

Beyond Single-Page Web Search Results*

Ramakrishna Varadarajan, Vagelis Hristidis and Tao Li

Abstract— Given a user keyword query, current Web search engines return a list of individual web pages ranked by their “goodness” with respect to the query. Thus the basic unit for search and retrieval is an individual page, even though information on a topic is often spread across multiple pages. This degrades the quality of search results especially for long or uncorrelated (multi-topic) queries (in which individual keywords rarely occur together in the same document) where a single page is unlikely to satisfy the user’s information need. We propose a technique that given a keyword query, on-the-fly generates new pages, called *composed pages*, which contain all query keywords. The composed pages are generated by extracting and stitching together relevant pieces from hyperlinked Web pages, and retaining links to the original Web pages. To rank the composed pages we consider both the hyperlink structure of the original pages, as well as the associations between the keywords within each page. Furthermore, we present and experimentally evaluate heuristic algorithms to efficiently generate the top composed pages. The quality of our method is compared to current approaches using user surveys. Finally, we also show how our techniques can be used to perform *query-specific* summarization of web pages.

Index Terms— Internet search, Search process, Web Search

----- ◆ -----

1 INTRODUCTION

Given a user keyword query, current Web search engines return a list of pages ranked by their “goodness” with respect to the query. However, the information for a topic, especially for long or uncorrelated (multi-topic) queries (in which individual query keywords occur relatively frequently in the document collection but rarely occur together in the same document), is often distributed among multiple physical pages connected via hyperlinks [26]. It is often the case that no single page contains all query keywords. Li et al. [26] make a first step towards this problem by returning a tree of hyperlinked pages that collectively contain all query keywords. The limitation of this approach is that it operates at the page-level, which ignores the specific context where the keywords are found in the pages. More importantly it is cumbersome for the user to locate the most desirable tree of pages due to the amount of data in each page and the large number of page trees.

We propose a technique that given a keyword query, on-the-fly generates new pages, called *composed pages*, which contain all query keywords. A preliminary version of this work was presented as a poster paper in SIGIR 2006 [39]. The composed pages are generated by stitching together appropriate pieces from hyperlinked Web pages (hyperlinks to the original Web pages are also displayed).

* The Project is partially supported by NSF Grants IIS-0534530 and IIS-0546280.

-
- R.Varadarajan is with the School of Computing and Information Sciences, Florida International University, University Park, ECS 234, 11200 S.W. 8th Street, Miami, FL 33199. E-mail: ramakrishna@cis.fiu.edu.
 - V. Hristidis is with the School of Computing and Information Sciences, Florida International University, University Park, ECS 384, 11200 S.W. 8th Street, Miami, FL 33199. E-mail: vagelis@cis.fiu.edu.
 - T.Li is with the School of Computing and Information Sciences, Florida International University, University Park, ECS 318, 11200 S.W. 8th Street, Miami FL 33199. E-mail: taoli@cis.fiu.edu

Manuscript received (insert date of submission if desired). Please note that all acknowledgments should be placed at the end of the paper, before the bibliography.

To rank the composed pages we consider both the hyperlink structure of the original (source) pages, as well as the associations between the keywords within each page.

Our technique has the following key steps: During the preprocessing stage, for each web page we create a labeled, weighted graph, called the *page graph*, by splitting the page to a set of *text fragments* (graph nodes) and computing the semantic associations between them (graph edges). Then, at query time, given a set of keywords, we first find a tree, called *web spanning tree*, of hyperlinked pages that collectively contain all the query keywords. Then we perform keyword proximity search on the each page’s page graph to discover how the keywords contained in the page are associated with each other. For each page in the web spanning tree we extract a *page spanning tree* that contains a subset of the query keywords. The page spanning trees of the pages of the web spanning tree are appropriately combined into a composed page, which is returned to the user. As we will explain later, smaller web spanning trees are preferable and hence single-page results, as created by current Web search engines for AND semantics are ranked higher.

Note that a key assumption we make in this paper is that hyperlinked pages are associated to each other. This is a reasonable assumption. Furthermore, each result should be composed of pages associated to each other to have a cohesive meaning. Hence, we only consider hyperlinked pages in building web spanning tree.

Example 1: Figure 1 shows a Web graph extracted from the *www.fiu.edu* Web site. The hyperlinks between pages are depicted in the Web graph as edges. The nodes in the graph represent the Web pages. Figure 2 shows the page graph of Page 1 in Figure 1. As denoted in Figure 1, Page 1 is split into 7 text fragments *v1...v7*, using the newline delimiter, and each one is represented by a node in the page graph. The edges denote semantic

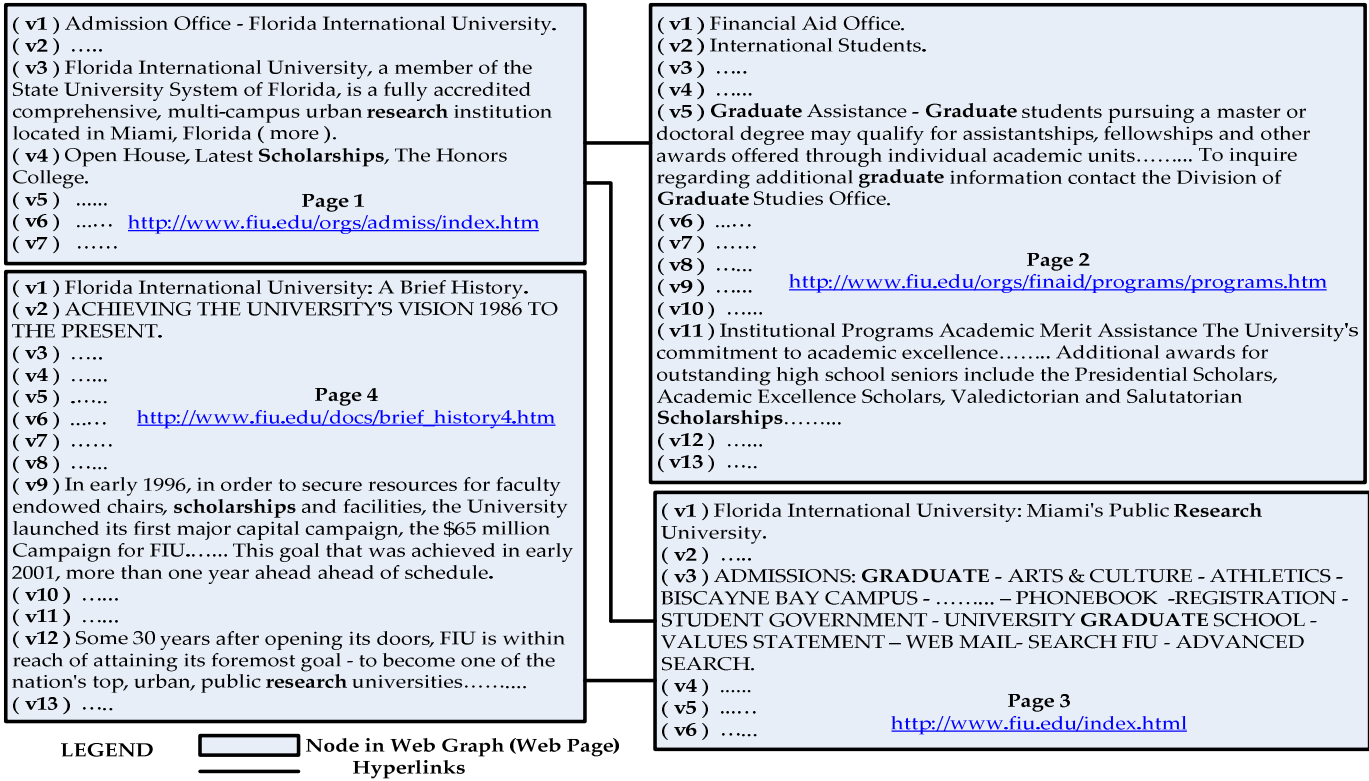


Figure 1. Sample Web pages from www.fiu.edu.

associations. Table 1 shows the Top-3 search results (composed pages) for the query “Graduate Research Scholarships”. We represent the nodes of a web spanning tree using rectangles and the nodes of a page spanning tree using circles. Hyperlinks are solid lines, while the semantic links within in a page graph are dotted lines. The page spanning trees represent the most “relevant pieces” of each page. □

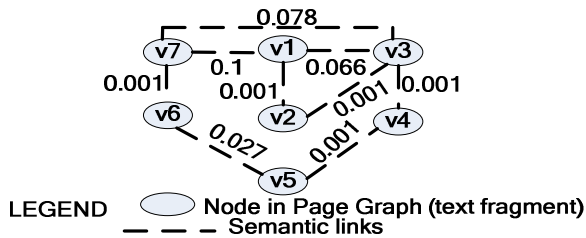


Figure 2. A page graph of Page 1 in Figure 1.

Table 1. Top-3 search results for query “Graduate Research Scholarships”.

Rank	Score	Search Results
1	12.50	<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="border: 1px solid black; padding: 2px;"> <div style="border: 1px solid black; border-radius: 50%; width: 15px; height: 15px; display: flex; align-items: center; justify-content: center;">v3</div> -- <div style="border: 1px solid black; border-radius: 50%; width: 15px; height: 15px; display: flex; align-items: center; justify-content: center;">v4</div> </div> <div style="border: 1px solid black; padding: 2px;"> <div style="border: 1px solid black; border-radius: 50%; width: 15px; height: 15px; display: flex; align-items: center; justify-content: center;">v5</div> </div> </div>
2	101.60	<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="border: 1px solid black; padding: 2px;"> <div style="border: 1px solid black; border-radius: 50%; width: 15px; height: 15px; display: flex; align-items: center; justify-content: center;">v3</div> -- <div style="border: 1px solid black; border-radius: 50%; width: 15px; height: 15px; display: flex; align-items: center; justify-content: center;">v1</div> </div> <div style="border: 1px solid black; padding: 2px;"> <div style="border: 1px solid black; border-radius: 50%; width: 15px; height: 15px; display: flex; align-items: center; justify-content: center;">v4</div> </div> </div>
3	209.89	<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="border: 1px solid black; padding: 2px;"> <div style="border: 1px solid black; border-radius: 50%; width: 15px; height: 15px; display: flex; align-items: center; justify-content: center;">v3</div> -- <div style="border: 1px solid black; border-radius: 50%; width: 15px; height: 15px; display: flex; align-items: center; justify-content: center;">v1</div> </div> <div style="border: 1px solid black; padding: 2px;"> <div style="border: 1px solid black; border-radius: 50%; width: 15px; height: 15px; display: flex; align-items: center; justify-content: center;">v9</div> </div> </div>

Query-Specific Summarization: The extraction of the most relevant pieces of information from a web page using the notion of the page spanning tree has another application (side product), in addition to being a component in creating composed pages. In particular, it is used to perform query-

specific summarization of web pages. The most popular use of query-specific summarization today is the snippets displayed for each of the page results of Web search engines. We show how the query-specific summaries corresponding to page spanning trees have better quality than current approaches.

Florida International University, a member of the State University System of Florida, is a fully accredited comprehensive, multi-campus urban **research** institution located in Miami, Florida (more)
 Open House, Latest **Scholarships**, Honors College

Figure 3. Top summary of web page 1 of Figure 1 for query “research scholarships”.

Example 1 (cont’d): For the web page 1 of Figure 1 and the keyword query “Research Scholarships”, the top summary v3-v4 is shown in Figure 3. The top summary is the top spanning tree of the page graph of page 1 shown in Figure 2. Nodes v3 and v4 are associated because they are adjacent in the text (stronger associations are assigned when the nodes have common words as explained below in the text). □

In summary, this work has the following contributions:

- We introduce the notion of composed pages to improve the quality of Web search. Composed pages are created on-the-fly by combining appropriate content from other pages.
- We show how the most relevant information is extracted from a page by viewing a page as a page graph and computing a query-specific page spanning tree. This has application to query-specific summarization, in addition to the generation of composed pages.

- We propose efficient heuristic algorithms to compute the top composed pages as well as query-specific summaries. The efficiency of the algorithms is evaluated experimentally.
- We show using user surveys that the inclusion of composed pages in the search results increases the user satisfaction. Further, we show that using the idea of the page spanning tree we produce query-specific summaries that are superior to current approaches.
- We have developed prototypes of the Composed Pages system available at <http://dbir.cs.fiu.edu/ComposedPages> and the summarization system available at <http://dbir.cs.fiu.edu/summarization>.

The rest of the paper is organized as follows: Section 2 describes the framework used in our paper; Section 3 describes how page graphs are built; Section 4 describes the idea of query-specific document summarization; Section 5 introduces the idea of searching the Web using composed pages; Section 6 presents the algorithms used in our system; Section 7 and 8 present the quality and performance experiments respectively; Section 9 describes the related work and finally in Section 10 we present our conclusions.

2 FRAMEWORK

2.1 Data Model

Web graph: Let $D=\{d_1, d_2, \dots, d_n\}$ be a set of Web pages d_1, d_2, \dots, 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 a Web page d is the number of occurrences of w in d . Inverse document frequency $idf(w, D)$ is the inverse of the number of Web pages containing term w in them.

The Web graph $G_W(V_W, E_W)$ of a set of Web pages d_1, d_2, \dots, d_n is defined as follows:

- A node $v_i \in V_W$, is created for each Web page d_i in D .
- An (undirected) edge $e(u, v) \in E_W$ is added between nodes $u, v \in V_W$ if there is a hyperlink between u and v .

An example of a web graph is shown in Figure 1. We view the Web graph as undirected since an association between pages occurs along both directions of a hyperlink.

Page graph: In contrast to previous works in Web search [25, 26, 31], we go beyond the page granularity. To do so, we view each page as a set of text fragments connected through semantic associations.

A key component in our work is the page graph $G_d(V_d, E_d)$ of a Web page d which is defined as follows:

- d is split into a set of non-overlapping text fragments and each fragment is represented by a node $v \in V_d$. A text fragment corresponding to a node v is denoted as $t(v)$.
- An undirected, weighted edge $e(u, v) \in E_d$ is added between nodes $u, v \in V_d$ if there is an association

(further discussed in Section 3) between $t(u)$ and $t(v)$ in d .

Figure 2 shows the page graph of Page 1 in Figure 1. The process of building page graphs is explained in Section 3. The page graph is equivalent to the document graph in [38]. Notice that there are many ways to define the page graph for a Web page. In this work we exploit the HTML tags to split the page into text fragments, and edges are added when the text fragments are associated through common (or related) words as we explain in Section 3. The semantic association between the nodes is used to compute the edge weights (query-independent) while the relevance of a node to the query is used to define the node weight (query-dependent). Note that the Web graph now becomes a graph of page graphs.

Search Result: A keyword query Q is a set of keywords $Q=\{w_1, \dots, w_m\}$. Before defining the result of a keyword query we need a few more definitions.

Definition 1 (Minimal Total Web Spanning Tree). Given a Web graph $G_W(V_W, E_W)$, a minimal total Web spanning tree of G_W with respect to a keyword query $Q=\{w_1, \dots, w_m\}$ is a sub-tree T of G_W that is both:

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

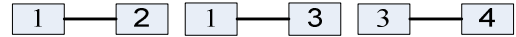


Figure 4. The Minimal Total Web Spanning Trees of Web graph in Figure 1 for query “Graduate Research Scholarships”.

Figure 4 shows the minimal total spanning trees for the query “Graduate Research Scholarships” on the web graph of Figure 1. A result of a keyword query Q at the page granularity is a minimal total Web spanning tree T . We go one step further in order to improve the user’s experience and locate the specific parts of each Web page in T that are relevant to Q . For that, we need the following definition.

Definition 2 (Minimal Total Page Spanning Tree). Given a page graph $G_d(V_d, E_d)$ for a Web page d and a set of keywords $Q_i \subseteq Q$ ($Q_i = Q$ for query-specific summarization), a minimal total page spanning tree p of G_d is a sub-tree of G_d that is both:

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

Figure 5 shows two minimal page spanning trees for Pages 2 and 4 respectively for the query “Graduate Research Scholarships”. In both cases v_2 is a Steiner node, i.e., it does not contain any query keyword in it, but is helpful in forming a minimal total spanning tree for the pages as it has semantic links to the nodes that contain the keywords.

There is a subtle difference in the page spanning tree computation for our two different applications - searching

using composed pages and query-specific summarization. For query-specific summarization of a web page we compute the page spanning tree that contains all the keywords in Q . For the composed pages application, for single-page results we compute the page spanning tree for Q , while for multi-page results we compute them for subsets of Q (see Definition 3). Note that for Steiner nodes, Q_i is empty. In this case p is an empty tree, which we represent by just displaying the *title* of the page in our system.

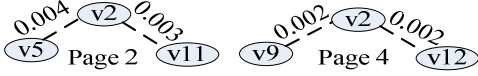


Figure 5. The Minimal Total Page Spanning Trees of Pages 2 and 4 in Figure 1 for query “Graduate Research Scholarships”.

A minimal total Web spanning tree T is “refined” by finding a minimal total page spanning tree p for each of the Web pages $d \in T$ as formally explained in Definition 3. Henceforth we omit the words “minimal total” for brevity if it is clear from the context when referring to minimal total Web spanning trees or page spanning trees. The size of a Web or page spanning tree is the number of edges it contains.

Definition 3 (Search Result). Given a Web graph $G_W(V_W, E_W)$, page graphs for each Web page in G_W , and keyword query $Q = \{w_1, \dots, w_m\}$, a search result R is a minimal total Web spanning tree T with nodes (pages) d_1, \dots, d_z , along with a minimal total page spanning tree for each d_i with respect to a subset Q_i of Q . Each page d_i is assigned a subset Q_i of Q (d_i must contain all keywords in Q_i although it may contain more keywords of Q than Q_i) such that $Q_i \cap Q_j = \emptyset$ for every $i \neq j$, and $Q_1 \cup \dots \cup Q_z = Q$. \square

For example, Table 1 shows the Top-3 search results for the query “Graduate Research Scholarships”. The Web spanning tree 3–1 gives rise to two search results. Page 3 contains keywords “graduate” and “research” and Page 1 contains “research” and “scholarships”, that is, keyword “research” appears in both pages. One search result is computed with subsets $Q_1 = \{\text{graduate, research}\}$ for Page 3 and $Q_2 = \{\text{scholarships}\}$ for Page 1, while the other with $Q_1 = \{\text{graduate}\}$ for Page 3 and $Q_2 = \{\text{research, scholarships}\}$ for Page 1. We only return the best (see Section 5.1 for ranking) search result for each Web spanning tree to the user as shown in Table 1.

Problem definitions: We are now ready to formally define the two problems addressed in this work. The scoring of search results and summaries trees is presented in Sections 5.1 and 3.1 respectively. Smaller scores correspond to higher ranking.

Problem 1 (Top-k Search Results). Given a Web graph G_W , the page graphs for all pages in G_W , and a keyword query Q , find the k search results R with minimum $\text{Score}(R)$. \square

Problem 2 (Query-Specific Summarization). Given a document $d \in D$ and its page graph G_d , and a keyword query Q , find the best summary, i.e., the minimal total spanning tree with minimum score. \square

Notice that typically a single summary per page is re-

quired and hence Problem 2 is a top-1 problem. Notice that the totality property implies that we use *conjunctive* query semantics (AND). Applying OR semantics to Problem 2 is straightforward, as we just replace Q by Q' , where Q' is the set of query keywords contained in the page. Applying OR semantics to Problem 1 is unintuitive since the primary purpose of the composed pages approach is to produce complete (total) answers to the user.

3 BUILDING PAGE GRAPHS

The page graph $G_d(V_d, E_d)$ of a page $d \in D$ is constructed as follows. First we parse d and split it into text fragments using parsing delimiters (e.g., $\langle p \rangle$, $\langle br \rangle$ tags). Each text fragment becomes a node in the page graph. A weighted undirected edge is added to the page graph between two nodes if they either correspond to adjacent text fragments in the text or they are semantically associated. The weight of an edge denotes the association degree of the association.

There are many possible ways to define the association degree between two text fragments. In this work we consider two fragments to be associated if they share common words (excluding stop words) and the degree of association is calculated by an adaptation of traditional IR term weighting formulas [35], as described below. We also consider a thesaurus to enhance the word matching capability of the system. In future versions of our system we will consider using WordNet and Latent Semantic Indexing (LSI) techniques to improve the quality of the edge weights. To avoid dealing with a highly interconnected graph, which would lead to slower execution times and higher maintenance cost, we only add edges with weights above a threshold. Also notice that the edge weights are query-independent, so they can be pre-computed. Q is only used in assigning weights to the nodes of G_d .

The following input parameters are required during the pre-computation stage to construct the page graph:

1. *Threshold for edge weights.* Only edges with weights not below *threshold* will be created in the page graph. The choice of the threshold is a tradeoff between performance and quality, since a zero threshold would build a dense graph which would increase the processing time, while a higher threshold would decrease the quality of results by not including enough edges.
2. *Parsing Delimiters.* Parsing delimiters are used to split the Web page into text fragments. Typical choices are the $\langle p \rangle$ (paragraph) tag (each text fragment corresponds to a paragraph) or the $\langle br \rangle$ (each text fragment is a sentence). Other tags that could be surrounding a possible text fragment are the $\langle table \rangle$ tag, $\langle ul \rangle$, $\langle ol \rangle$ tags and so on. For all these tags the text between the opening and closing counterparts constitute a text fragment. In this way we found a set of tags that when used as delimiters lead to paragraphs that are typically short and leads to more compact page graphs. For plain text documents, typical choices are newline characters (each text fragment corresponds to a paragraph)

or periods (each text fragment corresponds to a sentence).

3. *Maximum Text Fragment Size*. This is used in cases where a fragment is too long which would lead to large nodes (text fragments) and hence large summaries. Users typically desire concise and short summaries.

After parsing the page 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 $EScore(e) \geq \text{threshold}$, then e is added to E_d . The score of edge $e(u, v)$ where nodes u, v have text fragments $t(u), t(v)$ respectively is:

$$EScore(e) = \frac{\sum_{w \in (t(u) \cap t(v))} (tf(t(u), w) + tf(t(v), w)) \cdot idf(w))}{size(t(u)) + size(t(v))} \quad (1)$$

where $tf(d, w)$ is the number of occurrences of w in d , $idf(w, D)$ is the inverse of the number of pages containing w , and $size(d)$ is the size of the page (in number of words). That is, for every word w appearing in both text fragments we add a quantity proportional to the $tfidf$ score of w . Notice that stop words are ignored. Furthermore, we use thesaurus and stemmer (we rely on Oracle InterMedia [30] as explained in Section 8) to match words that are related. 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.

Edges between adjacent fragments: We consider adjacent fragment edges as a special case because two adjacent fragments are semantically related because of their close proximity. Furthermore, linking the adjacent nodes ensures the connectivity of the page graph. We use the following formula, which ensures that there is always an edge between nodes with adjacent text fragments:

$$EScore(e) = \max(EScore(e), \text{threshold}) \quad (2)$$

The calculation of the edge weights concludes the query-independent part of the page graph creation. Next, when a query Q arrives, the nodes in V_d 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 [35] (Equation 3). In order to accelerate this step of assigning node scores we build a full-text index on the set D of pages. The details of this index are out of the scope of this paper.

$$\sum_{t \in Q, d} \ln \frac{N - df + 0.5}{df + 0.5} \cdot \frac{(k_1 + 1)tf}{(k_1(1 - b) + b \frac{dl}{avdl}) + tf} \cdot \frac{(k_3 + 1)qtf}{k_3 + qtf} \quad (3)$$

where tf is the term's frequency in document (page), 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.

3.1 Ranking of Page Spanning Trees

In this section we present our ranking framework for page spanning trees. Recall that the top page spanning tree is the query-specific summary for Problem 2 (Section 2.1). Given the page graph G_d of page d and a query Q , a page spanning tree p is assigned a score $Score(p)$ by combining the scores of the nodes $v \in p$ and the edges $e \in p$.

$$Score(p) = a \sum_{edge e \in p} \frac{1}{EScore(e)} + b \sum_{node v \in p} \frac{1}{NScore(v)} \quad (4)$$

where a and b are constants discussed below. $EScore(e)$ is the score of edge e using Equation 1, $NScore(v)$ is the score of node v using Equation 3.

Intuitively, if p is larger (has more edges) then its score should degrade (increase) since larger trees denote looser semantic connections [2,7,21,22]. This is the reason we take the sum of the inverse of the edge scores in Equation 4. Furthermore, if more nodes of p are relevant to Q , the score should be improved (decreased). 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. We use $a=1$ and $b=0.5$ in our system, which we have found to produce high-quality answers.

4 QUERY-SPECIFIC SUMMARIZATION

This section tackles Problem 2 of Section 2.1. Given a query Q and a page graph G_d for a page d , the query-specific summary is the page spanning tree p of the G_d with minimum $Score(p)$, according to Equation 4.

Example 2. Figure 7 shows the page graph for the page of Figure 6. The page is first split into text fragments $v0 \dots v16$, which correspond to its paragraphs (the newline delimiter was used). Notice that the edge between nodes $v8$ and $v7$ has the highest weight because there are many infrequent (and hence with high idf value) words that are common between them like “Donoghue” and “BrainGate”. □

Table 2 shows the top-ranked spanning trees for the page graph of Figure 7 for the query “Brain chip research”. The values shown above the nodes in Table 2 indicate the node scores with respect to the query, computed by Equation 3. The top result (whose textual representation is shown in Figure 8) is the best summary 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 like “computer” and “brain” whereas in Result #3 they are connected through infrequent words like “Friehs”.

(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)
 (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)
 (v6)
 (v7) John Donoghue, professor of neuroscience at Brown and a co-founder of Cyberkinetics in 2001, said that BrainGate could help paralyzed people control wheelchairs and communicate using email and Internet-based phone systems.
 (v8)
 (v9)
 (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)
 (v13)
 (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)

Figure 6. Sample news page from www.cnn.com.

5 SEARCH USING COMPOSED PAGES

This section tackles Problem 1 of Section 2.1. In Section 5.1 we explain how a search result (Definition 3) is ranked, while Section 5.2 discusses how a composed page is constructed for a search result.

5.1 Ranking search results

Recall that a *search result* R is a Web spanning tree T where each page d in T is represented by its page spanning tree p . Clearly there is no optimal ranking function since it is possible to come up with different ranking functions for different domains or specific queries. In this work we adopt principles well-accepted in previous works on ranking Web pages [25,26,31] and trees of data [2,7,14,17,21,38].

The *first ranking principle* we adopt [26] is that search results involving fewer pages are ranked higher. Intuitively, if a search result is larger (has more edges) then its score should degrade (increase) since larger trees denote looser semantic connections. Hence, search results are primarily ranked by the (inverse of the) size of their Web spanning tree. Recall that by Definition 3, all search results contain all query keywords.

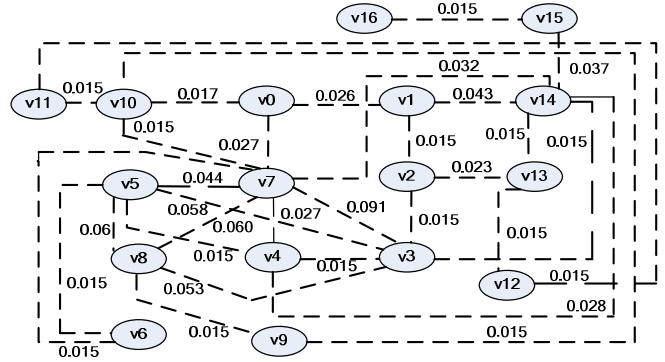


Figure 7. Page Graph of the sample page in Figure 6.

Table 2. Top Summaries for query “Brain Chip Research”.

Rank	Score	Summary
1	67.74	0.046 v0 0.017 v10 0.008
2	84.77	0.046 v0 0.027 v7 0.0 0.0003 v4 0.027
3	87.64	0.012 v1 0.043 v14 0.0 0.0005 v15 0.037
4	103.77	0.008 v10 0.015 v11 0.005
5	167.41	0.046 v0 0.027 v7 0.032 v14 0.0 0.037 v15 0.0005

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 8. Top Summary for document of Figure 6 for query “Brain Chip Research”.

Within search results with the same size of Web spanning tree, we rank according to the scores of the involved page spanning trees, computed by Equation 4. Note that the first ranking principle also applies in ranking individual page spanning trees as expressed in Equation 4, that is, page spanning trees with smaller size are ranked higher.

What is left, is to define how the scores of the constituting page spanning trees computed by Equation 4, are combined to compute the overall score of a search result. Again, we do not claim that we have the optimal combining function, but we rely on previous work to define the next



Figure 9. Composed Page for Search Result #1 for query “Graduate Research Scholarships”.

principle. The *second ranking principle* is that the scores of the page spanning trees are combined using a monotone combining function to compute the score of the search result. Notice that we already used another variant of this principle in Equation 4, where the scores of the nodes and edges are combined using a monotone function.

To incorporate the global importance of the pages used in constructing a search result, we use their PageRank [31] values. Equation 5 computes the score of a search result R given the scores of its page spanning trees p , where we chose summation as our monotone combining function.

$$Score(R) = \sum_{p \in R} \frac{Score(p)}{PR(p)} \quad (5)$$

where $PR(p)$ is the PageRank score of page d that contains the page spanning tree p .

5.2 Composed Pages

A composed page is a dynamic page created on-the-fly by stitching together pieces from other pages. Given a query Q , a composed page is a representation of a search result, as defined in Definition 3, in a Web page format. The score of a composed page is the score of the corresponding search result defined by Equation 5.

The key requirements in constructing a composed page are the following: First, display the tree-structured (more specifically tree of trees) search result in a page format. Second, allow users to easily navigate to the original pages that were used to construct the composed page. Figure 9 shows the composed page constructed for the Search Result #1 of Table 1. A composed page for a search result is constructed by displaying links to all pages in its Web spanning tree along with the text fragments of the page spanning trees. The page spanning trees are displayed in an unordered list format that depicts their structure. A sub-bulleted list denotes the parent-child relationship in the page spanning tree of text fragments.

6 ALGORITHMS

In this section we present various algorithms used in our system. Note that the algorithms used in the query-specific summarization problem are also used as a component of the composed pages problem. The pre-computation requirements are also the same. Section 6.1 presents the algorithm for preprocessing the Web pages and creating a database of *page graphs*. Section 6.2 presents algorithms to solve the query specific summarization problem (Problem 2), which are adapted in Section 6.3 to solve the top- k search results problem (Problem 1).

6.1 Preprocessing Algorithm

Figure 10 describes the preprocessing algorithm. Before any query arrives we pre-compute and store the following:

- *The page graph for each page.* In particular, we parse the HTML documents based on the tags, as described in Section 3 and compute the edge weights. The parameters described in Section 3 are taken as input and page graphs are built accordingly.
- *PageRank values* of each page by executing the PageRank algorithm [31].
- *A full-text index* to efficiently locate the pages and specifically the text fragments that contain the keywords and calculate their query-specific score.
- In order to boost the performance of the algorithms, *the all-pairs shortest paths* between the nodes of the page graph G_d of every page d . Note that the inverse of the edge weights is used since larger edge weights denote tighter association in our setting.

Preprocess (Web Graph G_w , Parsing Delimiters P , Threshold τ , Maximum Fragment size sz)

1. For each web page (node) d in G_w do {
/* create and store page graph G_d for d */
2. Parse d and split it into text fragments with maximum size sz using the delimiters in P ;
3. Create a node for each text fragment and add it to the page graph, G_d of d ;
4. For every pair of nodes in G_d find if they are semantically related by calculating the edge weight using Equation 1 and add it to G_d if the edge weight $\geq \tau$;
5. For every pair of adjacent nodes, build an edge e with weight equivalent to $\max(EScore(e), \tau)$ according to Equation 2; /*in close proximity as explained*/
6. Find All-pairs shortest path using Floyd Warshall's algorithm using the inverse of each edge's weight;}
7. Compute and store the PageRank values of all pages (nodes) in G_w ; /* compute PageRank values; build full-text index*/
8. For each keyword w locate and store all pages in D that contain w ; /*Stemming is used in this step. Stop words are ignored*/

Figure 10. Preprocessing Algorithm.

6.2 Compute Top-1 Page Spanning Trees (Query-Specific Summaries)

For both Problems 1 or 2, we need to solve a variant of the

Group Steiner Tree problem, which is referred to as keyword proximity search problem [7,14] and is defined as follows:

Given a weighted data graph $G(V, E)$, a keyword query Q which is a set of keywords, and an integer k , find the k minimum-weight sub-trees of G such that every keyword in Q is contained in at least one vertex of the sub-tree, and we cannot remove any node from it and still have a tree.

When $k = 1$, the keyword proximity search problem has been shown to be equivalent to the Group Steiner problem, which is NP-complete. The keyword proximity search 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 $G=(V, E)$; and given a family $R=\{R_1, \dots, R_k\}$ of disjoint groups of vertices, where R_i is a subset of V , find a minimum-cost tree T that 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.

This section presents two algorithms adapted from BANKS [7] to compute the top query-specific summary: the enumeration and the expanding search algorithms. The algorithms return a top-1 summary for a Web page d , given its page graph G_d and a query Q . The reason we employ top-1 summary algorithms is that typically the user only requests a single summary for a document, as in the case of snippets in Web search engine results.

Top-1 Enumeration Algorithm: This algorithm, which is abbreviated as *Top-1-MTPST-Enumeration* (Top-1-Minimal-TotalPageSpanningTree-Enumeration), is shown in Figure 11. First, we find all combinations of nodes in G_d that are minimal (no node is redundant) and total (collectively 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 of edges between them with weight equal to their pre-computed shortest-path distance. We then calculate all possible spanning trees in G_c , and compute their scores using Equation 4 and so on (see Figure 11 for more details). This algorithm accepts a Quality parameter ω . Higher values of ω yield higher quality results. Intuitively this parameter decides the number of different summaries that are considered before we pick the best one, given that this is an NP-complete problem.

Example 2 (cont'd). Consider the page graph in Figure 7 and the query “Brain chip research”. The nodes that contain the keywords are $v_0, v_1, v_3, v_4, v_{10}, v_{11}$, and v_{15} . We then find all minimal and total node combinations, which are $\{v_0, v_{10}\}$, $\{v_{15}, v_0\}$, $\{v_0, v_3\}$, $\{v_4, v_0\}$, and so on. For each combination we create a closure graph. For example, the closure graph for the second combination is $v_{15} \sim v_0$ with edge weight 88.74 (which is the length of the shortest path from v_{15} to v_0). We then find all possible spanning trees of this graph, which is just $v_{15} \sim v_0$, for the above closure graph. Then, we replace the edge between v_{15} and v_0 with the shortest path between them, which is $v_{15} \sim v_{14} \sim v_1 \sim v_0$. This tree is not minimal and hence we trim it to get the minimal result $v_{15} \sim v_{14} \sim v_1$ and output this result along with its score. \square

Top-1-MTPST-Enumeration (Page Graph G_d , Query Q , Quality parameter ω)

```

1. Results  $\leftarrow \emptyset$ ; /*stores summaries*/
2. Find all nodes in  $G_d$  that contain some keyword of  $Q$ ; /*use full-text index*/
3. Find all minimal combinations of nodes that collectively 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 Equation 4 by using shortest path weights between any two nodes;
8.   Pick the spanning tree  $p$  with the minimum score;
9.   Replace the edges  $u \sim v$  in  $p$  with their precomputed shortest paths  $u \sim u_1 \sim \dots \sim u_k \sim v$ ; /* i.e., we are adding the Steiner nodes.*/
10.  Trim  $p$  to make it a minimal total spanning tree;
11.  Recalculate the score of  $p$  using Equation 4 and add  $p$  to Results;
12.   $\omega--$ ;
13.  If( $\omega == 0$ ) Return the top ranked summary in Results; }
```

Figure 11. Top-1 Enumeration Algorithm.

Top-1 Expanding Search Algorithm: The basic idea is that an expanding area is created for each keyword node (node that contains a query keyword) of G_d and we start from the nodes that contain the query keywords and progressively expand them according to a shortest-paths algorithm until we find all minimal total spanning trees. In particular, the algorithm (Figure 12) finds (using the pre-computed full-text index) all the nodes that match some keywords in the query and starts expanding them incrementally. We call the sub-graph created from each keyword node v , expanding area of v . At each iteration, we expand each expanding area in parallel by adding all adjacent edges (later we discuss heuristics of expansion) to the expanding area of the previous iteration. A result (summary) is generated when a set of expanding areas meet at a common point (node) and form a minimal total page spanning tree for Q .

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 4.

Example 2 (cont'd). For the page graph in Figure 7 and the query “Brain chip research”, we grow the expanding area of v_0 to $v_0 \sim v_{10}$, which is the first precomputed shortest path of source v_0 and check for possible summaries. $v_0 \sim v_{10}$ is total as well as minimal and hence we add it to the set of results. We continue growing each expanding area using its precomputed shortest paths. Then we grow v_1 to $v_1 \sim v_2$, v_3 to $v_3 \sim v_2$, v_4 to $v_4 \sim v_3$, v_{10}

to $v_{10} \sim v_9$, v_{11} to $v_{11} \sim v_{10}$ and once we expand v_{11} we have another summary $v_{11} \sim v_{10}$ that is total and minimal. \square

```

Top-1-MTPST-ExpandingSearch(Page graph  $G_d$ , Query  $Q$ , Quality parameter  $\omega$ )
1.  $Results \leftarrow \emptyset$ ; /*stores summaries*/
2. Find all nodes  $N = \{N_1, \dots, N_m\}$  that contain the
   keywords in  $Q$  and create expanding areas for
   each; /* $N_i$  has the nodes that contain  $w_i$ */
3. Repeat until each expanding area spans the
   entire graph  $G$  {
4.   For each node  $v$  in  $N$  do {
5.     Add to the expanding area of  $v$  the mini-
       mum-score adjacent edge from the (pre-
       computed) 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);
       /*i.e., trees that contain a node from
       each of  $N_1, \dots, N_m$  */
7.     Trim summaries to make them minimal;
8.     Calculate the score of each summary  $p$ 
       using Equation 4 and store in  $Results$ ;
9.      $\omega--$ ;
10.    If( $\omega=0$ ) Return the top ranked summary
        in  $Results$ ;}}

```

Figure 12. Top-1 Expanding Search Algorithm.

6.3 Compute Top-k Search Results

This algorithm is an adaptation of the Top-1 expanding search algorithm of Section 6.2. It also uses the *Top-1-MTPST-ExpandingSearch* method as a subroutine to compute the page spanning trees of the pages in a Web spanning tree. We adapt expanding search and not the naïve enumeration algorithm since the former is shown to perform better in Section 8. The key differences from the algorithm of Figure 12 are the following. First, *Heuristic-Top-k-Expanding-Search* (Figure 13) operates on Web graphs instead of page graphs, and hence produces web spanning trees instead of page spanning trees. Second, we introduce the following heuristic based on Equation 5, which is our ranking function. In particular, we first expand towards pages d with highest *HeuristicWeight* value as defined by:

$$\text{HeuristicWeight}(d) = PR(d) * IRScore(d) \quad (6)$$

where d is a Web page, PR its PageRank value, and $IRScore(d)$ its Information Retrieval score for Q . The $PR(d)$ component of Equation 6 is intuitive since it also appears in the ranking equation (Equation 5). The $IRScore(d)$ component is a heuristic estimate of the $Score(p)$ component of Equation 5, where p is the page spanning tree for page d . The intuition is that a page with high IR score for Q is also expected to have page spanning trees with high score for Q . We use the full-text indexer to compute $IRScore(d)$. Finally, notice that *Heuristic-Top-k-Expanding-Search* algorithm has two steps: first it computes the Web spanning trees, and for each one of them it computes the top search results by computing the corresponding page spanning trees for its pages (*getTopSearchResult* method). The following are the key steps of the algorithm involved in computing the top-k

search results for a query Q .

- Compute a minimal total Web spanning tree, *WST* given the web graph G_w and query Q .
- Then compute the best search result for *WST*, given the page graphs of each page in *WST* and the query Q by considering all possible combinations of keyword assignments to the pages of *WST*, according to the constraints of Definition 3.

The above steps are repeated until k search results are computed. The *getTopSearchResult* method takes as input a web spanning tree and the page graphs of the constituent pages and returns the best search result after evaluating all possible search results. It uses the *Top-1-MTPST-ExpandingSearch* method to compute the top page spanning trees corresponding to the query.

Table 3. Real & Synthetic Datasets

Name	#nodes (Web pages)	#edges (Hyperlinks)	Size (MB)
FIU1	25,108	137,929	4564
FIU2	6,054	45,405	115

7 QUALITY EXPERIMENTS

To evaluate the quality of the results of our approach for Problems 1 and 2, we conducted three surveys, one for Problem 1 and two for Problem 2. The subjects of the survey are twenty students (of all levels and various majors) at Florida International University (FIU), who were not involved in the project. In these surveys the users were asked to evaluate the results based on their quality. Sections 7.1 and 7.2 present the results for Problems 1 and 2 respectively.

Datasets: We use two real datasets (Table 3). FIU1 is a hyperlinked set of 25,108 Web pages (nodes) crawled from the *fiu.edu* domain, connected through 137,929 hyperlinks (edges) used for performance evaluation. FIU2 is a subset of the web pages available in *fiu.edu* domain used for quality evaluation, which offers faster response times and more focused results that are easier to compare.

7.1 Composed Pages

We used FIU2 for our user surveys. The participants were asked to evaluate the quality of the search results with respect to ten queries. We chose both long and medium sized queries. For each query, users were asked to rate their satisfaction for the Top-5 search results produced from the Heuristic Top-k Expanding Search algorithm of Figure 13, and for the results produced by Google. We chose the first 5 results from Google that are included in the subset of crawled FIU web pages. The Google query was constrained to pages using the “site: fiu.edu” condition. Each participant was asked to assign a score between 1 and 5 to each alternative query answer, where 5 denote the highest user satisfaction. The results of the survey prove the superiority of our approach, as shown in Table 4.

7.2 Query-Specific Summarization

To evaluate the quality of our query-specific summaries we created two user surveys on a DUC and a Web dataset as explained below. The size of a result was also taken into

```

Heuristic-Top-k-Expanding-Search(Web graph  $G_w$ , Page graphs  $PG = \{G_{d1}, G_{d2} \dots G_{dn}\}$ , Keyword query  $Q = \{w_1, \dots, w_m\}$ )
1.  Results  $\leftarrow 0$ ; /* result count */
2.  Find all keyword nodes  $KN$  in  $G_w$  using the full text index; /*nodes that match some keyword in  $Q$ */
3.  Let  $Z_j$  be the set of nodes of  $G_w$  that contain  $w_j$ ;
4.  Let  $L_j$  be the set of expanding areas corresponding to the root nodes in  $Z_j$ ;
5.  Let  $buffer(i)$  be an array ordered by score to buffer search results containing  $i$  pages;
6.  For each node(page) $d$  contained in  $Z_1 \cap Z_2 \cap \dots \cap Z_m$  do { /*single-page search results*/
7.       $TSR \leftarrow getTopSearchResult(d, \{G_d\}, Q)$ ;
8.      Insert  $TSR$  into  $buffer(1)$ ; /* Insert  $TSR$  into the ordered  $buffer$  of single page search results */
9.      Results++;}
10. While (Results <  $k$ ) {
11.     For  $j$  in  $1 \dots m$  do {
12.         For each expanding area  $L$  in  $L_j$  do {
13.             Expand the expanding area  $L$ , with a node  $v$  having the maximum HeuristicWeight; /* Equation 6*/
14.             Join  $v$  to all previously expanded nodes  $u$  generated by the expanding areas  $L_s, s \neq j$ ;
15.             /* By "join" we mean find all instances of  $v$  as an end node in the already expanded nodes. */
16.             For each web spanning tree  $WST$  generated by the join {
17.                 Trim useless leaves to make it minimal;
18.                  $TSR \leftarrow getTopSearchResult(WST, \{G_{d1}, G_{d2} \dots G_{dx}\}, Q)$ ;
19.                 Insert  $TSR$  in to  $buffer(length(TSR))$ ; /* length( $TSR$ ) equals number of pages in  $TSR$  */
20.                 Results++; If (Results =  $k$ ) { Output results in  $buffer$  and return; } } } } }
MODULE: getTopSearchResult(Web spanning tree  $WST$ , Page graphs  $WPG = \{G_{d1}, G_{d2} \dots G_{dx}\}$  of  $WST$ , Keyword query  $Q = \{w_1, \dots, w_m\}$ )
1.  SearchResults  $\leftarrow \emptyset$ ; /*stores search results*/
2.  Find the set of possible partitions  $PQ$  of  $Q$  as per Definition 3;
3.  For each partition  $\{Q_1, \dots, Q_z\}$  of the keywords in  $PQ$  do{
4.      For each page  $d_i$  in  $WST$  do {
5.           $PSP_i \leftarrow \emptyset$ ;
6.          If ( $Q_i \neq \emptyset$ ) {
7.               $PSP_i \leftarrow Top-1-MTPST-ExpandingSearch(G_{d_i}, Q_i, \omega)$ ; } } /*  $Q_i$  is the subset of  $Q$  assigned to page  $d_i$ ,  $\omega$  is the quality factor*/
8.          Create a search result  $R$  with each  $PSP_i$  and  $WST$ ; /*if  $PSP_i = \emptyset$  we use the title of page  $d_i$  (this corresponds to the Steiner node which has no keywords in it)*/
9.          Compute  $Score(R)$  using Equation 5 and add  $R$  to  $SearchResults$ ;
10. Return the top ranked search result in  $SearchResults$ ;

```

Figure 13. The Heuristic Top-k Expanding Search Algorithm.

Table 4. Average Top-5 search result ratings for 10 queries.

Keyword Queries	Google Search	Heuristic Expanding Search	Keyword Queries	Google Search	Heuristic Expanding Search
<i>Undergraduate Housing safety</i>	2.06	3.41	<i>Undergraduate Summer athletics accomplishments</i>	2.25	4.5
<i>Graduate financial aid regulations</i>	2.41	3.59	<i>Physics alumni achievements</i>	3.25	3.00
<i>Computer Science Internship opportunities</i>	2.88	3.65	<i>Electrical transfer student eligibility</i>	2.66	4.66
<i>Campus Safety requirement regulations</i>	2.24	3.35	<i>Freshman internship opportunities</i>	1.66	4.66
<i>Biomedical Research fellowship eligibility</i>	1.24	3.35	<i>Mechanical Graduate admission policies</i>	1.66	4.66
Average Rating	2.16	3.47	Average Rating	2.29	4.29

consideration by the participants – a longer result 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 a summary that is most (least) descriptive.

Comparison with DUC dataset

The dataset used in this survey consists of twenty documents and four queries taken from the DUC 2005 dataset [10] as shown in Table 6. 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 twenty documents over four topics. The sentences that have

Table 5. Average summary ratings for documents.

Documents	Keyword Queries														
	Google Desktop Summary					MSN Desktop Summary					Top-1 Expanding Summary				
	Q1	Q2	Q3	Q4	Q5	Q1	Q2	Q3	Q4	Q5	Q1	Q2	Q3	Q4	Q5
D1	2.33	2.00	3.00	1.67	2.00	2.33	2.00	0.67	1.67	3.00	4.87	4.33	4.93	4.67	4.00
D2	3.67	3.33	2.67	2.67	1.67	3.67	3.00	3.00	3.00	1.00	3.67	3.33	4.00	4.00	3.67
D3	1.60	1.60	2.00	1.60	2.00	1.60	1.00	1.80	2.20	1.20	4.00	4.20	4.00	3.60	3.40
D4	1.00	1.33	0.66	1.33	2.33	2.66	2.00	1.33	1.66	1.33	3.66	3.66	4.00	4.00	3.33
D5	2.50	3.00	2.50	1.00	3.00	1.50	1.50	1.50	2.00	3.50	4.00	3.50	4.00	4.00	3.50
D6	1.00	1.50	1.50	2.50	1.00	2.00	2.50	1.50	3.50	2.00	4.00	4.50	4.00	2.50	4.00
D7	3.00	1.00	3.00	1.00	1.00	1.50	2.50	1.50	2.50	2.00	3.00	4.00	3.00	4.50	4.50
Average Rating	1.97					2.00					3.89				

Table 6. Average summary ratings for DUC topics.

Query 1 (International Organized Crime) DUC Topic ID: d301i			Query 2 (Women in Parliaments) DUC Topic ID: d321f			Query 3 (Drugs Mental Illness) DUC Topic ID: d383j			Query 4 (Stolen Art Recovered) DUC Topic ID: d422c		
Doc. ID	DUC Peer	Top-1 Expanding	Doc. ID	DUC Peer	Top-1 Expanding	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	FT933-4868	2.00	4.33	LA051889-0110	4.00	3.00
FT944-8297	2.50	3.33	FT922-190	2.00	4.00	FT942-16465	1.00	5.00	FT911-5359	2.00	3.00
FT931-3563	2.83	3.00	FT921-937	2.00	4.33	LA090389-0060	1.66	4.33	LA070990-0048	2.33	4.33
FT943-16477	4.00	4.17	FT922-13353	2.83	4.17	FT922-715	1.00	4.33	LA032090-0091	3.00	3.66
FT943-16238	3.67	3.67	FT921-74	2.33	3.67	LA111290-0137	1.66	4.33	FT923-1946	4.33	3.00
Average	3.06	3.77	Average	2.63	3.73	Average	1.46	4.46	Average	3.13	3.40

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.

The results of the survey prove the superiority of our approach, as shown in Table 6. 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.

Comparison with Google and MSN Desktop

The dataset used in this survey consists of seven news documents taken from the technology section of *cnn.com*. The participants were asked to evaluate the quality of the summaries of the seven documents with respect to five queries each (35 queries in total). 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 com-

pare 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 4, 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 7. Queries used for documents.

Query #	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 show the superiority of our approach, are presented in Table 5, while the queries are shown in Table 7 (only 10 queries are shown while the remaining 25 are omitted due to space constraints). 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 between them. In contrast, our approach always finds a meaningful way to connect them.

8 PERFORMANCE EXPERIMENTS

We evaluate the performance of the algorithms presented

in Section 6. Section 8.1 shows our results for Problem 2 while Section 8.2, for Problem 1. We used a Linux machine with Power 4+1.7GHz processor and 3.7 GB of RAM. The algorithms were implemented in Java. To build the full-text index we used Oracle interMedia [30] and stored the documents in the database. JDBC was used to connect to the database system. We used the pre-computation technique described in Section 6.1. We used FIU1 (described in Section 7) for performance evaluation as FIU2 is a very small dataset for this purpose.

8.1 Query-Specific Summarization

First, we compare the performance of the two algorithms of Section 6.2 for summarizing keyword queries of various lengths. The execution time consists of two parts: (a) the computation of the scores of the nodes of the page graph (remember that this is query-specific and cannot be pre-computed), and (b) the generation of the top summaries (minimal total page spanning trees) in the page graph. The first part is handled by Oracle interMedia [30] and the average time for a single page for various-length queries is shown in Table 8. The second part of the execution is handled separately by the two algorithms and the results are shown in Figure 14. In particular, Figure 14 compares the performance of the Top-1 Enumeration and Top-1 Expanding search algorithms.

Table 8. Average times to calculate node weights.

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

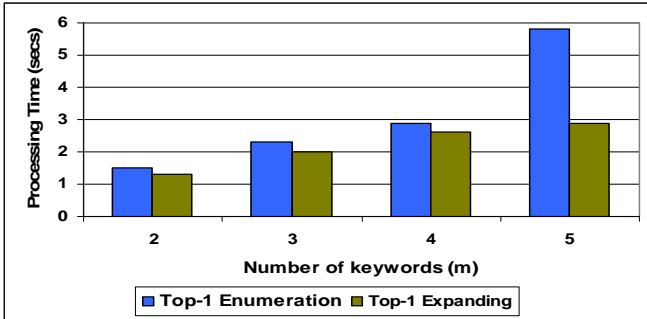


Figure 14. Processing time of Top-1 Algorithms.

We observe that the expanding search algorithms are faster than the enumeration ones, especially for long queries. Notice that we do not compare the performance of our algorithms to BANKS, since our Top-1 algorithms are adaptations of the BANKS algorithms to our problem. In particular, we use the pre-computed all-pairs shortest paths data to efficiently grow the expanding area in the Top-1 Expanding search algorithm and efficiently construct the page spanning tree from the spanning trees of the closure graph in the Top-1 Enumeration algorithm.

Finally, we measure the accuracy of the Top-1 algorithms. In order to have a yardstick to compare the Top-1 algorithm results, we first perform an exhaustive search to find all summaries along with their optimal scores. In particular, we measure (Table 9) the average rank of the summary of the Top-1 algorithms in the optimal list of summaries. We observe that the Top-1 expanding algorithm better

approximates the Top summary in the optimal list of summaries when compared to the Top-1 enumeration algorithm as can be seen in Table 9.

Table 9. Average ranks of Top-1 Algorithms with respect to the Optimal list of summaries.

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

8.2 Searching Using Composed Pages

First, we measure the quality of the *Heuristic-Top-k-Expanding-Search* algorithm as follows. In order to have a yardstick to compare our results, we first perform an exhaustive search to find all search results along with their optimal scores. Then, we measure the quality of the heuristic top- k algorithm by comparing its top- k search results produced with the optimal top- k search results. We compare 2 top- k lists by using Spearman's rho metric as mentioned in Equation 7 below:

$$\rho(\sigma_1, \sigma_2) = \left(\sum_{i=1}^K |\sigma_1(i) - \sigma_2(i)|^2 \right)^{1/2} \quad (7)$$

where ρ is the Spearman's rho metric, σ_1 and σ_2 are 2 top- k lists and $\sigma_1(i)$ and $\sigma_2(i)$ are the ranks of i^{th} search result in each of the top- k lists. Figure 16 shows the average quality of the results (over 50 queries) of our heuristic search and the *Non-Heuristic* expanding search (where a random page is chosen for expansion at every step) compared to the optimal exhaustive search. Figure 16(a) shows the quality of the Heuristic and Non-Heuristic Top- k for fixed $m=2$ (2 keyword queries) and varying number k of requested results. Figure 16(b) shows the quality for fixed number $k=25$ of requested results and varying the query length. As can be seen in Figure 16, expansion based on the *Heuristic-Weight* (Equation 6) yields better Top- k results.

Next, we compare the execution time of the algorithms which consists of two parts: (a) the computation of the web spanning trees in the Web graph, and (b) the generation of the top- k search results. Figure 15 shows the execution time of the different algorithms for computing Top- k search results (Definition 3). As before, we measure the performance with changing k and fixed $m=2$ (Figure 15(a)) and changing m with fixed $k=25$ (Figure 15(b)). Notice that, Figure 15 shows the total execution time of the *Heuristic-Top-k-Expanding-Search* algorithm and its Non-Heuristic and Optimal counterparts. The Heuristic algorithm has a longer execution time when compared to the Non-Heuristic algorithm because during each expansion step it has to select among the available neighbors, the one with the highest *HeuristicWeight* (Equation 6) while the Non-Heuristic algorithm selects a random page for expansion.

9 RELATED WORK

Document Summarization: A large corpus of work has focused on generating query-independent summaries [3,5,6,15]. The OCELOT system [6] provides the summary of a web page by selecting and arranging the most (query-independent) "important" words of the page. Amitay and

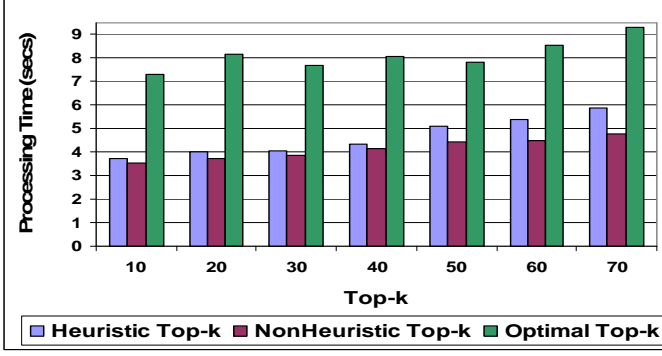
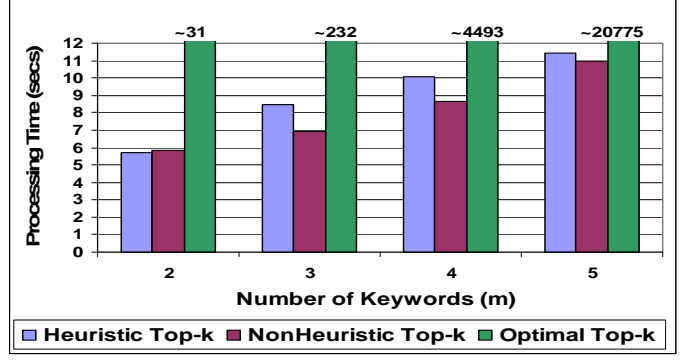
(a) Performance with changing k (with $m = 2$)(b) Performance with changing m (with $k = 25$)

Figure 15. Execution time for Top-k Search Results.

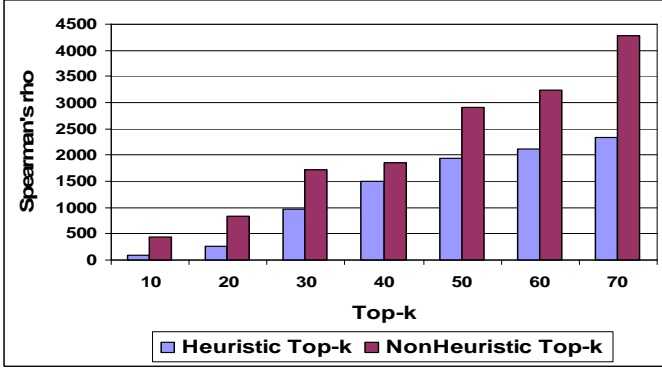
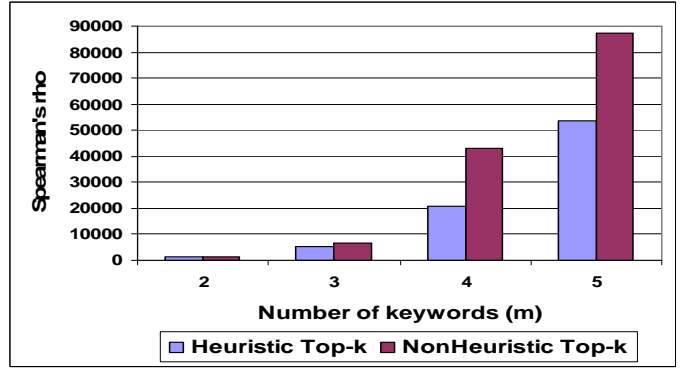
(a) Spearman's rho vs. Top-k (with $m = 2$)(b) Spearman's rho vs. Query size (with $k = 25$)

Figure 16. Quality of Algorithms.

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. [5] uses lexical chains for text summarization.

The majority of systems participating in the past Document Understanding Conference [10] (a large scale summarization evaluation effort sponsored by the United States government), and the Text Summarization Challenge [13] are extraction based. Extraction-based automatic text summarization systems extract parts of original documents and output the results as summaries [9,11,15,19]. Other systems based on information extraction [32] and discourse analysis [27] 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 (as in [38]) we also show the semantic connections between the extracted fragments, which improve the quality as shown in Section 7.2.

White et al. [40], Tombros and Sanderson [37] and Goldstein et al. [15] 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. A number of the highest-scoring sentences are then chosen as the summary. [1,18,34] 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. [33] provide a technique for multi-document summarization used to cluster

the results of a web keyword query. [12,28] 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. [8] and Song et al. [36], where they extract query-independent rankings for the fragments, for the purpose of improving the performance of web search. Cai et al. [8] partition a web page into blocks using the vision-based page segmentation algorithm. Song et al. [36] provide learning algorithms for block importance.

Finally, all major Web search engines generate query-specific snippets of the 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 [16,29].

Keyword search in data graphs: For both Problems 1 and 2, when the page graphs are already created and a query arrives, the system searches the page graphs (also the web graph) for sub-trees that contain all (or a subset of) query keywords. This problem has been studied by the database and graph-algorithms communities. In particular, recent work [2,7,14,17,20,21,23,24] has addressed the problem of free-form keyword search on structured and semi-structured data. Li et al. [26] tackle the problem of proximity search on the Web, which is viewed as a graph of hyperlinked pages.

10. CONCLUSIONS

In this paper, we describe a technique to improve the quality of web search results by on-the-fly creating and ranking composed pages. This technique is particularly successful for long or multi-topic queries where single-page results are unlikely to satisfy the user's information need. We also describe a technique for query-specific web page summarization, which, in addition to having its own merit, is used for computing the top-k composed pages. We have presented and evaluated efficient algorithms for both problems. We also conducted user surveys to measure user satisfaction.

REFERENCES

- [1] J. Abracos and G. Pereira-Lopes: "Statistical methods for retrieving most significant paragraphs in newspaper articles", 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", IEEE ICDE, 2002.
- [3] E. Amitay and C. Paris: "Automatically Summarizing Web Sites - Is there any way around it?" CIKM, 2000.
- [4] A. Balmin, V. Hristidis and 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", ACM SIGIR, 2000.
- [7] G. Bhalotia, C. Nakhe, A. Hulgeri, S. Chakrabarti and S. Sudarshan: "Keyword Searching and Browsing in Databases using BANKS", IEEE ICDE, 2002.
- [8] D. Cai, X. He, J. Wen and W. Ma: "Block-level Link Analysis", ACM SIGIR, 2004.
- [9] H.H. Chen, J.J. Kuo and T.C. Su: "Clustering and Visualization in a Multi-Lingual Multi- Document Summarization System", ECIR, 2003.
- [10] Document Understanding Conference, <http://duc.nist.gov>, 2005.
- [11] H.P. Edmundson: "New Methods in Automatic Abstracting", ACM Journal, vol.16, no.2, pp. 264-285, 1969.
- [12] G. Erkan and D.R. Radev: "Lexrank: Graph-based centrality as salience in text summarization", JAIR, vol.22, pp 457-479, 2004.
- [13] T. Fukusima and M. Okumura: "Text Summarization Challenge Text Summarization Evaluation in Japan", WAS, 2001.
- [14] R. Goldman, N. Shivakumar, S. Venkatasubramanian and H. Garcia-Molina: "Proximity Search in Databases", VLDB, 1998.
- [15] J. Goldstein, M. Kantrowitz, V. Mittal and J. Carbonell: "Summarizing text documents: Sentence selection and evaluation metrics", ACM SIGIR, 1999.
- [16] Google Desktop search, <http://desktop.google.com/>, 2007.
- [17] L. Guo, F. Shao, C. Botev and J. Shanmugasundaram: "XRANK: Ranked Keyword Search over XML Documents", ACM SIGMOD, 2003.
- [18] M.A. Hearst: "Using categories to provide context for full-text retrieval results", In Proceedings of the RIAO, 1994.
- [19] E. Hovy and C.Y. Lin: "The automated acquisition of topic signatures for text summarization", ICCL, 2000.
- [20] V. Hristidis, L. Gravano and Y. Papakonstantinou: "Efficient IR-Style Keyword Search over Relational Databases", VLDB, 2003.
- [21] V. Hristidis and Y. Papakonstantinou: "DISCOVER: Keyword Search in Relational Databases", VLDB, 2002.
- [22] V. Hristidis, Y. Papakonstantinou and A. Balmin: "Keyword Proximity Search on XML Graphs", IEEE ICDE, 2003.
- [23] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai and H. Karambelkar: "Bidirectional Expansion for Keyword Search on Graph Databases", VLDB, 2005.
- [24] B. Kimelfeld and Y. Sagiv: "Finding and Approximating top-k answers in Keyword Proximity Search", PODS, 2006.
- [25] J. Kleinberg: "Authoritative Sources in a Hyperlinked Environment", ACM Journal, vol.46, no.5, pp. 604-632, 1999.
- [26] W.S. Li, K. S. Candan, Q. Vu and D. Agrawal: "Retrieving and Organizing Web Pages by 'Information Unit'", WWW, 2001.
- [27] D. Marcu: "Discourse trees are good indicators of importance in text", Advances in Automatic Text Summarization, 1999.
- [28] R. Mihalcea and P. Tarau: "TextRank: Bringing Order into Texts", EMNLP 2004.
- [29] MSN Desktop search, <http://toolbar.msn.com/>, 2007.
- [30] Oracle interMedia : <http://www.oracle.com/technology/products/intermedia>, 2007.
- [31] L. Page, S. Brin, R. Motwani and T. Winograd: "The pagerank citation ranking: Bringing order to the web", Technical report, Stanford University, 1998.
- [32] D.R. Radev and K.R. McKeown: "Generating Natural Language Summaries from Multiple On-line Sources. Computational Linguistics", vol.24, no.3, pp. 470-500, 1998.
- [33] D.R. Radev, W. Fan and Z. Zhang: "WebInEssence: A Personalized Web-Based Multi-Document Summarization and Recommendation System", NAACL Workshop on Automatic Summarization, 2001.
- [34] G. Salton, A. Singhal, M. Mitra and C. Buckley: "Automatic text structuring and summarization", Information Processing and Management, vol.33, no.2, pp.193-207, 1997.
- [35] A. Singhal: "Modern Information Retrieval: A Brief Overview", IEEE Data Eng. Bull, 2001.
- [36] R. Song, H. Liu, J. Wen and W. Ma: "Learning Block Importance Models for Web Pages", WWW 2004.
- [37] A. Tombros and M. Sanderson: "Advantages of Query Biased Summaries in Information Retrieval", ACM SIGIR 1998.
- [38] R. Varadarajan and V. Hristidis: "A System for Query-Specific Document Summarization", ACM CIKM 2006.
- [39] R. Varadarajan, V. Hristidis and T. Li: "Searching the Web Using Composed Pages", Poster at ACM SIGIR 2006.
- [40] R. W. White, I. Ruthven and J. M. Jose: "Finding Relevant Documents using Top Ranking Sentences: An Evaluation of Two Alternative Schemes", ACM SIGIR 2002.



clinical databases.



Yannis Papakonstantinou. His main research work addresses the problem of bridging the gap between Databases and Information Retrieval. For more information, please visit <http://www.cs.fiu.edu/~vagelis/>.



bioinformatics. For more information, please visit <http://www.cs.fiu.edu/~taoli/>.

Ramakrishna Varadarajan received the B.E degree in computer science and engineering from the University of Madras, India in 2004 and the M.S degree in computer science from the Florida International University, Miami in 2006. He is currently a PhD candidate in the School of Computing and Information Sciences at the Florida International University in Miami. His current research interests include web search, ranked keyword search in graph databases and various domains like biological and

Vagelis Hristidis is an assistant professor at the School of Computing and Information Sciences at the Florida International University in Miami. He received his B.S. in Electrical and Computer Engineering from the National Technical University of Athens and his M.S. and Ph.D. in Computer Science from the University of California, San Diego in 2004 under the supervision of

Dr. Tao Li is currently an assistant professor in the School of Computing and Information Sciences at Florida International University in Miami. He received his Ph.D. in computer science from the Department of Computer Science, University of Rochester in 2004. His research interests are in data mining, machine learning, information retrieval, and