A System for Query-Specific Document Summarization[*]

Ramakrishna Varadarajan
School of Computing and Information Sciences
Florida International University
Miami, FL 33199
ramakrishna@cis.fiu.edu

Vagelis Hristidis
School of Computing and Information Sciences
Florida International University
Miami, FL 33199
vagelis@cis.fiu.edu

ABSTRACT
There has been a great amount of work on query-independent summarization of documents. However, due to the success of Web search engines *query-specific* document summarization (query result snippets) has become an important problem, which has received little attention. We present a method to create query-specific summaries by identifying the most query-relevant fragments and combining them using the semantic associations within the document. In particular, we first add structure to the documents in the preprocessing stage and convert them to *document graphs*. Then, the best summaries are computed by calculating the top spanning trees on the document graphs. We present and experimentally evaluate efficient algorithms that support computing summaries in interactive time. Furthermore, the quality of our summarization method is compared to current approaches using a user survey.

Categories and Subject Descriptors
H.3.3 [**Information Search and Retrieval**]: Search process

General Terms
Algorithms, Performance, Experimentation.

Keywords
query-specific summarization, keyword search, Steiner tree problem, user survey

1. INTRODUCTION
As the number of documents available on users' desktops and the Internet increases, so does the need to provide high-quality summaries in order to allow the user to quickly locate the desired information. A compelling application of document summarization is the snippets generated by Web search engines for each query result, which assist users in further exploring individual results. The Information Retrieval (IR) community has largely viewed text documents as linear sequences of words for the purpose of summarization (with some exceptions as explained in Section 2). Although this model has proven quite successful in efficiently answering keyword queries, it is clearly not optimal since it ignores the inherent structure in documents.

Furthermore, most summarization techniques are *query-independent* and follow one of the following two extreme approaches: Either they simply extract relevant passages viewing the document as an unstructured set of passages, or they employ Natural Language Processing techniques. The former approach ignores the structural information of documents while the latter is too expensive for large datasets (e.g., the Web) and sensitive to the writing style of the documents.

In this paper, we propose a method to add structure, in form of a graph, to text documents in order to allow effective *query-specific* summarization. That is, we view a document as a set of interconnected text fragments (passages). We focus on keyword queries since keyword search is the most popular information discovery method on documents, because of its power and ease of use. Our technique has the following key steps: First, at the preprocessing stage, we add structure to every document (explained later), which can then be viewed as a labeled, weighted graph, called the *document graph*. Then, at query time, given a set of keywords, we perform keyword proximity search on the document graphs to discover how the keywords are associated in the document graphs. For each document its summary is the minimum spanning tree on the corresponding document graph that contains all the keywords (or equivalent based on a thesaurus).

The document graph is constructed as follows. First we parse the document and split it into text fragments using a delimiter (e.g., the new line character). Each text fragments becomes a node in the document graph. A weighted edge is added to the document graph between two nodes if they either correspond to adjacent text fragments in the text or if they are semantically related, and the weight of an edge denotes the degree of the relationship. There are many possible ways to define the degree of the relationship between two text fragments.

In this work we consider two fragments to be related if they share common words (not stop words) and the degree of relationship is calculated by an adaptation of traditional IR term weighting formulas. We also consider a thesaurus to enhance the word matching capability of the system. To avoid dealing with a highly interconnected graph, which would lead to slower execution times and higher maintenance cost, we only add edges

[*] Project partly supported by NSF grant IIS-0534530.

(v0) Brain chip offers hope for paralyzed
(v1) A team of neuroscientists have successfully implanted a **chip** into the **brain** of a quadriplegic man, allowing him to control a computer.
(v2) Since the insertion of the tiny device in June, the 25-year-old has been able to check email and play computer games simply using thoughts. He can also turn lights on and off and control a television, all while talking and moving his head.
(v3) The **chip**, called BrainGate, is being developed by Massachusetts-based neurotechnology company Cyberkinetics, following **research** undertaken at Brown University, Rhode Island.
(v4) Results of the pilot clinical study will be presented to the Society for Neuroscience annual conference in San Diego, California, on Sunday. Up to five more patients are to be recruited for further **research** into the safety and potential utility of the device.
(v5) BrainGate offers the possibility of hitherto unimaginable levels of independence for the severely disabled.
(v6) Although many are able to control computers with their eyes or tongue, such techniques remain dependent on muscular function and require extensive training.
(v7) John Donoghue, professor of neuroscience at Brown and a co-founder of Cyberkinetics in 2001, said that BrainGate could help paralyzed peopled control wheelchairs and communicate using email and Internet-based phone systems.
(v8) "Our ultimate goal is to develop the BrainGate System so that it can be linked to many useful devices," said Donoghue, who this month received an innovation award from Discover Magazine for his work on BrainGate.
(v9) "This includes medical devices such as muscle stimulators, to give the physically disabled a significant improvement in their ability to interact with the world."
(v10) Donoghue's initial **research**, published in the science journal Nature in 2002, consisted of attaching an implant to a monkey's **brain** that enabled it to play a simple pinball computer game remotely.
(v11) The four-millimeter square **chip**, which is placed on the surface of the motor cortex area of the **brain**, contains 100 electrodes each thinner than a hair which detect neural electrical activity. The sensor is then connected to a computer via a small wire attached to a pedestal mounted on the skull.
(v12) "While these results are preliminary, I am extremely encouraged by what has been achieved to-date," said John Mukand of the Sargent Rehabilitation Center, who oversaw the pilot study.
(v13) "We now have early evidence that a person unable to move their arms, hands and legs can quickly gain control of a system which uses thoughts to control a computer and perform meaningful tasks. With additional development this may represent a significant breakthrough for people with severe disabilities."
(v14) Surgeon Gerhard Friehs, associate professor of clinical neurosciences at Brown Medical School, who implanted the device, described the results as "spectacular" and "almost unbelievable."
(v15) "Here we have a **research** participant who is capable of controlling his environment by thought alone -- something we have only found in science fiction so far," said Friehs.
(v16) "I hope that the trial will continue as successfully as it has started and that all other candidates will have as great an experience as our first candidate did."

Figure 1. Sample news document from www.cnn.com

with weight above a threshold. Also notice that the edge weights are query-independent, so they can be precomputed.

Example 1. Figure 2 shows the document graph for the document of Figure.1. The document is first split to text fragments v0…v16, which correspond to its paragraphs (other delimiters are possible as we explain below). Notice that the edge between nodes v8 and v7 has the highest weight because there are many infrequent (and hence with higher idf value) words that are common between them like "Donoghue" and "BrainGate". □

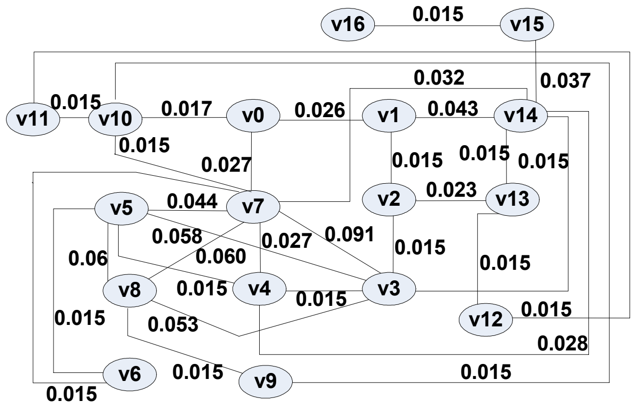


Figure 2. Document Graph for the document in Figure 1.

At query time, the precomputed document graph of a document is processed as follows to create the best query-specific summary. First, each node of the document graph is assigned a score according to the relevance of the corresponding text fragment to the query. To do so we employ traditional IR ranking functions. Notice that a full-text index is used to accelerate this step. Then, we execute our keyword proximity algorithms, which are inspired by the techniques developed in previous work on proximity search on graphs [7], where approximation algorithms are presented for the Group Steiner Tree problem (which is equivalent[1] to the proximity search problem). The best summary is the top-ranked spanning tree that contains all the keywords. The ranking considers both the node and the edge weights (which are query-dependent and independent respectively). Notice that the problem can be easily modified to allow summaries that do not contain all keywords, although this case is not further discussed in this paper.

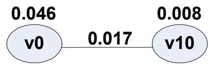
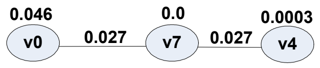
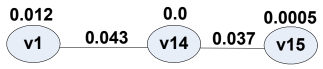
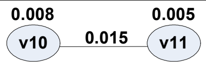
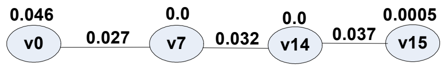
Example 1 (cont'd). Table 1 shows the top-ranked spanning trees for the document graph of Figure 2 for the query "Brain chip research". The values shown above the nodes in Table 1 indicate the node scores with respect to the query. The scores of the spanning trees are a function of their node and edge scores, as explained in Section 4. Notice that all results contain all query keywords. The top result is the best summary of the document of Figure 1 (the keywords of the query are shown in bold) for this query. Intuitively, this result is the best because it contains the minimum possible number of nodes and the edge that connects the two nodes is strong.

Also observe that Result #4 is ranked lower than Result #3 even though it has fewer nodes. The reason is that the nodes of Result #4 are connected through very commonly occurring words

[1] The proximity problem is slightly harder since the sets of nodes do not have to be disjoint.

like “computer” and “brain” whereas in Result #3 they are connected through infrequent words like “Friehs”. Notice that to compute the frequency of a keyword we consider all documents of the corpus. □

Table 1. Top-5 summaries for query “Brain Chip Research”

Rank	Score	Summary
1	67.74	
2	84.77	
3	87.64	
4	103.77	
5	167.41	

The contributions of this paper are the following:

- We present a framework to add structure to text documents, which is used for summarization purposes in this work, although it can be leveraged for other problems as well, like ranking of query results.
- We show how we can use the generated document graphs to create high-quality query-specific summaries. We performed two user surveys to compare the quality of our approach to other current approaches-Desktop search engines and DUC peers[2].
- We present and experimentally evaluate execution algorithms that prove the feasibility of our approach.
- We built a prototype of the system, which is available on the Web at *http://dbir.cs.fiu.edu/summarization*.

The paper is organized as follows. Section 2 presents the related work. Section 3 formally defines the problem, while Section 4 explains how we add structure to documents. Section 5 presents the various algorithms for efficient summary computation using the document graphs. Sections 6 and 7 present the quality and performance experiments respectively. Section 8 describes the developed prototype. Finally Section 9 discusses our conclusions and future work.

2. RELATED WORK

2.1 Document Summarization.

A large corpus of work has focused on generating query-independent summaries [6,3,16,5]. The OCELOT system [6] provides the summary of a web page by selecting and arranging the most (query-independent) “important” words of the page. OCELOT uses probabilistic models to guide the selection and ordering of words into a summary. Amitay and Paris [3] propose a new fully automatic pseudo-summarization technique for Web pages, where the anchor text of hyperlinked pages is used to construct summaries. This approach is unique since it ignores the actual content of a document. [5] uses lexical chains for text summarization. In particular, they use Wordnet to create all lexical chains and choose the strongest ones as a summary of the document.

The majority of systems participating in the past Document Understanding Conference [11] (a large scale summarization evaluation effort sponsored by the United States government), and the Text Summarization Challenge [14] are extraction based. Extraction-based automatic text summarization systems extract parts of original documents and output the results as summaries [10,12,16,20,25]. Other systems based on information extraction [35,47] and discourse analysis [30,42] also exist but they are not yet usable for general-domain summarization. However these works do not exploit the inherent structure of the document and mostly focus on query-independent summaries. In this work (a preliminary version appears in [44]) we also show the semantic connections between the extracted fragments, which improve the quality as shown in Section 6.

[29,31] use natural language processing techniques to create summaries for documents, which cannot scale to large corpora like the Web and are limited to the writing style of the page authors.

White et al. [45,46], Tombros and Sanderson [43], Zechner [47] and Goldstein et al. [16] create query-dependent summaries using a sentence extraction model in which the documents (web pages) are broken up into their component sentences and scored according to factors such as their position, the words they contain, and the proportion of query terms they contain. A number of the highest-scoring sentences are then chosen as the summary. Lin [27] compresses the sentences to achieve better summaries. [1,19,38,39] select the best passage of a document as its summary. However, these works ignore possible semantic connections between the sentences or the possibility that linking a relevant set of text fragments will provide a better summary. Radev et al. [36] provide a technique for multi-document summarization used to cluster the results of a web keyword query. However, their clustering and summarization techniques are query-independent in contrast to our work. [13,32] provide a technique to rank sentences based on their similarity with other sentences across multiple documents and then provide a summary with the top ranked sentences. However, their methods are query-independent in contrast to our work.

The idea of splitting a Web page to fragments has been used by Cai et al. [9], Lee et al. [26] and Song et al. [41], where they extract query-independent rankings for the fragments, for the purpose of improving the performance of web search and also to facilitate web mining and accessibility. Cai et al. [9] partition a web page into blocks using the vision-based page segmentation algorithm. Based on block-level link analysis, they proposed two new algorithms, Block Level PageRank and Block Level HITS to extract authoritative parts of a page. Lee et al. [26] discuss a Web block classification algorithm after Web page division into semantic blocks, while Song et al.[41] provide learning algorithms for block importance.

Finally, all major Web search engines (Google, Yahoo!, MSN Search, and so on) generate query-specific snippets of the

[2] http://duc.nist.gov/

returned results. Although their algorithms are not published, we observed that they simply extract some of the query keywords and their surrounding words. Recently, some of these companies made available tools to provide the same search and snippet functionality on a user's desktop [17,33]. We include these snippets in our user study of Section 6.

2.2 IR Ranking.

In creating the document graph and computing the node weights, we adopt ranking principles from the Information Retrieval community. Various methods for weighting terms have been developed [40]. The most widely used are the Okapi (Equation 1) and the pivoted normalization weighting, which are based on the *tf-idf* principle.

$$\sum_{t \in Q,d} \ln \frac{N - df + 0.5}{df + 0.5} \cdot \frac{(k_1 + 1)tf}{(k_1(1-b) + b\frac{dl}{avdl}) + tf} \cdot \frac{(k_3 + 1)qtf}{k_3 + qtf} \qquad \textbf{(1)}$$

tf is the term's frequency in document,
qtf is the term's frequency in query,
N is the total number of documents in the collection,
df is the number of documents that contain the term,
dl is the document length (in words),
avdl is the average document length and
k1 (between 1.0–2.0), b (usually 0.75), and k3 (between 0–1000) are constants

2.3 Keyword search in data graphs.

In the second stage of our approach, when the document graphs are already created and a query arrives, the system searches the document graphs for sub-trees that contain all query keywords. This problem has been studied by the database and graph-algorithms communities. In particular, recent work [15,7,2,22,21, 24,18] has addressed the problem of free-form keyword search on structured and semi-structured data. These works follow various techniques to overcome the NP-completeness of the Group Steiner problem, to which the keyword proximity search problems can be reduced.

Goldman et al. [15] use precomputation to minimize the runtime cost. BANKS [7] views the database as a graph and proposes algorithms to approximate the Group Steiner Tree problem. We consider and experimentally evaluate modifications of these algorithms in this work. XRANK [18] works on XML trees, which simplifies the problem. Li et al. [28] tackle the problem of proximity search on the Web, which is viewed as a graph of hyperlinked pages. They use of the concept of *information unit*, which can be viewed as a logical Web document consisting of multiple *physical* pages. [2,22,21] perform keyword search on relational databases and exploit the schema properties to achieve efficient execution.

Finally, notice that Buneman et al. [8] view the problem of adding structure to unstructured data from a completely different angle: how to define a schema to describe a labeled graph (e.g., an XML document).

3. PROBLEM DEFINITION

Let $D=\{d_1,d_2,,...,d_n\}$ be a set of documents $d_1,d_2,,...,d_n$. Also let $size(d_i)$ be the length of d_i in number of words. *Term frequency* $tf(d,w)$ of term (word) w in document d is the number of occurrences of w in d. *Inverse document frequency* $idf(w)$ is the inverse of the number of documents containing term w in them.

A *keyword query* Q is a set of keywords $Q=\{w_1,...,w_m\}$. The result of the keyword query, which is not the focus of this work, is a list of documents from D ranked according to their relevance to Q. A key component in our work is the *document graph* $G(V,E)$ of a document d, which is defined as follows:

Definition 1 (Document Graph). The *document graph* $G(V,E)$ of a document d is defined as follows:

- d is split to a set of non-overlapping text fragments $t(v)$, each corresponding to a node $v \in V$.
- An edge $e(u,v) \in E$ is added between nodes $u,v \in V$ if there is an association (further discussed in Section 4) between $t(u)$ and $t(v)$ in d. □

Hence, we can view G as an equivalent representation of d, where the associations between text fragments of d are depicted. Example 1 explains a possible document graph for the document of Figure 1. Notice that there are many ways to define the document graph for a document. In this work we follow a semantic approach where a delimiter is chosen to create text fragments, and edges are added when the text fragments contain common (or equivalent) words as we explain in Section 4.

Furthermore, notice that the nodes and edges of the document graph may be weighed according to a variety of reasons, both query-dependent and independent. For example, authority flow techniques [4] on the document graph can be employed to assign both query-dependent and independent scores. In this work (see Section 4) we consider query-dependent (resp. independent) weights for the nodes (resp. edges).

Definition 2 (Minimal Total Spanning Tree). Given a document graph $G(V,E)$, a *minimal total spanning tree* of G with respect to a keyword query $Q=\{w_1,...,w_m\}$ is a sub-tree T of G that is both:

- *Total*: every keyword $w \in Q$ is contained in at least one node of T.
- *Minimal*: we cannot remove any node from T and still have a total sub-tree. □

A *summary* of a document d with document graph G, with respect to a keyword query $Q=\{w_1,...,w_m\}$, is a minimal total spanning tree of G for Q.

Problem 1 (Summarization). Given a document $d \in D$ and its document graph G, and a keyword query Q, find the top summary, i.e., the minimum score minimal total spanning tree.

Notice that the totality property implies that we use *AND* semantics, that is, require all keywords to be in the summary. Another alternative is *OR* semantics where not all keywords are required to be in the summary. *OR* semantics are useful in the following scenarios: (a) the keywords are rare and hence no document contains all of them, so to summarize the query result we need *OR* semantics and (b) in order to have more compact summaries we may choose to not display the less important keywords. In this paper we only present our results on *AND* semantics due to space limitations. Our techniques and algorithms can be extended to generate summaries for *OR* semantics, by relaxing the totality constraint on summary spanning trees.

Furthermore, the fact that the summaries are minimal means that we do not allow any nodes not containing any keyword as leaf nodes in the summary tree. However, nodes with no keywords can be internal nodes. For example, in the second result of Table 1, node *v7* has no keywords, but it acts as a connector between nodes *v0* and *v4* which contain the keywords.

The score of a (summary) tree *T* is calculated using a scoring function based on the weights of the nodes and edges of *T*. The scoring function used in this work is presented in Section 4.

Example 1 (cont'd). For the document of Figure 1 and the keyword query "Brain Chip Research", the top summary is shown in Figure 3.

Brain chip offers hope for paralyzed.
⌊Donoghue's initial **research** published in the science journal Nature in 2002 consisted of attaching an implant to a monkey's **brain** that enabled it to play a simple pinball computer game remotely.

Figure 3. Top summary of the document of Figure 1 for query "Brain Chip Research"

Notice that by the definition of Problem 1, a summary may contain internal text fragments with no query keywords, which are called Steiner nodes. The reason we include such nodes is to achieve semantic coherence in the generated summaries, which increases the user satisfaction as we show in Section 6. If brevity is the top priority then Steiner nodes can be omitted.

4. ADDING STRUCTURE TO DOCUMENTS

As we explain below, there are many ways to create and assign weights to a document graph. In this section we present the specific approach we follow to create a document graph. In particular, given a document $d\in D$, a query *Q* and a set of input parameters (explained below), we construct a document graph *G(V,E)*. Notice that *Q* is only used in assigning weights to the nodes of *G*, which is a desirable property since the rest of *G* can be computed before queries arrive.

The following input parameters are required at the precomputation stage to create the document graph:

1. *Threshold for edge weights*. Only edges with weight not below *threshold* will be created in the document graph.
2. *Parsing Delimiter*. The parsing delimiter is used to split the document to text fragments. Typical choices are the new-line character (each text fragment corresponds to a paragraph) or the period (each text fragment is a sentence). We found that for the domain of news articles that we experimented with (see Section 6) the new-line is preferable since paragraphs are typically short and leads to more compact document graphs.

Example 1 (cont'd) The new-line character was used to parse the document of Figure 1 into 17 text fragments *v0,…,v16*. □

After parsing the document and creating the graph nodes (text fragments), for each pair of nodes *u,v* we compute the association degree between them, that is, the score (weight) *EScore(e)* of the edge *e(u,v)*. If *EScore(e)≥threshold*, then *e* is added to *E*. The score of edge *e(u,v)* where nodes *u*, *v* have text fragments *t(u)*, *t(v)* respectively is:

$$EScore(e)=\frac{\sum_{w\in(t(u)\cap t(v))}\left(\left(tf(t(u),w)+tf(t(v),w)\right)\cdot idf(w)\right)}{size(t(u))+size(t(v))} \qquad \textbf{(2)}$$

where *tf(d,w)* is the number of occurrences of *w* in *d, idf(w)* is the inverse of the number of documents containing *w*, and *size(d)* is the size of the document (in words).

That is, for every word *w* appearing in both text fragments we add a quantity equal to the *tf·idf* score of *w*. Notice that stop words are ignored. Furthermore, we use thesaurus and stemmer (we rely on Oracle interMedia as explained in Section 7) to match words that are equivalent. The sum is divided by the sum of the lengths of the text fragments in the same way as the document length (*dl*) is used in traditional IR formulas. Notice that Equation 2 is an adaptation of traditional IR formulas for a pair of documents.

Notice that alternative ways to compute the edge weights are possible, like the cosine document distance, which however have similar effect as the *tf·idf* method that we employ. In future versions of our system we plan to also use Wordnet and Latent Semantic Indexing techniques to improve the quality of the edge weights, which is challenging on the performance level since our system is interactive.

The calculation of the edge weights concludes the query-independent part of the document graph creation. Next, when a query *Q* arrives, the nodes in *V* are assigned query-dependent weights according to their relevance to *Q*. In particular, we assign to each node *v* corresponding to a text fragment *t(v)* node score *NScore(v)* defined by the Okapi formula (Equation 1). In order to accelerate this step of assigning node scores we built a full-text index on the set *D* of documents that efficiently allows locating the nodes that contain the query keywords and also calculate the query-dependent score. The details of this index are out of the scope of this paper.

4.1 Summary Scoring Function

Given the document graph G and a query Q, a summary (subtree of G) T is assigned a score Score(T) by combining the scores of the nodes v∈T and the edges e∈T. In particular Equation 3 computes the summary score.

$$\text{Score(T)} = a\sum_{edge\ e\in T}\frac{1}{EScore(e)}+b\frac{1}{\sum_{node\ v\in T}NScore(v)} \qquad \textbf{(3)}$$

where a and b are constants (we use a=1 and b=0.5), *EScore(e)* is the score of edge e using Equation 2, *NScore(v)* is the score of node v using Equation 1.

Intuitively, if T is larger (has more edges) then its score should degrade (increase) since larger trees denote looser semantic connections [23,2,22,7]. This is the reason we take the sum of the inverse of the edge scores in Equation 3. Furthermore, if more nodes of T are relevant to Q, the score should improve (decrease). Hence, we take the inverse of the sum of the node scores.

Constants a and b are used to calibrate the importance of the size of the summary (in number of edges) versus the amount of relevant information contained. In particular, higher a values boost the score of smaller and tightly connected summaries, whereas higher b values benefit summaries with more relevant content (i.e., containing nodes with high score with respect to the query). Notice that a and b can also be viewed as adjusting parameters for the query-independent and dependent parts of the scoring function respectively.

Example 1 (cont'd). The top summary T for the document of Figure 1 with document graph shown in Figure 2 is shown in Figure 4. T has a single edge e*(v0,v10)* with score determined by the common word "brain" between *v0* and *v10*. Also, the scores of nodes *v0*, *v10* are computed using Equation 1, for the query "Brain chip research". □

5. EFFICIENT SUMMARY COMPUTATION

This section tackles the problem of how; given the document graph G of a document d for a query Q, to compute the top summary (or summaries) for d with respect to Q. For clarity, we only present algorithms for *AND* semantics. Notice that the problem of finding the top summary (total minimal spanning tree) is very similar to the Group Steiner Tree problem [37], which is known to be *NP-complete*. Our problem is slightly more complex since the groups of nodes are not disjoint, in contrast to the Group Steiner Problem, which is defined as follows:

Given an undirected, connected, and weighted graph $G=(V, E, l)$, where V is the set of all vertices in G, E is the set of edges in G, and l is a weight function which maps each edge $e \in E$ to a non-negative real number; and given a family $R=\{R_1,....R_k\}$ of disjoint groups of vertices, where R_i is a subset of V, the problem is to find a minimum-cost tree T which contains at least one vertex from each group R_i. Since the weights of the graph are non-negative, the solution is a tree-structure.

In contrast to previous work on proximity search on data graphs (see Section 2) where the top-k [21,7] or all [2,22] total minimal spanning trees are requested, in our summarization problem we typically care for only the single top summary, that is, the top-1 total minimal spanning tree. This allows more efficient algorithms as we explain below. Notice that the presented algorithms, which can be viewed as approximations of the Group Steiner Tree problem, can be divided along two dimensions. First, we have multi-result and top-1 algorithms, which compute a set of summaries or a single summary for a document and query pair respectively. Second, we have enumeration and expanding algorithms, which follow different execution approaches as explained below.

Precomputation. In order to boost the performance of the algorithms, we precompute and store the following information:

- A full-text index is built, as discussed above, to efficiently locate the nodes that contain the keywords and calculate their query-specific score.
- The all-pairs shortest paths between the nodes of the document graph G of every document d. That is, for each pair of nodes $u,v \in G$, we precompute and store the shortest path $u \sim u_1 \sim ... \sim u_r \sim v$.

5.1 Multi-Result Enumeration Algorithm

This algorithm returns a ranked list of summaries for a document and a query. In particular, it returns a summary for each possible combination of nodes that contain the keywords.

The algorithm (Figure 4) proceeds as follows. First, we find all combinations of nodes in G that are minimal (no node is redundant) and total (contain all keywords in Q). Then, for each combination we create a complete graph G_c (called *closure graph*) that contains all nodes in the combination and all-pairs edges between them with weight equal to their distance (taken by the precomputed all-pairs shortest paths). Then, we calculate all possible spanning trees in G_c, and compute their scores using Equation 3. Then, for the top spanning tree we insert the Steiner nodes and trim redundant nodes to make it minimal. Then its accurate score is computed and added to the results list. Finally, the results are ranked and displayed.

```
MultiResultEnumeration (document graph G, query Q)
1. Results ←∅; /*stores summaries*/
2. Find all nodes in G that contain some keyword of Q; /*use
   full-text index*/
3. Find all minimal combinations of nodes that when taken
   together contain all keywords in Q;
4. For each minimal node combination C do
   {
5.   Create closure graph $G_c$ that contains only the nodes in C;
6.   Find all possible spanning trees S of $G_c$;
7.   Calculate the score of each spanning tree in S using Eq. 3;
8.   Pick the spanning tree T with the minimum score;
9.   Replace the edges u~v in T with their precomputed shortest
     paths $u \sim u_1 \sim ... \sim u_k \sim v$; /* i.e., we are adding the Steiner
     nodes.*/
10.  Trim T to make it a minimal total spanning tree;
11.  Calculate the score of T using Equation 3 and add T to
     Results;
12.  Sort and output summaries in Results;
   }
```

Figure 4. Multi-Result Enumeration Algorithm

Example 2. Consider the document graph in Figure 2 and the query "Brain chip research". The nodes that contain the keywords are *v0, v1, v3, v4, v10, v11,* and *v15*. We then find all minimal and total node combinations, which are *{v0, v10}*, *{v15, v0}*, *{v0, v3}*, *{v4, v0},* and so on. For each combination we create a closure graph. For example, the closure graph for the second combination is *v15~v0* with edge weight 0.096 (which is the length of the shortest path from *v15* to *v0*). We then find all possible spanning trees of this graph, which is just *v15~v0,* for the above closure graph. Then, we replace the edge between *v15* and *v0* with the shortest path between them, which is *v15~v14~v7~v0.* This tree is already minimal and hence we output this result along with its score. The Steiner nodes in this result are *v14* and *v7*, which don't have any keywords in them but are used to relate the other 2 nodes, *v15* and *v0*. □

5.2 Top-1 Enumeration Algorithm

The Top-1 enumeration algorithm returns only one summary per document, for a query. The reason we created top-1 variants for both the enumeration and the expanding search (Sections 5.3, 5.4) is that typically the user only requests a single summary for a document, as in the case of snippets in Web search engine results.

This algorithm is similar but more efficient than the multi-result enumeration algorithm, because it only adds the Steiner nodes (line 9 in Figure 4) for a single spanning tree. In particular, this algorithm finds the top spanning tree among all node combinations and then substitutes the Steiner nodes, while the multi-result algorithm finds the top spanning tree and substitutes the Steiner nodes for each node combination. The pseudo-code for this algorithm has the following difference with respect to Figure 4: Lines 8-11 are moved outside the for-loop, that is, the for-loops ends at line 7.

Example 2 (cont'd). For the document graph in Figure 2 and query "Brain chip research", this algorithm goes through the same steps as in the case of enumeration algorithm. It computes all node combinations as explained in the previous example. The only difference is that this algorithm first finds the minimum-score spanning tree *v1~v3* with edge weight 0.03 (which is the length of the shortest path from *v1* to *v3*) among all spanning trees of all node combinations, and then replaces that edge with the shortest path *v1~v2~v3*, where *v2* is the Steiner node and recomputes the score and displays it as the summary of the document.

5.3 Multi-Result Expanding Search Algorithm

The basic idea behind this algorithm (inspired by the algorithm in BANKS [7]) is that we start from the nodes that contain the query keywords and progressively expand them in parallel until we find all minimal total spanning trees. The advantage of this algorithm compared to the enumeration algorithms is that we do not need to repeat the processing for all combinations of nodes, which may be too many if the document is large and contains many occurrences of the query keywords.

In particular, the algorithm (Figure 5) finds (using the full-text index) all the nodes that match some keywords in the query and starts expanding them incrementally, the best (maximum-score) edge at a time. We call the subgraph created from each keyword node *v expanding area* of *v*. Notice that, in contrast to BANKS, we use the precomputed all-pairs shortest paths data to efficiently grow the expanding area. That is, we only consider the edges that are contained in a shortest path from the current node *v* to any other node *u* that contains additional query keywords than *v*. When two or more expanding areas meet we check for possible new summaries. If a summary is found, it is trimmed to become minimal and its score is calculated using Equation 3. The parallel expansion of the expanding areas terminated when for each combination of nodes that contains all keywords, their expanding areas have met.

Example 3. For the document graph in Figure 2 and the query "Brain chip research", the keyword nodes are *v0, v1, v3, v4, v10, v11,* and *v15*. We grow the expanding area of *v0* to *v0~v10,* which is the first precomputed single source shortest path of source *v0* and check for possible summaries. *v0~v10* is total as well as minimal and hence we add it to the set of results. We grow each expanding area using its precomputed shortest paths. Then we grow *v1* to *v1~v2, v3* to *v3~v2, v4* to *v4~v3*, *v10* to *v10~v9*, *v11* to *v11~v10* and once we expand *v11* we have another summary *v11~v10* that is total and minimal. We keep doing this until the expanding areas of all the keywords nodes have been met and hence we can't have any more possible summaries and hence we terminate. □

```
MultiResultExpandingSearch(document graph G, query Q)
1. Results ← ∅; /*stores summaries*/
2. Find all nodes $N=\{N_1,\ldots,N_m\}$ that contain the keywords in
   Q; /*$N_i$ has the nodes that contain $w_i$*/
3. Repeat until the expanding areas of all combinations of
   nodes in $N_1,\ldots,N_m$ meet. {
4.   For each node v in N do
     {
5.     Add to the expanding area of v the maximum-score
       adjacent edge from the (precomputed) shortest paths
       starting at v and ending at a node in N not containing the
       same keywords as v;
6.     Check for new results (summaries) T; /*i.e., trees that
       contain a node from each of $N_1,\ldots,N_m$ */
7.     Trim summaries T to become minimal;
8.     Calculate the score of T using Equation 3 and store in
       Results;
9.     Sort and output summaries in Results;
     } }
```

Figure 5. Multi-Result Expanding Search Algorithm

5.4 Top-1 Expanding Search Algorithm

This algorithm differs from the multi-result expanding search algorithm in that it stops expanding the expanding areas once the first summary is produced. Intuitively this greedy approach produces a high-quality summary, as the trees produced first have smaller sizes, which implies smaller scores (Equation 3). The pseudo code for the Top-1 expanding search algorithm differs from the multi-result variant in that, once it finds a summary in line 6 it trims it, calculates the score and adds it to results and exits the loop. So we have an extra line in Figure 5: "7a. break;".

Example 3 (cont'd). For the document graph in Figure 2 and the query "Brain chip research", this algorithm goes through the same steps as its multi-result variant, but stops expanding once it finds the first summary, which is *v10~v11* as explained in the previous example.

6. QUALITY EXPERIMENTS

To evaluate the quality of the results of our approach, we conducted two surveys. The subjects of the survey are fifteen students (of all levels and various majors) of FIU, who were not involved in the project. In this survey the users were asked to evaluate the summaries based on their quality and size (a longer summary carries more information but is less desirable).Each participant was asked to compare the summaries and rank them, assigning a score of 1 to 5, according to their quality for the corresponding query. A rank of 5 (1) represents the summary that is most (least) descriptive.

6.1 Comparison with DUC dataset

The dataset used in this survey consists of ten documents and two queries taken from the DUC 2005 dataset[3], as shown in Table 2. We compare our summaries with DUC Peer summaries for quality. DUC peers are human and automatic summaries used in quality evaluation. We compared our summaries against the DUC peers with highest linguistic quality. Unfortunately, most of the summaries in the DUC datasets are query-independent and the few query-dependent ones are multi-document. Hence, in order to compare our work to that of DUC we used the following method to extract single-document summaries from query-dependent multi-document summaries for a set of ten documents over two topics. The sentences that have been extracted from a document *d* to construct the multi-document summary are viewed as *d*'s single-document summary for the query/topic. Notice that the DUC summaries are created by extracting whole sentences from documents.

Table 2. Average summary ratings for DUC topics

Query 1 (*International Organized Crime*) DUC Topic ID: d301i			Query 2 (*Women in Parliaments*) DUC Topic ID: d321f		
Doc. ID	DUC Peer	Top-1 Expanding	Doc. ID	DUC Peer	Top-1 Expanding
FT941-3237	2.33	4.66	FT921-7786	4.00	2.50
FT944-8297	2.50	3.33	FT922-190	2.00	4.00
FT931-3563	2.83	3.00	FT921-937	2.00	4.33
FT943-16477	4.00	4.17	FT922-13353	2.83	4.17
FT943-16238	3.67	3.67	FT921-74	2.33	3.67

The results of the survey prove the superiority of our approach, as shown in Table 2. Our method of combining extracted sentences using semantic connections in the form of Steiner trees leads to higher user satisfaction than the traditional sentence extraction methods. In particular, the Steiner sentences in summaries provide coherency in the aggregation of the keyword-containing-sentences.

6.2 Comparison with Google and MSN Desktop

The dataset used in this survey consists of two news documents taken from the technology section of cnn.com. The participants were asked to evaluate the quality of the summaries of the two documents with respect to five queries. We chose queries where keywords appear both close and far from each other. For each query-document pair, three summaries are displayed corresponding to (a) the result of the Top-1 expanding search algorithm, (b) Google Desktop's summary, and (c) MSN Desktop's summary. Summaries (b) and (c) were created by indexing the two documents in our desktop and then submitting the five queries to the Desktop engines. The summaries are the snippets output for these documents. In order to compare apples to apples, we chose queries for which the length of the summaries produced by all three methods are similar, since clearly it is not fair to compare summaries of different lengths as some people favor conciseness while others the amount of information.

In this survey we set constant *a* to 1 and *b* to 0.5 in Equation 3, which we found to produce higher-quality summaries. Notice that by increasing the value of constant *a*, we favor short results, while by increasing constant *b* we favor longer and more informative results. Hence, by setting *a* to 1 and *b* to 0.5 we favor shorter summaries, which have similar size to the ones produced by Google and MSN Desktop. This makes their comparison fairer.

Table 3. Average summary ratings for documents *D1* and *D2*

Queries	Google Desktop		MSN Desktop		Top-1 Expanding	
	D1	*D2*	*D1*	*D2*	*D1*	*D2*
1	2.33	3.67	2.33	3.67	4.87	3.67
2	2.00	3.33	2.00	3.00	4.33	3.33
3	3.00	2.67	0.67	3.00	4.93	4.00
4	1.67	2.67	1.67	3.00	4.67	4.00
5	2.00	1.67	3.00	1.00	4.00	3.67

Table 4. Queries used for documents *D1* and *D2*

Queries	Document *D1*	Document *D2*
1	Microsoft worm protection	IT Research awards
2	Anti-virus protection	Algorithms development research
3	Recovering worm deleted files	Software projects
4	Worm affected agencies	Large research grants
5	Deleted computer software	Computer network security project

The results of the survey, which prove the superiority of our approach, are shown in Table 3. Notice that Google and MSN Desktop systems do not always include all keywords in the summary when they are more than two and have big distances

[3] http://duc.nist.gov/

between them. In contrast, our approach always finds a meaningful way to connect them.

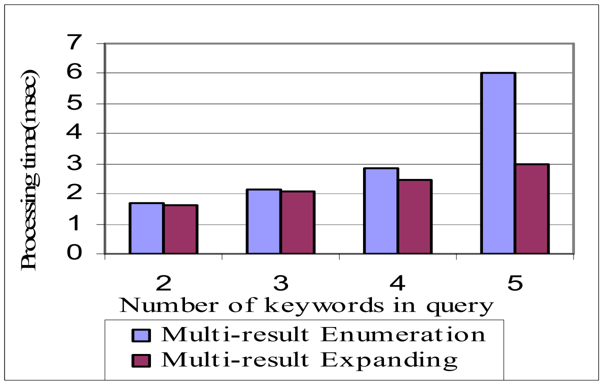
7. PERFORMANCE EXPERIMENTS

To evaluate the performance of our approach we used a dataset of 200 news documents taken from the technology section of cnn.com. We used a PC with Pentium 4 2.44GHz processor and 256MB of RAM running Windows XP. The algorithms were implemented in Java. To build the full-text index we used Oracle interMedia [34] and stored the documents in the database. JDBC was used to connect to the database system.

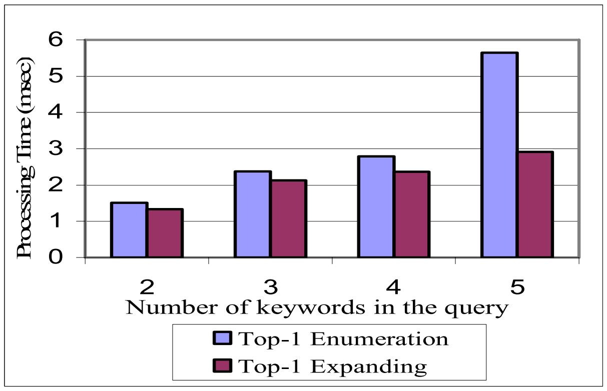
First, we compare the performance of the four algorithms of Section 5 for summarizing keyword queries of various lengths. The execution times consist of two parts: (a) the computation of the scores of the nodes of the document graph (remember that this is query-specific and cannot be precomputed), and (b) the generation of the top summaries (minimal total spanning trees) in the document graph. The first part is handled by Oracle interMedia and the average times for a single document for various-length queries are shown in Table 5.

Table 5. Average times to calculate node weights

Number of keywords	2	3	4	5
Time (msec)	5.31	9.37	11.50	17.33



(a) Multi-Result algorithms



(b) Top-1 algorithms

Figure 6. Processing times

The second part of the execution is handled separately by the four algorithms and the results are shown in Figures 6 (a) and (b). In particular, Figure 6 (a) compares the performance of the Multi-Result algorithms, whereas Figure 6 (b) the Top-1 algorithms. We observe that the expanding search algorithms are faster than the enumeration ones, especially for long queries. Also, notice that there is only a slight difference in the performance of the Top-1 and the Multi-Result algorithms, because the document graphs are relatively small and hence there is no big difference between computing one or more summaries.

Notice that we do not compare the performance of our algorithms to BANKS, since our Multi-Result algorithms are adaptations of the BANKS algorithms to our problem, which is different as we explain in Section 5.

Finally, we measure the accuracy of the Top-1 versions of the algorithms. In particular, we measure (Table 6) the average rank of the summary of the Top-1 algorithms in the list of summaries created by the Multi-Result algorithms. For example, if the summary of the Top-1 algorithm appears as the third summary of the Multi-Result algorithm, then the rank is 3. We observe that the Top-1 expanding algorithm better approximates the corresponding Multi-Result algorithm's results.

Table 6. Average ranks of Top-1 Algorithms with respect to Multi-Result algorithms

Number of keywords	2	3	4	5
Top-1 Enumeration Algorithm.	1.4	1.8	2.1	2.78
Top-1 Expanding Search Algorithm.	1.1	1.3	1.4	1.8

8. CONCLUSIONS AND FUTURE WORK

In this work we presented a structure-based technique to create query-specific summaries for text documents. In particular, we first create the document graph of a document to represent the hidden semantic structure of the document and then perform keyword proximity search on this graph. We show with a user survey that our approach performs better than other state of the art approaches. Furthermore, we show the feasibility of our approach with a performance evaluation.

In the future, we plan to extend our work to account for links between documents of the dataset. For example, exploit hyperlinks in providing summarization on the Web. Furthermore, we are investigating how the document graph can be used to rank documents with respect to keyword queries. Finally, we plan to work on more elaborate techniques to split a document to text fragments and assign weights on the edges of the document graph.

9. REFERENCES

[1] J.Abracos and G. Pereira-Lopes. Statistical methods for retrieving most significant paragraphs in newspaper articles. In ACL/EACL Workshop on Intelligent Scalable Text Summarization, 1997

[2] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A System For Keyword-Based Search Over Relational Databases. ICDE, 2002

[3] E. Amitay, C. Paris: Automatically Summarizing Web Sites -Is there any way around it? CIKM, 2000

[4] A. Balmin, V. Hristidis, Y. Papakonstantinou: Authority-Based Keyword Queries in Databases using ObjectRank. VLDB, 2004

[5] R. Barzilay and M. Elhadad: Using lexical chains for text summarization. ISTS, 1997

[6] A. L. Berger and V. O. Mittal, OCELOT: A System for summarizing web pages. SIGIR, 2000

[7] G. Bhalotia, C. Nakhe, A. Hulgeri, S. Chakrabarti and S,Sudarshan: Keyword Searching and Browsing in Databases using BANKS. ICDE, 2002

[8] P. Buneman, S. Davidson, M. Fernandez, D. Suciu "Adding Structure to Unstructured Data". ICDM, 2003

[9] D. Cai, X. He, J.Wen, W.Ma: Block-level Link Analysis. SIGIR, 2004

[10] H.H. Chen, J.J. Kuo, and T.C. Su: Clustering and Visualization in a Multi-Lingual Multi- Document Summarization System. ECIR, 2003

[11] Document Understanding Conference http://duc.nist.gov, 2002

[12] H.P. Edmundson: New Methods in Automatic Abstracting. ACM Journal, 1969

[13] G. Erkan and D.R. Radev. Lexrank: Graph-based centrality as salience in text summarization. JAIR, 2004

[14] T. Fukusima and M. Okumura: Text Summarization Challenge Text Summarization Evaluation in Japan. WAS, 2001

[15] R. Goldman, N. Shivakumar, S. Venkatasubramanian, H. Garcia-Molina: Proximity Search in Databases. VLDB, 1998

[16] J. Goldstein, M. Kantrowitz, V. Mittal, J. Carbonell: Summarizing text documents: Sentence selection and evaluation metrics. ACM SIGIR, 1999

[17] Google Desktop search http://desktop.google.com/

[18] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. ACM SIGMOD, 2003

[19] M.A. Hearst . Using categories to provide context for full-text retrieval results. In Proceedings of the RIAO, 1994

[20] E. Hovy and C.Y. Lin: The automated acquisition of topic signatures for text summarization. ICCL, 2000

[21] V. Hristidis, L. Gravano, Y. Papakonstantinou: Efficient IR-Style Keyword Search over Relational Databases. VLDB, 2003

[22] V. Hristidis, Y. Papakonstantinou: DISCOVER: Keyword Search in Relational Databases. VLDB, 2002

[23] V. Hristidis, Y. Papakonstantinou, A. Balmin: Keyword Proximity Search on XML Graphs. ICDE, 2003

[24] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, H. Karambelkar. Bidirectional Expansion For Keyword Search on Graph Databases. VLDB, 2005

[25] J. Kupiec, J. Pederson, and F. Chen: A Trainable Document Summarizer, SIGIR, 1995

[26] C.H. Lee, M.Y. Kan, S. Lai: Stylistic and Lexical Co-training for Web Block Classification. WIDM, 2004

[27] C.Y. Lin: Improving Summarization Performance by Sentence Compression - A Pilot Study. IRAL, 2003

[28] W.S. Li, K. S. Candan, Q. Vu, and D. Agrawal: Retrieving and Organizing Web Pages by "Information Unit", WWW, 2001

[29] C. Y. Lin and E. Hovy. Identifying topics by position. In Proceedings of the ACL Conference on Applied Natural Language Processing, 1997

[30] D. Marcu. Discourse trees are good indicators of importance in text. Advances in Automatic Text Summarization, 1999

[31] D. Marcu. The rhetorical parsing of natural language texts. In Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics, 1997

[32] R. Mihalcea, P. Tarau, TextRank: Bringing Order into Texts, EMNLP 2004

[33] MSN Desktop search http://toolbar.msn.com/

[34] Oracle interMedia http://www.oracle.com/technology/products/intermedia, 2005

[35] D.R. Radev and K.R. McKeown: Generating Natural Language Summaries from Multiple On-line Sources. Computational Linguistics, 1998

[36] D.R.Radev, W. Fan, Z. Zhang: WebInEssence: A Personalized Web-Based Multi-Document Summarization and Recommendation System. NAACL Workshop on Automatic Summarization, 2001

[37] P. W. G. Reich. Beyond Steiner's Problem: A VLSI Oriented Generalization. Workshop on Graph-Theoretic Concepts in Computer Science, 1989

[38] G. Salton, A. Singhal, C. Buckley, M. Mitra. Automatic text decomposition using text segments and text themes. Hypertext, 1996

[39] G. Salton , A. Singhal , M. Mitra, and C. Buckley: Automatic text structuring and summarization. Information Processing and Management, 1997

[40] A. Singhal: Modern Information Retrieval: A Brief Overview, Google, IEEE Data Eng. Bull, 2001

[41] R. Song, H. Liu, J. Wen, W. Ma: Learning Block Importance Models for Web Pages. WWW, 2004

[42] T. Strzalkowski, G. Stein, J. Wang, and B, Wise. A Robust Practical Text Summarizer. In I. Mani and M. Maybury (eds), Advances in Automatic Text Summarization, 1999

[43] A. Tombros, M. Sanderson. Advantages of Query Biased Summaries in Information Retrieval. SIGIR 1998

[44] R. Varadarajan, V Hristidis: Structure-Based Query-Specific Document Summarization. Poster paper at CIKM 2005

[45] R. W. White, I. Ruthven and J. M. Jose: Finding Relevant Documents using Top Ranking Sentences: An Evaluation of Two Alternative Schemes, SIGIR, 2002

[46] M. White, T. Korelsky, C. Cardie, V. Ng, D. Pierce, and K. Wagstaff.: Multidocument Summarization via Information Extraction. HLT, 2001

[47] K. Zechner. Fast generation of abstracts from general domain text corpora by extracting relevant sentences. In Proceedings of the International Conference on Computational Linguistics, 1996