The Anatomy of a Domain-Specific Search-Phrase Suggestion Tool for Literature Digital Libraries

Sulieman Bani-Ahmad
Department of Information Technology. Al-Balqa Applied University, Jordan

Abstract *Suggesting search terms in keyword-based search interfaces is becoming common place. Google Suggester, for instance, in a history-based one as it provides suggestions to queries using a dynamic list of pre-saved popular queries. However, this approach suffers from accuracy and efficiency problems. In this paper, the anatomy of a large-scale efficient suggester for literature digital libraries is proposed. This suggester makes suggestion from the pre-analyzed document collection to be searched. The proposed suggester promises providing human-like suggestions that are automatically extracted from the repository of web documents, i.e. publications in the context of digital libraries webpages in the context of the web. The proposed tool also provides an accurate term ranker to assign importance scores to suggested keywords within the contexts where they are observed. The proposed tool promises a more scalable and user-friendly search-keyword suggester when compared to its history-based competitors.*

Keywords: *Keyword-based search interfaces. Search-keyword Suggester, Google Suggest, the CompleteSearch engine.*

Received March 7, 2010; Accepted March 9, 2011

1. Introduction

Search-phrase suggestion tools (SPST) help users to (i) formulate their queries, (ii) explore alternative spellings for their queries, and (iii) save keystrokes [13, 3, 2], which is important to less-experienced or older searchers. Studies show that web users spend considerable amounts of time in search sessions to properly select keywords [11], and to modify their search keywords in order to successfully locate documents that they are searching for. By utilizing an efficient SPST users are less likely to face unsuccessful search attempts. Figure 1 shows how Google Suggest, an innovative feature by Google, can be useful through suggesting terms for the user that helps him/her accurately aggregate focused search terms.

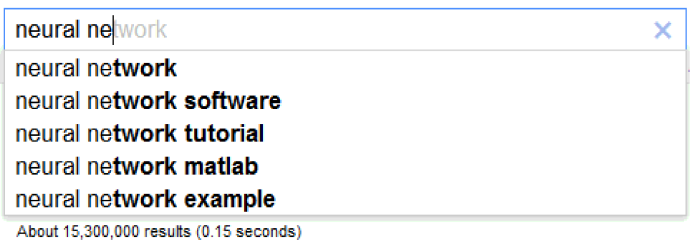

Figure 1. Sample Google suggest refinements term "*neural ne*".

In the case of web queries of search engines, frequently, users are not sure as to how to characterize the search using keywords [11], and gradually build more focused search keywords that do not cause the topic diffusion problem [7]. The topic diffusion problem occurs when the provided search terms are observed in multiple contexts. This fact caused that documents from multiple and relatively topics appear in the search results list.

One scenario where users find difficulty formulating their queries is when a search term has synonyms that the user does not remember. As an example, the "Big O notation", which is a mathematical notation used to describe the asymptotic behavior of functions, is also referred to as "Landau notation" or "asymptotic notation" [1]. Another scenario is when the same keyword has different meanings in different contexts, i.e., polysemy [8]. This may force the user to add more keywords to prune out irrelevant contexts. A possible approach to solve these problems is to provide users with immediate feedback on the digital library contents as well as on how focused their search keywords are, at an early stage, i.e., as they enter search keywords. In this paper, we propose and evaluate such the framework of a content-based SPST system.

In contrast with our approach, Google [2] provides an SPST through Google Suggest (Figure 1) which employs users' search-history repository. Studies show that this approach has multiple shortcomings that make it inadequate for the literature digital library domain as it affects both the way people search and what keywords they use [26].

Content-based (CB-) SPSTs, as opposed to Google's Search-History-Driven SPST, have recently received more attention [13, 4, 3]. In general, a CB-SPST anticipates users' search keywords by parsing

the document collection to be searched and preparing offline refinements to search keywords.

In this paper, we present the framework of a CB-SPST that improves the performance of the search-engine auto-completion tools. The proposed overcomes the shortcomings of Google Suggest.

2. An overview of the basic features of the proposed CB-SPST

A CB-SPST is based on an a priori analysis of the document collection to be searched and is optimized to fit literature digital libraries. The proposed SPST utilizes the following components:

1. An English language parser: to parse the document collection using the Link Grammar parser, a syntactic parser of English,
2. A domain-specific grouping tool for publications: to group the publication based on their "most-specific" research topics (using the notion of research pyramid [2]),
3. A Token ranking tool: we use TextRank, a text summarization tool, to assign topic-sensitive scores to keywords,
4. Suggestions focusing tool: we use the identified research topics to help user aggregate focused search keywords prior to actual search query execution.

2.1. The linguistic pre-processing step of the proposed SPST

In the *linguistic pre-processing step*, documents are tokenized to transform them into *tokens*. Later, when forming complex tokens, i.e., combining more than one token into one complex token, stopwords will be used to produce syntactically and semantically correct suggestions.

Example: Tokenizing the title *"The Linear Complexity of a Graph"* generates the following *simple* tokens (i) "the", "of" and "a" which are *stopwords*, and (ii) "linear", "complexity", and "graph" which are *non-stopwords* that are expected to appear in user search queries. Stopwords are useful in forming *compound tokens* through combining two or more simple tokens at a time. For instance "linear complexity" and "graph" can be linked using "of" to form the full title. Simple and compound tokens then serve as building blocks for expected user search keywords.

2.2. Building the token-hierarchy and making focused suggestions

The identified tokens are put into a *token hierarchy*. During a search query session, the proposed SPST gradually recommends search keywords by traversing this token hierarchy. During each search session, *the Single Token Anticipator*, STA, is used to make recommend autocompletions based on the first few letters entered by the user. The *suggestion scope* is continually reduced based on the previously fed terms within the same session. In the contect of literature digital libraries, the suggestion scope is defined as the set of most-specific research "topics" where suggestions are extracted. The SPST guides the current user towards building focused search keywords. In our case, research topics are represented by *research pyramids*, where a research pyramid is a set of publications that are related to the same research topic [2]. The token hierarchy involves the following levels

1. The single token level.
2. The keyphrases (compound token level)
3. The publication title level
4. The research pyramid level, each research pyramid represents a specific research topic.

The proposed SPST uses the identified research-pyramid structures to assign topic-sensitive significance score to tokens and refinements. This helps to propose refinements from research topics where the user's entered keywords are of most significance.

The CompleteSearch engine SPST that is proposed in [13, 3] works by building and index named HYB. HYB is prepared by preprocessing the document collection to pre-compute inverted lists of *compound tokens*. Compound tokens are identified using proximity measures between words separated by w, that is, the pre-determined window size. To maintain a good level of locality of search, *similar* words are placed in the same block within the index in the form of document-word pairs. As the user enters his search words, relevant blocks, i.e., blocks where search terms are observed, are identified and, thus, (searching) scope, or context, narrows down to only relevant documents.

2.3 Summary- Comparing the proposed SPST to its competitors

In summary, the main contribution of this paper is to design and evaluate a CB-SPST that (i) eliminates the drawbacks of Google's search history-based SPST, and (ii) boosts the performance of the techniques used in the CompleteSearch.

Since our proposal extends CompleteSearch suggester, our approach maintains all the advantages of CompleteSearch. For instance, our approach has an excellent *locality of access*. Moreover, the completion of subwords and phrases is automatically supported since phrases are linguistically pre-computed.

Finally, our proposed tool does not only help user refine his/her search terms, but also helps make search keywords more specific, and thus reduces the number of irrelevant documents appearing in the result set.

3. Design Principles of the CB-SPST

The CB-SPST problem involves the anticipation of the search keywords that the user is attempting to specify. An SK-Suggestion query is a 5-tuple Q(W, I, R, β_s, β_p), where W is all possible completions of the last word that the user started typing, and R and I are the sets of relevant topics (research pyramids) and compound tokens from the preceding query. β_s and β_p are thresholds for the maximum scope and the minimum popularity required to control the number of suggestions made available to the user. Processing query Q involves the following steps: (i) compute the subsets W' of W, and a word in W' that occurs in at least one compound token in I, (ii) compute R' and I' that form the set dominant research topics and compound tokens respectively, where I' $\subseteq$ I and R' $\subseteq$ R. Alternatively, the user may choose to be shown a fixed number of suggestions as in Google Suggest and the CompleteSearch engine, in which case the query becomes a 4-tuple of the form Q(W, I, R, β_k).The main design goals of the proposed SPST are:

- *The SK-Suggester should provide instant feedback to users prior to query execution.* The primary goal of the proposed SK-Suggester is to help focus user's search term to what is already available prior to performing the search, and thus reduce the time spent on search failure. To meet this goal, the CB-SPST should provide instant feedback as to how focused the search keywords are to the user, prior to query execution.
- *The SPST should suggest linguistically valid search keywords*. For that, we utilize an English language parser to tokenize and parse the digital library collection contents, and to build linguistically valid search keywords. An alternative approach, used in the CompleteSearch engine [13, 3], is to show the user snippets of text; however, this approach needs preprocessing and more effort by the user to interpret them.
- *The SPST should provide guidance to the user, as (s)he builds up the search keywords.* We achieve this by providing statistics on the search output prior to search execution. The proposed SPST provides the user with (i) the *scope* of each of the search keywords (i.e. the set of papers to be returned), which also warns the user against keywords that are very common and may lead to large search outputs. Notice that the number of documents where search terms are observed is not a good indicator of how focused search keywords are. Given that the user is interested in a particular research topic, some research topics (represented as research pyramids) are large because many researchers are working on that topic [2], and thus large numbers of documents may be found relevant to a user query. Consequently, the fact that a search keyword is observed in large numbers of documents does not necessarily indicate that the keywords are not focused enough. A better indicator of how focused search keywords are, is the number of relevant research topics, which is the number of research pyramids.
- *The SPST should work online efficiently, and suggest refinements to keywords on the fly.* For efficiency, we need to parse only properly selected parts of each document in the collection, and recognizes nouns, adjectives and verbs a priori. Further, all of the time consuming tasks are performed offline.

4. Constructing Token Hierarchy

4.1. The Linguistic pre-processing step

We first present a number of natural language (English Language properties and definitions that will be used throughout this paper.

English Language property 1: *Simple Sentence types* include (1) *declarative*, (2) *interrogative*, (3) *imperative*, and (4) *conditional types*. Compound sentences have the format "<simple sentence> <conjunction> <simple sentence>". Declarative sentences consist of a *subject* and a *predicate*. Subject may be *simple* (i.e., consists of a noun phrase or nominative personal pronoun) or *compound* (i.e., consists of multiple subjects combined with conjunctions).

Figure 2 shows the parser output for the title *"Outlier detection for high dimensional data"* with multiple linkages identified within the title. Each linkage represents a linguistic relationship between two tokens (see English Language Property 3 next). A full list of linkages that the parser can identify is available in [12]; however, only a few of them are common and observable in publication titles.

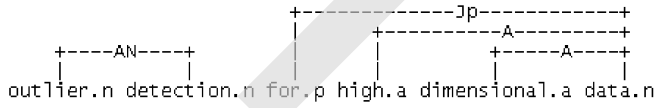

Figure 2. Link-grammar parser output.

To suggest linguistically valid keywords, we utilize the linkages identified by the parser to form compound tokens out of simple tokens. The following definitions and observations form the basis of our discussion on how the token hierarchy is built.

Definition: A *simple token* is a categorized block of text consisting of indivisible characters. A *compound token* is a linguistically valid combination of one or more *simple tokens*.

As an example, "sort", "merge" and "join" are simple tokens. "sort-merge" and "sort-merge-join" are linguistically valid compound tokens; but, "join sort-

merge" is linguistically invalid as the adjective should precede the noun in English.

Note that, not all linguistically valid compound tokens are "observed" in a digital library. For instance, "merge-sort join" is linguistically valid, but there is no such join algorithm in the data management field. We will refer to linguistically valid, but not necessarily observed, compound tokens as *unrealistic* compound tokens.

English Language property 2: *Part-of-speech token types* include (1) *articles*, (2) *nouns* (*subjects* or *objects*), (3) *adjectives*, (4) *adverbs*, (5) *pronouns*, (6) *conjunctions*, (7) *verbs*, and (8) *prepositions*.

To form realistic compound tokens, we identify *part-of-speech tokens* that are linguistically adjacent. The goal is to make keyword suggestions that make sense to the user. We use the link-grammar-based parser proposed in [12] to identify linguistically adjacent tokens and build the token hierarchy of the publication set.

English Language property 3: Possible *linguistically adjacent or related token type* cases include (1) (*subject, verb*), (2) (*verb, object*), (3) (*adjective, noun*), (4) *Compound subjects*, (5) *Compound objects* (6) (*noun-possessive, noun*), (7) (*article, noun*), (8) (*adverb, verb*).

Example: the title "Adaptive Rank-Aware Query Optimization in Relational Databases" has the following compound tokens. (i) (Adjective, noun): "relational databases", (ii) (compound adjective): "rank-aware", (iii) (adjective, noun) "adaptive query", (4) "query optimization". Note that more complicated combinations of tokens are also possible, e.g., (i) (compound subject, verb), (ii) (verb, compound object), or (iii) (simple subject, verb, object).

A full list of all linkage-types can be found in [12]. Building blocks of the token hierarchy are *nouns*, *adjectives* and *verbal nouns* (which are sometimes identified as *verbs* or *gerands* by the parser). These linkages are used to build meaningful compound tokens. The token hierarchy is constructed by collapsing the observed linguistically adjacent tokens into compound tokens. Any two tokens that are linked via a linkage are considered to be linguistically adjacent even if they are separated by other tokens or stopwords. A *super-linkage*, that is, a linkage that encompasses one or more linkages, is used to construct further compound tokens. We give an example.

Example (using linkages to form recommendations): Figure 3 shows the parser results for the title "*a model for querying annotated documents*". Two levels of linkages are identified: (i) the 'A' linkage is at level 1 and used to form the compound token "*annotated documents*", (ii) the super-linkage 'OP' at level 2 (which encompasses the 'A' linkage) is used to form the compound token "*querying annotated documents*".

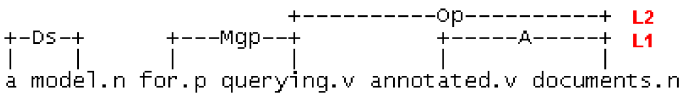

Figure 3. levels of linkages and super-linkages.

4.2. Topic-Sensitive Token Weight and reducing the topic diffusion problem

Each user search session can be viewed as aiming at finding information about a specific topic. This implies that the user's suggestions of search keywords should be chosen as close to the topic being targeted as possible. However, the topic being targeted is unknown to us.

If the SPST fails to focus search terms and choose them from most related research topics, the search results of the digital library will most-likely be topically diffused. Consider the following example (figure 4). The following figure shows a sample search result for the query 'topic diffusion problem in information retrieval' run against ACM digital library. The output shows only the top 200 relevant publications, and the relevancy score of all retrieved publications are the same. However, not all of them are related to the search terms provided by the user possibly because the proposed search keywords provided by user involve multiple sub-topics mixed together.

Figure 4 A query that illustrates the topic diffusion problem.

To remedy this problem, the proposed SPST uses the already entered search keywords to prune out topics (i.e. research pyramids) where these keywords are not or rarely observed. We refer to this phenomenon as the *locality principle of search*.

Observation: (The *locality* principle *of search*): Within a single search session, the user targets documents within a specific topic.

This principle allows us to narrow down the suggestion scope as the user enters more search keywords. The token-hierarchy relation is then accessed once in between keystrokes each time the user modifies the search keywords by typing one more character. Given our hypothesis that the document(s) that the user is looking for belongs to a specific research topic or few related topics, we can reduce dramatically the diversity of the collection set by suggesting keywords from the *most relevant* research topic(s), which we refer to as the *suggestion scope*.
One issue is that a term may be used in more than one research topic. To solve this problem, we weigh tokens within each research topic. The goal is to identify the significance of tokens in each research topic, and thus prevent search-keyword refinement from topics where keywords are of lesser significance.

To weigh tokens in each research-pyramid, we use the TextRank algorithm [9]. Briefly, TextRank algorithm constructs a graph between a properly selected set of tokens of a document (nouns and adjectives), where an edge between two tokens exists only when they co-appear together in a window of some size. Then we apply the PageRank algorithm on the formed graph to identify the most important tokens. PageRank is an algorithm applied on graphs to measure relative importances of vertices [5]. Finally, phrases are manually constructed out of the top-scored tokens; these phrases represent keyphrases of the document.

We use TextRank at research pyramid level to compute topic-sensitive significance score of terms. We apply TextRank on each research pyramid *r* as follows. (i) The titles of all publications that belong to *r* are tokenized and annotated with part-of-speech tags using the link-grammar parser [12]. (ii) The tokens are filtered through a syntactic filter which selects only lexical units of certain parts of speech, namely; *nouns (*as well as *verbal nouns* and *gerands)* and *adjectives*, that give the best results [9]. (iii) A graph $G_r(V,E)$ is formed using the tokens returned by the filter. *V*, or the set of vertices, is the set of tokens. E, i.e. the edge list, is constructed such that an edge is created between any two tokens that appear in the same title. (iv) PageRank [5] is used to measure relative importance of all tokens. Tokens that have high PageRank scores are expected to be more significant and better representatives of the research pyramid *r*.

5. Suggesting Search Keywords

The task of recommending search terms and autocompletions is performed online; thus, real-time performance is critical. Notice that and SPST is triggered online "in-between keystrokes". After each keystroke, the search terms already entered can be passed to CB-SPST through an AJAX-enabled interface. The Single-Token-Anticipation (STA) and the Query-Refinement (QR) Modules of the CB-SPST at the server side process these search terms. An AJAX-enabled search interface is needed in this case in order to provide an immediate, flexible, and responsive interaction [14, 4]. In more details, online steps of our approach are:

1. Single token anticipation. The STA Module is triggered each time the user starts entering a new search keyword. This module suggests completions to the incomplete term entered by the user from the current suggestion scope (by using R and I in definition 1). At the beginning, the suggestion scope is all the research pyramids and all the compound tokens, which is the most time-consuming step [3].
2. Search keyword refinement suggestion. The QR Module suggests the top-scored compound tokens I to the user as possible refinements to the user's search terms.
3. Focusing suggestion scope: In this step, the subsets R' and I' are computed and saved in the *search session status structure* to be used in query refinements after the next keystroke. This task is responsibility of the feedback module.
4. Post-processing suggestions: the Presentation module of the CB-SPST is responsible of properly presenting the suggestions to the user. This specific task is performed at the client-side. This should make the search suggester to be more scalable.

5.1. Important performance-related notes

Next, we list and discuss the advantages of our proposed framework as compared to the autocompletion tool proposed in [3]:

1. Tokens, simple and compound, observed in the same research pyramid, or multiple strongly related research pyramids, are stored within the same block as in [3]. This gives better locality of search and reduces I/O operations especially after suggestion scope reduces to few relevant research pyramids.
2. A term may be used in more than one research topic. To solve this problem, we weigh tokens within each research topic. The goal is to identify the significance of tokens in each research topic, and thus prevent search-keyword refinement from topics where keywords are of lesser significance.
3. In [3], suggestions are presented to the user as snippets of text from the documents in the LDL. This puts an extra burden on the user to isolate useful information from the presented text. Users usually type fast and may not have enough time for post-processing the presented suggestions. In our case, the repository is linguistically preprocessed to identify compound tokens, or compound tokens,

that will be presented to the user isolated from the surrounding text.

4. Scalability: in order to suggest phrases instead of single words, Bast and Weber [3] use text-based adjacency (within a predetermined window size) as an indicator of token-to-token proximity. We observed that this proximity measure generates long lists of possible phrases which (a) significantly increase the index size [3]; this problem is solved by viewing the auto-completion problem as a multi-dimensional range searching problem [3] and (b) may result in meaningless phrases. Our proposal uses linguistic adjacency (see English Language property 3) which produces meaningful and much smaller lists. Consequently, our approach is more scalable.

In order to match completions with the being entered query word, Bast and Weber [3] store the positions of terms within each document in an array separate from the index. We refer to this technique by the text-based. Online processing of this extra array takes time. In our case, we use linguistic *linkage-based proximity* of tokens to build compound tokens. This gives more realistic and better results as tokens (nouns and adjectives, for instance) may be separated by intermediate words but still linguistically related. Thus, depending on the assigned *proximity window*, some close terms may be missed in the case of small widow sizes, or false positives may appear in the case of big window sizes. Our approach, in some sense, uses proximity windows with variable sizes.

5.2. CB-SPST Query Execution

Compound tokens are used as refinements to user queries, and vary in their sizes. To avoid proposing a long suggestion, compared to user search terms, we propose a gradual expansion of the user query as follows. Given user's search terms W, refinements of length up to ***ef**** |W| are presented to the user, where ***ef*** is the expansion factor, and |W| is the number of tokens in W.

We empirically observed that initially choosing the expansion factor to be 1.5 gives good results allow for a gradual expansion during user's search term construction. However, when the user chooses terms that are separated by a relatively large distance, i.e., separated by long series of words which is the case in large compound tokens, a particular choice of an expansion factor may fail to retrieve refinements. To remedy this problem, we propose dynamically choosing ***ef*** through probing as follows. First, we choose ***ef***=*1.5*. If no suggestions can be retrieved, the value of ***ef*** is increased up to $\boldsymbol{ef}_{\max}$ which is chosen as the length of largest compound token observed. The amount by which ***ef*** is increased is left to the LDL server to estimate, based on how many online users are available and whether real-time performance is achieved or not.

5.3. Guiding Statistics

Next we present a list of statistics that are used to guide the user selection of search keywords.

Suggestion Scope: Research Pyramid based suggestion scope considers the number of research topics (or research pyramids) where the search keywords ***w*** are observed, that is, the scope is

$$\text{Scope}(w)=(\#\text{ of RPs where } \boldsymbol{w} \text{ appears})$$

Topic-Sensitive Popularity of Search Keywords: For a set of words (W'), the topic-sensitive popularity of W' with respect to the topic represented by some research pyramid **r**, i.e., TSP(W', **r**), is computed as the sum of TextRank scores of all words in W'. TextRank scores are topic-sensitive and computed within each research pyramid **r**. The suggestions are retrieved from dominant research pyramids computed by the feedback module in figure 8.

Query refinements (W') are presented to the user in the order of their *matching scores*.

One more statistic used is the *Specifity of Individual terms*. Specifity of token *t* is measured as

$$\text{Specifity}(t)=-\log[(\#\text{ of Docs where t appears})/(\text{total \# of Docs})]$$

We use this number to color user's already entered terms to indicate how general his/her individual search terms are. This helps when user's search terms consist of stopwords or terms used in a wide range of research topics.

6. Experimental Results

Our experiments are conducted on a prototype digital library with a repository of around 15,000 publications from ACM SIGMOD Anthology, a digital library from the field of data management. Publications titles of this collection were analyzed and parsed by the Link-Grammar parser, a syntactic parser of English, that is developed at Carnegie Mellon University and proposed in [11, 12].

6.1. Linguistic Pre-processing and Quality of Suggestions

In the following example, we show how the proposed SK suggester also serves in early construction of successful search keywords for *k-word proximity search,* which is a very useful technique in narrowing down the results to more relevant ones, and at the same time allowing users to better express what they are looking for [6, 10].

Example (k-word *Proximity Search)*: In figure 5, notice that the search keywords "query graph" are already identified as one compound token. Suggesting

query refinements based on compound tokens may help towards a successful proximity search. Notice that item (3) is probably irrelevant to the query at search time since this publication most probably belongs to different research pyramid from the first and second hits; this false positive is pruned or pushed down in ranking query results. Informing users of the linguistic proximity of search terms prior to query execution can thus be useful. Furthermore, informing the user of the order in which terms appear may help eliminate false hits like hit (4) in figure 5, which is called k-word ordered proximity search [6].

(1) Multiple Query Processing In Deductive Databases Using Query Graphs
(2) Query Graphs Implementing Trees And Freely Reorderable Outerjoins
(3) Effective Graph Clustering For Path Queries In Digital Map Databases
(4) Query By Diagram, A Graphic Query System

Figure 5. Possible hits of the query "query graph".

*Linkage*s are used to construct compound tokens. This technique generates significantly smaller numbers of constructs than the text-based adjacency used in Bast and Weber [3]. For instance, by parsing the titles of around 9 thousand publications from the repository, 5,652 tokens were retrieved (6,896 tokens including stopwords). And, around 5 thousand compound tokens are constructed. Considering text-based adjacency using the same window sizes generated 220,000 of links between tokens. These links are to be processed further to identify the most significant compound tokens.

The value of using linguistic pre-processing to identify compound tokens comes from the quality of pre-computed compound tokens that can be constructed. Along with the post-processing required by the text-based adjacency approach, both factors balance the time needed to perform the linguistic pre-processing step (which is done offline).

6.2. Convergence of Suggestion Scope

One more factor that is critical in producing search-keyword suggestions in realtime is the locality principle of search and the convergence speed of the suggestion scope.

Figure 6 shows the distribution of scope of the observed filtered tokens, i.e., excluding the stopwords.

Observation (figure 6)**:** Filtered tokens have limited scope.

The above observation is important as it significantly affects the QR module performance. Considering this observation, we may prune the scope of the suggestions, which is the set of dominant research pyramids from where suggestions are retrieved.

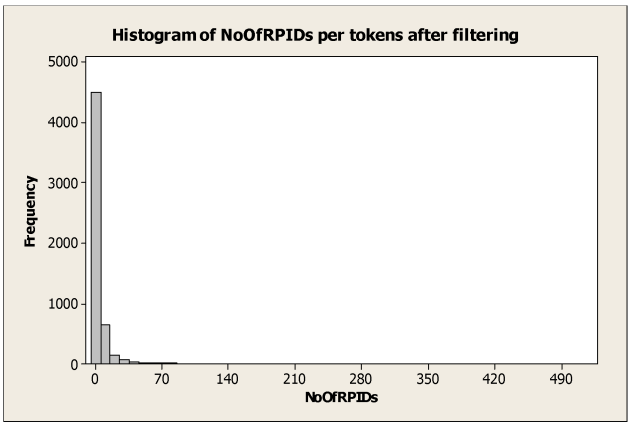

Figure 6. Distribution of token scope (after filtering).

Notice that some research topics may have wide range of origins. This makes the diversity of terms used in such research topics wide as well. For example, the publication "TextRank: Bringing order into text" has origins in linguistics (tokenizing and parsing), graph theory and graph-based ranking.

To measure how wide and diverse the origin of the research topic (or a research pyramid) **r** is, we use the notion of *RP* coverage computed as follows:

$$\text{Coverage}(\mathbf{r}) = -\log[(\#\text{ of tokens used in } \mathbf{r})/(\text{total } \#\text{ of tokens})]$$

This means that the higher the coverage factor of **r** is, the less diverse its tokens become. Zero coverage of **r** indicates that all tokens ever observed in the collection are used in **r**.

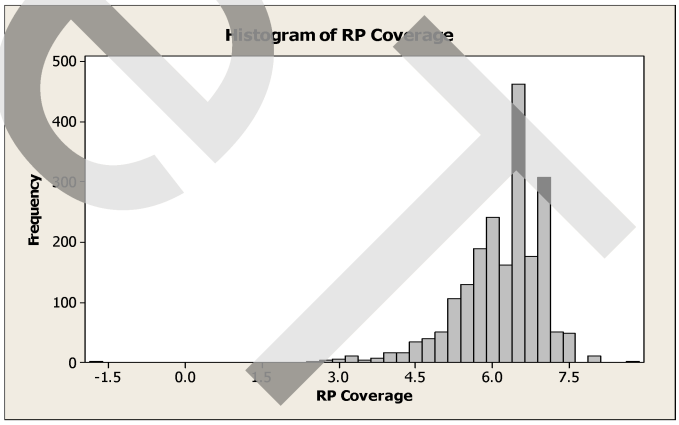

Figure 7. Distribution of RP coverage.

Figure 7 shows the distribution of coverage values of all research pyramids. Coverage values range between 3 and 8, which means that (i) the tokens within each research pyramid are of low diversity, and (ii) this signifies the importance of ranking tokens within research topics. This serves in pushing refinements extracted from dominant research topic(s) up in the suggestion list. We achieve this goal by using the topic-sensitive popularity (TSP) of search terms to order the list of computed refinements.

One critical factor that affects the STA module performance is the speed of convergence of the

suggestion scope at the beginning of each search session.

We have experimentally observed that, usually within 3 characters entered by the user, the suggestion scope significantly decreases. We have also observed that the suggestion scope of STA reaches a saturation region within 4 characters entered by the user.

Figure 8 shows the distribution of specifity values of all tokens extracted from the document collection. High specifity value of a token ***t*** indicates that ***t*** is observed in many. Figure 8 shows that high percentage of observed tokens have high specifity values, and thus, may lead to large search output lists. This further signifies the importance of warning users against such popular tokens and encourage him/her properly choose tokens of lesser specifity values.

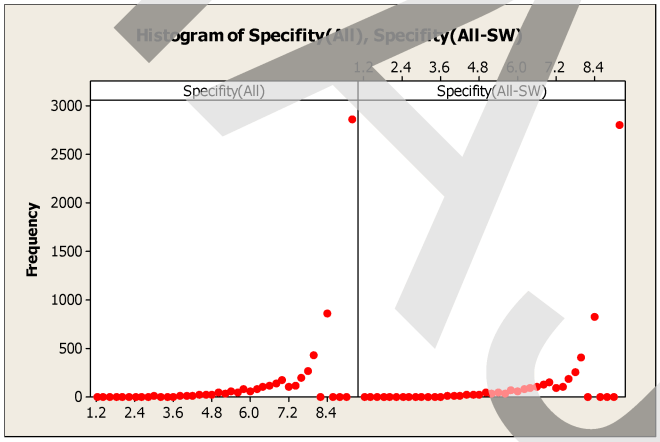

Figure 8. Distribution of specifity values of (a) all tokens (left) and (b) all tokens except stopwords.

Figure 9 shows TextRank score distribution of all simple tokens that pass through the syntactic filter. High TextRank scores indicate popular and more significant tokens. Thus, tokens that score high (>0.8) are content-bearing and better represent the research topic of the corresponding research pyramid. At the other extreme, low-scored tokens are usually widely used tokens.

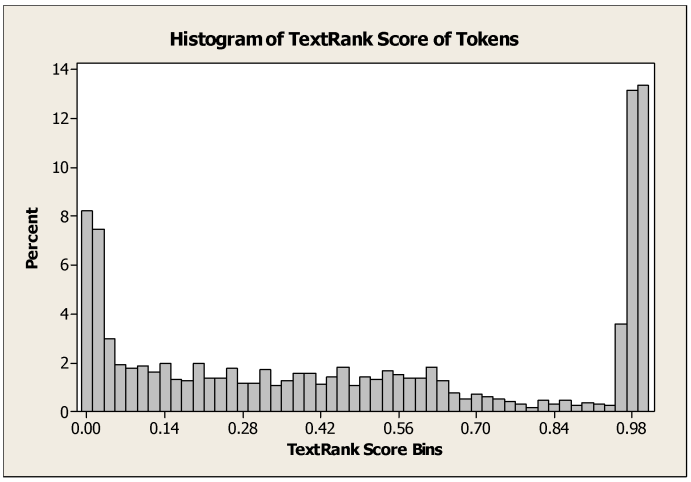

Figure 9. Distribution of TextRank scores for simple tokens.

We use topic sensitive popularity scores of tokens to order computed refinements such that the most relevant refinement from dominant research pyramids appear close to the top of the suggestion-list.

Since the post-processing module processes the final selected list of single token completions and the computed refinements, it is scalable and takes constant time to finalize the suggester output in the proper HTML format. Further, this module can be run at the client side using client-side scripting language.

7. Conclusions

We have proposed a content-driven search-keyword suggester and autocompletion tool. We have shown that the proposed framework, which is optimized to work on literature digital libraries, promises a more scalable, high quality, and user-friendly search-keyword suggester when compared to its competitors.

References

[1] Bani-Ahmad, S. and Ozsoyoglu, G.: "elGiza, A Research-Pyramid Based Search Tool for Vertical Literature Digital Libraries", DBRank 2008. http://dx.doi.org/10.1109/ICDEW.2007.4401002

[2] Bani-Ahmad, S. and Ozsoyoglu, T: "Improved Publication Scores for Online Digital Libraries via Research Pyramids", ECDL 2007.

[3] Bast, H. and Weber, I.: "Type less, find more: fast autocompletion search with a succinct index". SIGIR 2006: 364-371.

[4] Bast, H.; Majumdar, D.; and Weber, I.: "Efficient interactive query expansion with complete search". CIKM '07.

[5] Brin, S. and Page, L.: "The anatomy of a large-scale hypertextual web search engine", Computer Networks and ISDN Systems, 1998.

[6] Gupta, C.: "Efficient K-Word Proximity Search", MS Thesis, CWRU, EECS Department, 2008.

[7] iProspect Inc.: "iProspect Search Engine User Behavior study", iProspect 2006.

[8] Krovetz, R.: "Homonymy and polysemy in information retrieval". In Proceedings of the Eighth Conference on European Chapter of the Association for Computational Linguistics (Madrid, Spain, July 07 - 12, 1997). European Chapter Meeting of the ACL.

[9] Mihalcea, R. and Tarau, P.: "TextRank: Bringing Order into Texts", in Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2004), Barcelona, Spain, July 2004.

[10] Sadakane, K. and Imai, H.: "Text Retrieval by Using k-word Proximity Search,", 1999 International Symposium on Database

Applications in Non-Traditional Environments (DANTE'99), 1999.

[11] Sisson D.; "A Thoughtful Approach to Web Site Quality", http://www.philosophe.com/

[12] Temperley, D.; Sleator, D., Lafferty J.; "Link Gramar Parser 4.1" – http://www.link.cs.cmu.edu/link, 2005.

[13] The CompleteSearch engine, http://search.mpi-inf.mpg.de/

Wusteman J. and O'hIceadha P.: "Using Ajax to Empower Dynamic Searching", Information Technolgy and Libraries, Vol. 25 No 2, June 2006, PP 57-64.

Sulieman Bani-Ahmad has received his B.Sc. degree in Electrical Engineering / Computer Engineering from the department of Electrical Engineering Jordan University of Science and technology in 1999. He received an MS in Computer Science from the school of Information Technology at Al-Albayt University in Jordan, in 2001. He received his Ph.D. degree in Computing and Information Systems from the department of Electrical Engineering and Computer Science at Case Western Reserve University, Cleveland - Ohio, USA, in 2008. He is presently an assistant professor at Al-Balqa Applied University. Bani-Ahmad's research interests cover topics from Web-computing and high performance computing.