vertices to node Vi
$Out(V_i)$ is set of vertices that V_i points to .

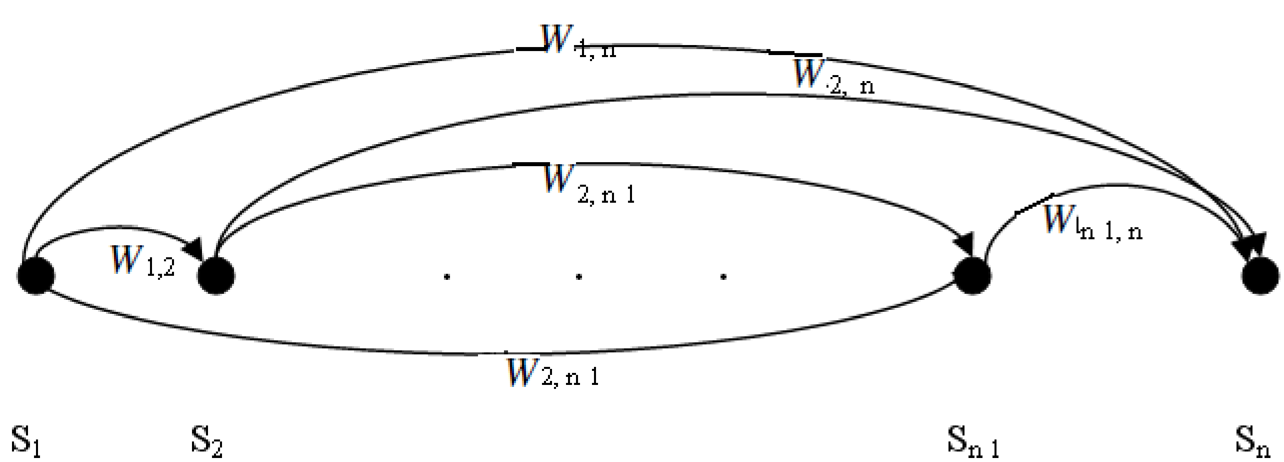
Graph Implementation possibilities

Forward directed : The edges only go out from a sentence to one or more sentences following it.

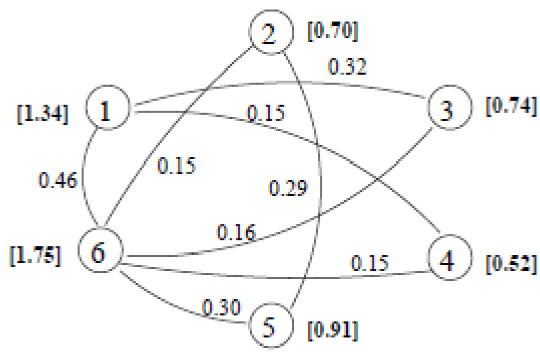
Backward directed : The edges only go out from a sentence to one or more sentences preceding it.

The forward DAG representation has been implemented ,tested and found that the algorithm seems to be biased and has been consistently ranking the sentences in the latter part of the document better than the starting portions. Hence going by the fact that 2 sentences are similar if one's contents are similar and one follows the other or vice versa we can use an undirected graph. This implementation seems to be giving positive and impressive results than its forward directed counter part. The following rules governing the graph structure would be adhered to :
1)There is no chronological differences between the sentences , only the contents carry importance.
2)There is also no self-edge, the similarity of every sentence to itself is considered to be 0.
This assumption is sorted here:
• $i < N$: $W(si, si) = 0$

Weighted forward directed DAG representation for a document



Weighted Undirected Graph



The nodes are indexed and the values beside them in square brackets are page rank. Additional space characters, tabs and punctuation marks will be ignored while performing content analysis.
DATA : Obtained the documents and summaries of NIST's Document Understanding Conferences (DUC) from ISI .

BASELINE ALGORITHM:

```
Feed the input document
MASTER = entire document.
OUTPUT = empty
Configure/Set  the maximum sentences in the summary <= Total sentences in document.

While ( ! done)
        if (maxSentences <= totalsSentences)
                for each sentence in list
                        if (counter <= maxSentences)
                                OUTPUT.put(currentsentence)
                        else
                                done = true
                                break
                if (done)
                        break
        else
                OUTPUT "Too many sentences"
                break
DISPLAY  OUTPUT.
```

PAGE RANK ALGORITHM:

```
Initialize all Ranks = 1 (# of ranks = # of sentences in the document)
while ! Converged  iterate
   for  i between [1 and numSentences]
      sum <--- 0.0 //////This is master sum.
      for  j   between [1 and  numSentences]
        if (j equals  i)//////Do not evaluate any sentence with itself.
           continue
```

```
                Wji <--- (j < i) ?getSimilarity(j and i):getSimilarity(i and j)
                PRVj <--- getRank(j)
                denSum <--- 0.0 /////This is denominator partial sum
                for  k   between [1 and  numSentences]
                      if (k equals j)
                        continue
                      Wjk <--- (j < k)?getSimilarity(j and k):getSimilarity(k and j)
                      denSum <----  denSum  + Wjk
                end for
                                 Wji * pageRankVj
              sum <----- sum + ------------------------
                                      denSum
          end for
      rank <--------  (1- DAMPING_FACTOR) + DAMPING_FACTOR X sum
      tmpranks.save(i, rank)
    end for
    diplayRanks(tmpranks)
    updateRanks(tmpranks)//This will clear / erase the older ranks and replace them with
newer rank, which is used in next iteration and so on.
  end while
```

SUMMARIZER ALGORITHM:

```
    Feed the input document
    MASTER = entire document.
    OUTPUT = empty
    Configure/Set  the maximum sentences in the summary <= Total sentences in document.
    for each sentence
        createGraphNode( )
        indexGraphNode( )
        findWordCounts( )
        findInverseSentenceCounts( )
    end for

    for each node in Graph
            find MAX term count
    end for

    for each node in Graph
            for each word in sentence
                                          term count of word
                  find termfreq <------  -----------------------
                                            MAX term count
                                                          Total sentences
                  find invSentencefreq <------  log ( --------------------------- )
                                                      inverse sentence counts
    end for

    for each sentence  [ call it source node]
          for every other sentence  [call it sink node]
```

```
            find prod <--------  termfreq X invSentenceFreq [for both source and sinks]
            if  [sentence contains more words than other]
                then
                        set prod <------- 0 [ for the corresponding indexes]
            Save (source prods) //Array of prods calculated for each word in src sentence
            Save (sink prods)//Array of prods calculated for each word in sink sentence
                            getDotProduct ( source prods,  sink prods )//Scalar product
            weight <-----  ---------------------------------------------------
                       RootSumSquares( source prods) X  RootSumSquares(sink prods )
            /////These edge weights are similarities.

        end for
    end for
    InvokePageRankAlgorithm();
    DISPLAY  OUTPUT.
```

EVALUATION: The summary data obtained from DUC will be used as reference against which the program generated summary will be compared to. Precision and Recall parameters were computed to measure the quality of Summary and accuracies estimated. .

The below table shows metrics depicting summarizer's performance.

Topic	Precision	Recall	F1 score
D0703A	0.6557788944723618	0.26337033299697277	0.3758099352051836
D0704A	0.6319444444444444	0.2746478873239437	0.38288920056100983
D0705A	0.6502890173410405	0.22842639593908629	0.33809166040571
D0706B	0.5971014492753624	0.206	0.30631970260223046

RESULTS: The PageRank algorithm was extremely fast in convergence, hence we required around 5 maximum iterations for most of the documents. The files in the DUC data are assorted topic wise. For instance the topic numbered 703A is all the files inside a directory named D0703A . Hence for ease of processing the content, we choose to merge all the files in a topic which is in turn used for summarizing.

The figure 'a' , below shows the distribution of F1 scores achieved by summarizing the all the documents with topics from 701A ,through 705A and from 706B through 710B. The low variance 0.12 shows that consistency of the summarizers performance though not extremely accurate. Actually targeted for accuracy about 50 % but could achieve 35 % – 42 %, the maximum being 42 %. For each topic there were summaries available from 2 or more human professionals generated . Hence compared the automated summarizer output with each of them and statistical mean was depicted as ultimate scores, giving equal weightage to all human generated ones.

As a future enhancement proposal we could have easily introduced more parameters like Readability factor (RF) and Topic relation factor(TRF) , calculated on each iteration which would strengthen the summary and make it more intelligible to humans.

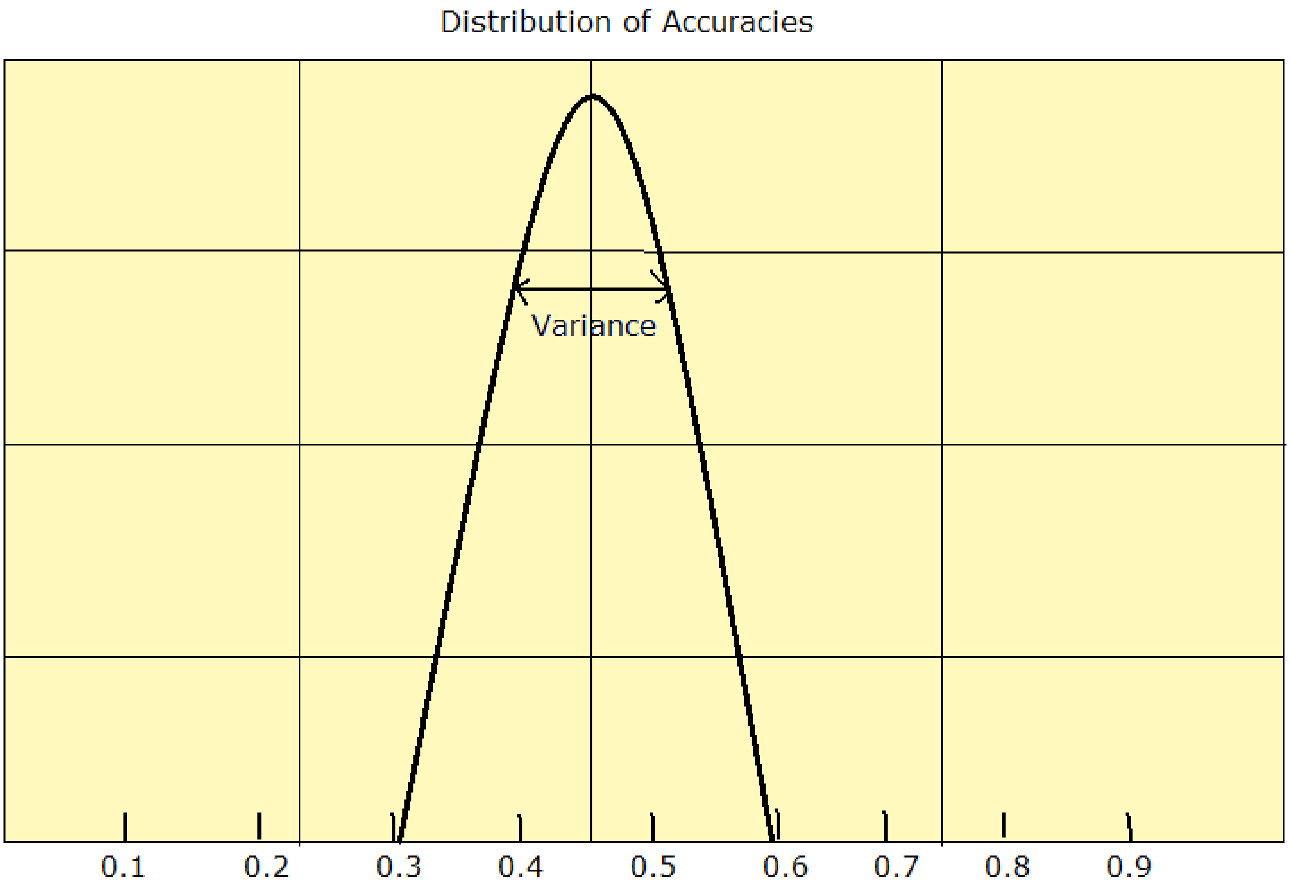


Figure a

CASE STUDY: The following is an instance of Summarizer in action.

INPUT DOCUMENT:

However, beware of one thing when you use this constructor. Algorithmic random number generators are not truly random. They are really algorithms that generate a fixed but random-looking sequence of numbers. When you create a random number generator, it initializes its sequence from a value called its "seed". The parameterless constructor for Random uses the current time as a seed, which is usually as good a seed as any other. However, the time is only measured to a resolution of 1 millisecond. So if you create two random number generators within one millisecond of each other. They will both generate exactly the same sequence of numbers. To generate a random integer from a Random object, send the object a "nextInt" message. This message takes no parameters, and returns the next integer in the generator's random sequence. Any Java integer, positive or negative, may be returned. Integers returned by this message are uniformly distributed over the range of Java integers.

EXTRACTED SUMMARY: (Configured maximum sentences to 6) SUMMARIZER OUTPUT

They are really algorithms that generate a fixed but random-looking sequence of numbers. However the time is only measured to a resolution of 1 millisecond. So if you create two random number generators within one millisecond of each other. They will both generate exactly the same sequence of numbers.
To generate a random integer from a random object send the object a nextint message. Integers returned by this message are uniformly distributed over the range

`of java integers.`

CONCLUSION:

It is intuitive that graph based iterative ranking algorithms works good on the task of extractive summarization as it not only accounts the local context of text unit but recursively draws information from the entire text (graph). As the graph is built, connections between various entities in a text, and implements the concept of *recommendation*. A text unit recommends other related text units, and the strength of the recommendation is recursively computed based on the importance of the units making the recommendation. In the process of identifying important sentences in a text, a sentence recommends another sentence that addresses similar concepts as being useful for the overall understanding of the text. Sentences that are highly recommended by other sentences are likely to be more informative, and will be therefore given a higher score.

REFERENCES:

Qazvinian, V.,Hassanabadi, L.S. and Halavati, R. (2008) 'Summarising text with a genetic algorithm-based sentence extraction', *Int. J. Knowledge Management Studies*, Vol. 2, No. 4, pp.426–444.

S. Brin and L. Page. 1998. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7).

DUC. 2002. Document understanding conference 2002. http://www-nlpir.nist.gov/projects/duc/.