WORDGRAPH: Keyword-in-Context Visualization for NETSPEAK's Wildcard Search

Patrick Riehmann, Henning Gruendl, Martin Potthast, Martin Trenkmann, Benno Stein, and Bernd Froehlich, *Member*, *IEEE Computer Society*

Abstract—The WORDGRAPH helps writers in visually choosing phrases while writing a text. It checks for the commonness of phrases and allows for the retrieval of alternatives by means of wildcard queries. To support such queries, we implement a scalable retrieval engine, which returns high-quality results within milliseconds using a probabilistic retrieval strategy. The results are displayed as WORDGRAPH visualization or as a textual list. The graphical interface provides an effective means for interactive exploration of search results using filter techniques, query expansion, and navigation. Our observations indicate that, of three investigated retrieval tasks, the textual interface is sufficient for the phrase verification task, wherein both interfaces support context-sensitive word choice, and the WORDGRAPH best supports the exploration of a phrase's context or the underlying corpus. Our user study confirms these observations and shows that WORDGRAPH is generally the preferred interface over the textual result list for queries containing multiple wildcards.

Index Terms—Information visualization, visual queries, text visualization, information retrieval, Web n-grams, wildcard search.

1 INTRODUCTION

THE WORDGRAPH is a visual tool for context-sensitive word choice. It allows its users to check the commonness of a phrase and to express uncertainties in choosing words by formulating queries that contain wildcards. The search results are transformed to a legible and interactive graph visualization—the WORDGRAPH (see Fig. 1). The graph layers follow the structure of a query, showing one layer for every literal word and wildcard, which are filled dynamically with the results obtained from NETSPEAK's retrieval engine. Search results visualized by the WORDGRAPH can be interactively explored, refined, and expanded by means of filter techniques and navigation. Queries are processed by a scalable retrieval engine called NETSPEAK, which returns high-quality results within milliseconds using a probabilistic retrieval strategy. It indexes a corpus of more than 3 billion word n-grams up to a length of $n = 5$ words along with their occurrence frequencies in a large portion of the Web.

WORDGRAPH's intended audiences are people who have doubts about how certain phrases are commonly formed. In particular, second language speakers face difficulties in this respect, since their innate sense of language—their sprachgefühl—is often not sufficiently developed. They might ask themselves how others would formulate a particular phrase; a piece of information that is generally hard to come by. NETSPEAK implements a statistical solution by contrasting alternative phrases based on their absolute and relative occurrence frequency. Our working hypothesis is that choosing more common phrases over uncommon ones may improve readability, comprehensibility, and writing style. Obviously, this is not true in general, but as nonnative speakers we found NETSPEAK's suggestions immensely helpful in all our daily writing tasks.

A variety of visualizations of relations among words, phrases, or collocations (also called "keywords in context") have appeared in recent years, such as the WORDTREE, PHRASENETS [1], Google Scribe, and AWKCHECKER, to name only a few. The WORDTREE [2] employs suffix trees to index text and to visualize the tree starting from the query word(s). PHRASENETS encode subject-predicate-object triplets in a directed graph, which are mined from a text by specifying the predicate and considering subject and object as wildcards (e.g., `? loves ?`). Google Scribe [3] assists authors with writing by suggesting the next word in a phrase—very similar to AWKCHECKER [4]. Both systems are (likely to be) based on language model theory.

The WORDGRAPH visualization and the NETSPEAK engine can be considered as a generalization and a combination of the aforementioned approaches: the WORDGRAPH has more complex layout constraints than the tree layout of the WORDTREE. PHRASENETS visualize only triplets with a fixed predicate and the force-based layout limits the phrase legibility. Both tools focus on corpus exploration instead of being interactive word choice tools. AWKCHECKER and Google Scribe do not employ visualizations at all, as they are not intended for ad hoc wildcard queries.

This paper is an extended version of [5]. The contributions are threefold. First, we present the WORDGRAPH, a dynamic graph visualization for interactive exploration of search results for complex keywords-in-context queries. Second, we introduce our new and scalable NETSPEAK retrieval

- *P. Riehmann, H. Gruendl, and B. Froehlich are with the Virtual Reality Systems Group, Faculty of Media, Bauhaus-Universität Weimar, Bauhausstrasse 11, Weimar 99421, Germany. E-mail: {patrick.riehmann, henning.gruendl, bernd.froehlich}@uni-weimar.de.*
- *M. Potthast, M. Trenkmann, and B. Stein are with the Web Technology and Information Systems Group, Faculty of Media, Bauhaus-Universität Weimar, Bauhausstrasse 11, Weimar 99421, Germany. E-mail: {martin.potthast, martin.trenkmann, benno.stein}@uni-weimar.de.*

Manuscript received 13 June 2011; revised 13 Jan. 2012; accepted 15 Mar. 2012; published online 22 Mar. 2012.
Recommended for acceptance by G. Di Battista, J.-D. Fekete, and H. Qu.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCGSI-2011-06-0129.
Digital Object Identifier no. 10.1109/TVCG.2012.96.

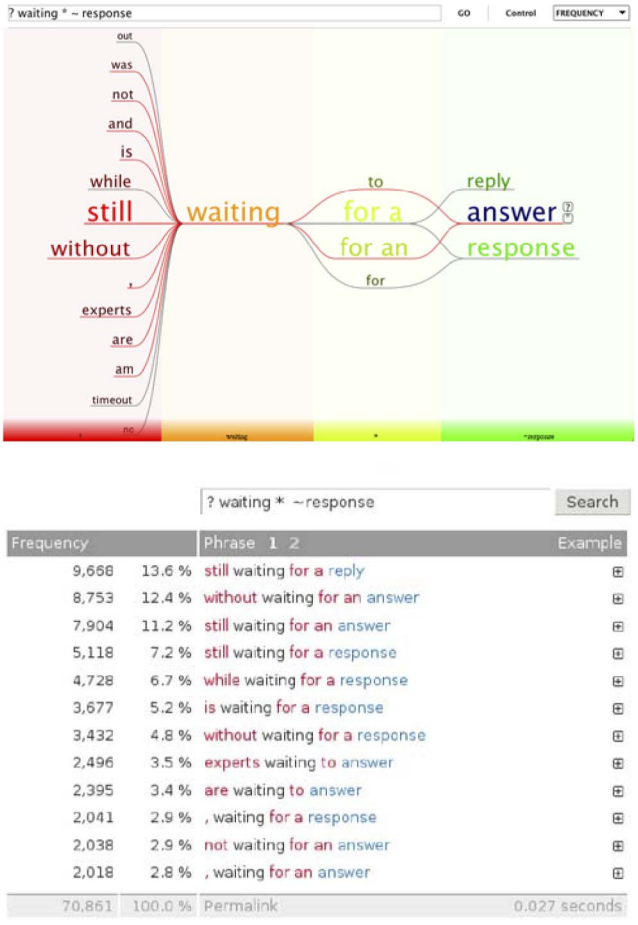

Fig. 1. The WORDGRAPH visualization (above) and the textual Web interface (below).

engine that operates efficiently on a large corpus of text from the Web. Lastly, we perform a user study comparing WORDGRAPH visualization and the textual Web interface, analyze query logs of the NETSPEAK service and investigate typical retrieval tasks related to choosing words.

2 Related Work

In addition to the aforementioned systems, several other keyword-in-context tools exist or are being developed. Viégas and Wattenberg present the Web Seer prototype [6], which allows one to contrast the query suggestions of the Google Web search engine for two different queries. The visualization encompasses two trees whose roots represent one of the queries each, while the children represent the suggestions obtained from Google. Shared suggestions are unified, thus visualizing tree similarity, while edge thickness and node positions tell something about how often a suggested query has been posed. Paley's Textarc [7] visualizes the sentences of a text centrifugal along an ellipse shape. Frequent words of the text are depicted inside the ellipse. The legibility of individual phrases is limited with this approach.

C. Harrison [8] has generated static word graphs from small portions of the Google n-gram corpus as showcase examples. However, no means is provided to generate these visualizations on demand, and it also lacks interaction so that it can only be viewed as it is. Collins et al. [9] visualize the text produced by automatic machine translation tools in the form of lattice graphs in order to support translators. Uncertainties of the tools in choosing the right translation for a word are represented by alternative paths in the lattice graph, where the commonness of an alternative, as determined by the language model underlying the machine translator, is encoded by size and shade of the nodes and their edges. Here, however, no manual wildcard queries are possible. Although our system displays a graph instead of a tree, the Degree-of-Interest Trees (DOITrees) by Heer and Card [10] and the SpaceTree by Plaisant et al. [11] provide some convenient patterns for the navigation of different levels of detail, supported by animated transitions in huge tree structures.

Corpora of n-grams are frequently used in natural language processing and information retrieval in order to support computational linguistics [12], such as data-driven error correction [13] or query segmentation [14]. While n-grams are usually exploited in a preprocessing fashion or to fully automate an analysis task, with NETSPEAK we have been among the first to consider literal n-grams by offering them directly as search results to the user. To the best of our knowledge, only the recently published Google Books n-Gram Viewer goes into a similar direction [15]. This viewer targets researchers in the humanities who study language use over time. However, it does not offer wildcard search capabilities, and its interface requires expert knowledge.

Search engines for linguistics that provide query operators comparable to NETSPEAK include WEBASCORPUS [16], WEBCORP [17], PHRASESINENGLISH [18], and LSE [19]. Cafarella and Etzioni [20] implement a search engine that allows to formulate parts-of-speech queries. The aforementioned approaches target linguistics researchers, for whom scalability as well as performance of the implemented indexes is only of secondary concern. Other related work can be found in the field of string processing where researchers study the scale-up of regular expression search for large text databases [21]. This body of work, however, aims at full text search, allowing for complete regular expressions, which brings about various problems in terms of runtime complexity and space efficiency. Again, the user of a regular expression search engine is different from ours. Since NETSPEAK and its visualization target the casual and average writer, we put strong emphasis on performance while focusing on a reasonable palette of search options to formulate queries against a database of short phrases. These constraints are exploited within our retrieval algorithms, and we are the first to study efficient wildcard search on n-gram databases.

3 Netspeak

The NETSPEAK text interface provides a straightforward way to search for phrases [22]. It is designed to conform to current and traditional best practices of Web interfaces for search engines, with an emphasis on simplicity and minimalism (Fig. 1, bottom). It utilizes a query language that is defined by the grammar shown in Table 1.

A query is a sequence of literal words and wildcard operators, wherein the literal words must occur in the expression sought after, while the wildcard operators allow specification of uncertainties. Currently five operators are supported: the question mark, which matches exactly one

TABLE 1
EBNF Grammar of the NETSPEAK Query Language

query	=	$\{$ word | wildcard $\}_1^5$
word	=	([apostrophe] (letter { alpha })) | ","
letter	=	"a" | ... | "z" | "A" | ... | "Z"
alpha	=	letter | "0" | ... | "9"
apostrophe	=	"'"
wildcard	=	"?" | "*" | synonyms | multiset | optionset
synonyms	=	"~" word
multiset	=	"{" word { word } "}"
optionset	=	"[" word { word } "]"

word; the asterisk, which matches any sequence of words; the tilde sign in front of a word, which matches any of the word's synonyms; the multiset operator, which matches any ordering of the enumerated words; and the optionset operator, which matches any one word from a list of options. The interface displays the search results for the given query as a ranked list of phrases, ordered by decreasing occurrence of absolute and relative frequencies. This way, the user can be more confident when choosing a particular phrase by judging both its absolute and relative frequencies. For example, a phrase may have a low relative frequency but a high absolute frequency, or vice versa, which in both cases indicates that the phrase is not the worst of all choices. Furthermore, the textual Web interface offers example sentences for each phrase, which are retrieved on demand when clicking on the plus sign next to a phrase. This allows users who are still in doubt to get an idea of the larger context of a phrase.

The NETSPEAK web service has been publicly available with a textual interface since 2008. The analysis of 50,000 queries of NETSPEAK's log files reveals that the average query length (words or wildcards) was 3.3 tokens (see Table 2). The most used wildcards are the asterisk and the question mark (over 90 percent), the synonym-operator is used in less than 5 percent of the queries and optionset as well as multiset are hardly used. Interestingly, the fraction of queries that do not contain any wildcards is about 20 percent, so that in turn, an almost 80 percent of the queries do. Queries without wildcards supposedly only check for the existence or the commonness of a phrase. More than 70 percent of the query phrases include only one wildcard (see Table 3).

An in-depth analysis indicates interesting patterns of user behavior. Most users interact with NETSPEAK in sessions (i.e., by posing a series of queries within a certain time frame). Only 18 percent of the queries belong to single-query sessions. We have identified two different session types:

1. *Bunch of Queries.* In this case, a session consists of unrelated queries, where none of the queries have words or wildcards in common with previous or successive queries. It appears as if users first write a large chunk of text and then check those phrases about which they are uncertain.
2. *Query Refinement Session.* In this case, a session consists of related queries, where queries following each other are very likely to have words and wildcards in common. It appears as if a user is working on a particular phrase, searching for alternatives. This session type is most common.

TABLE 2
Relative Fractions According to Query Length (from NETSPEAK's Query Log)

Query Length	1	2	3	4	5	≥6
Fraction	6.2 %	13.9 %	53.5 %	15.9 %	8.08 %	1.9%

TABLE 3
Relative Fractions According to the Number of Contained Wildcards (from NETSPEAK's Query Log)

Wildcards	0	1	2	3	≥4
Fraction	20.7 %	71.5 %	7.09 %	0.60 %	0.11 %

The average number of queries per session is 5.6 and the average duration of a session is about 6.5 minutes. A few sessions took very long indeed, lasting more than half an hour. In some of the refinement sessions the users obviously struggled with a certain phrase and continually exchanged words and wildcards over a long period of time. Long refinement sessions are sometimes interrupted by unrelated queries and then continued later on.

NETSPEAK's textual interface does not support the concept of sessions. Thus, query refinement sessions require the user to memorize the results of already performed queries and relate them to each other and to further queries in his mind. This is a challenging cognitive task and was a strong motivation for the development of the WORDGRAPH. The WORDGRAPH interface allows the user to start with a simple query, which can be visually refined and extended while the word graph visualization shows animated transitions between the changing result sets.

4 WORDGRAPH

The WORDGRAPH visualizes the resulting n-grams of a query in a layered graph (Fig. 1, top) and offers interactions with the result set. The nodes of the graph correspond to the words of the n-grams, and an edge represents the connection between two subsequent words of an n-gram. Consequently, each n-gram of a result set is represented as a path through the graph. The layers of the graph are arranged in vertical columns to facilitate reading. Every column corresponds to one element of the query, which can be a literal word or a wildcard character, as defined in Table 1. Multiple occurrences of the same word in a column are merged into a single node.

The graph can be drawn in two ways, a *split path view* and a *condensed path view* (Fig. 2). The *split path view* displays the n-gram paths of a result set independently—similar to the text view—and reveals the overall complexity of the result set. The *condensed path view* merges shared subpaths of different n-grams, which is a compact abstraction of the result set, where individual n-grams are no longer directly visible. While merging the paths significantly enhances overall legibility of large graphs, it also brings about the problem of spurious results since more paths can be created than are actually supported by the result set. However, the condensed path view is the preferred view in practice, which is why we have developed several techniques to

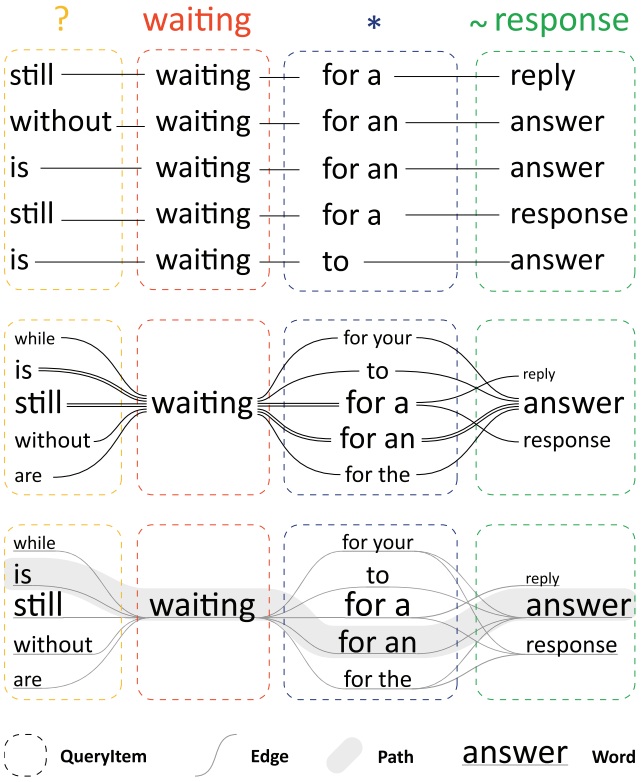

Fig. 2. The query `? waiting * ~response` combines one word and three wildcards; the search results are shown. The WORDGRAPH visualizes n-grams as paths through the graph, merging multiple occurrences of a word within a column. Every path can be drawn individually (split path view, middle) or shared subpaths can be merged (condensed path view, bottom). The latter view increases the probability for spurious results. That is, a path for `is waiting for a reply` is shown although this 5-gram is not part of the result set.

counter this problem and to allow users to interactively explore the result set.

While the text interface of NETSPEAK offers no interaction beyond the retrieval of example sentences for a particular n-gram, the WORDGRAPH provides various means for exploring the search results, including filter techniques and support for navigation. Even more importantly, visual query expansion and query modification are supported along with animated transitions among subsequent result sets.

4.1 Graph Filter

The filter operations allow users to reveal the paths passing through a certain node, to emphasize certain paths, and to select a subgraph by specifying a set of nodes (Fig. 3). The filter operations are orthogonal, as in they can be applied repeatedly in an arbitrary order. Users may also switch between the condensed path view and the split path view at any time. Transitions between different views are animated to facilitate the understanding of the relationships between the different layouts.

4.2 Horizontal Query Expansion

Since the basis of our retrieval engine is the Google n-gram corpus, only n-grams up to a length of $n = 5$ words can be retrieved. By means of our query expansion technique we can address this limitation and allow the retrieval of longer phrases based on those already displayed in WORDGRAPH

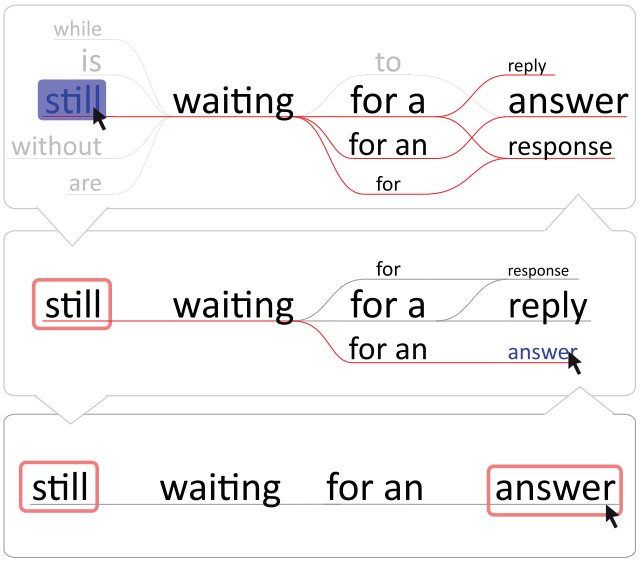

Fig. 3. The WORDGRAPH offers several filter operations: (1) Hovering the mouse above a node highlights all n-gram paths passing through the node. (2) Selecting a node deemphasizes all paths of n-grams that do not contain the selected word. Multiselection is supported. (3) The subgraph filter hides elements that do not belong to selected paths.

(Fig. 4). By clicking on the expansion icons shown next to a word while hovering over it, new queries are constructed for all n-grams whose paths go through the word's node, using up to four preceding words and appending the respective wildcard ? or *. The union of the result sets of all these queries is then integrated into the existing graph structure, and new columns are added as needed. Every expansion entails $O(k^3)$ new queries, where k denotes an upper bound on the number of incoming edges of a word in the WORDGRAPH with k typically being between 4 and 10. The n-grams formed in this way may be longer than the n-grams contained in the underlying corpus and thus may be incorrect or meaningless. Nevertheless, sensible results have been observed in many cases.

4.3 Vertical Query Expansion

In addition to the horizontal query expansion, we provide a means to vertically expand the graph (i.e., within a column)

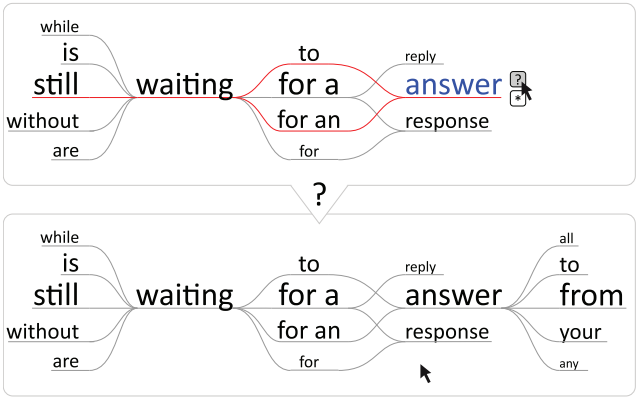

Fig. 4. Query expansion: By clicking on the wildcard icon next to the word `answer`, a set of queries is generated for all n-grams whose paths pass through this word's node, complemented by the respective wildcard. The n-grams retrieved with these queries are integrated into WORDGRAPH which results in a new column.

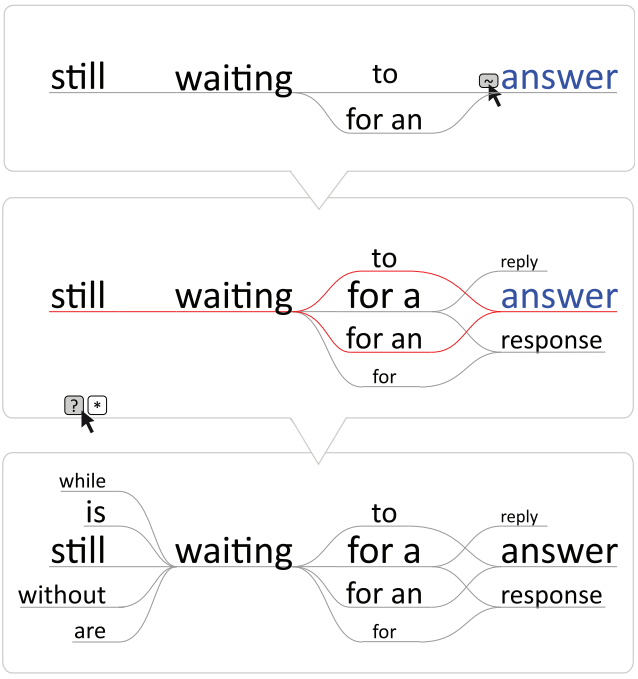

Fig. 5. Vertical query expansion. The user starts with a simple query containing one wildcard (top). In the next step, the query is expanded with synonyms of the word `answer` and the query results are being integrated into WORDGRAPH (middle). The third step changes a column representing a query word into a wildcard column (bottom).

by replacing words by wildcards or wildcards by other wildcards. We distinguish between operations acting on a single word and operations that are related to an entire column (Fig. 5). Currently, we offer only the synonym-operator for word-related query expansion and the wildcards `?` and `*` for column-related query expansion, which replace the word or operator in the corresponding column of the original query. In principle, for both cases all three operations could be made available. For word-related expansions the word is replaced by the corresponding wildcard in all paths through this word. Column-related query expansion could be transformed simply into word-related query expansions by applying the chosen wildcard to each word in the column.

For each expansion the corresponding queries are generated and the results are integrated into the graph structure. The word-related operations just add the new n-grams to the existing structure. Column-related expansions transform the graph into a new one by removing paths only contained in the old result set, preserving paths that exist in both sets, and adding new ones.

Horizontal and vertical query expansion lead toward visual query specification and modification, which replaces the common sequence of manually typed-in queries. The animated transition between the query results visually relates the results of the subsequent queries to each other instead of simply replacing the previous result set by a new one.

4.4 Navigation

The query expansion technique produces word graphs that have too many columns to fit on the screen. To allow for navigation, we implemented horizontal panning and scrolling support by directly dragging the entire graph. An overview bar at the bottom of the screen (Fig. 6) helps the

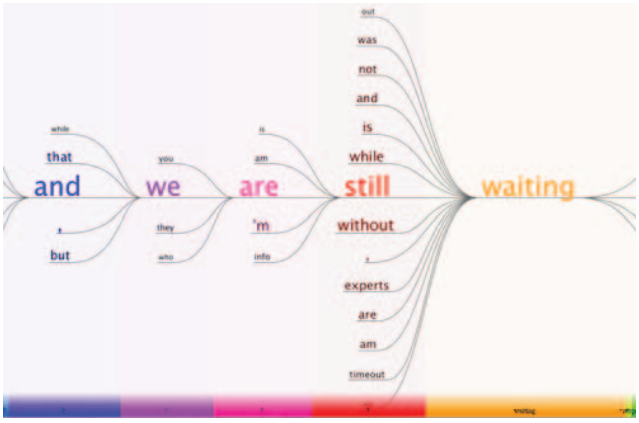

Fig. 6. Horizontal navigation: The overview bar at the bottom of the screen shows the columns of WORDGRAPH. Selecting a column moves that column into the center using an animated transition. Also, the whole graph can be moved horizontally while columns are collapsed and expanded as necessary.

user to control the horizontal panning and make it possible to jump immediately to a specific column, which automatically scrolls into the center of the screen. Columns which do not fit on the screen appear collapsed on the overview bar.

Vertical navigation becomes necessary if a column does not provide sufficient space for the set of retrieved words. We experimented with two different strategies for dealing with this case: an explicit focus-and-context approach and a clipping technique (Fig. 7).

Our focus-and-context technique distorts the font size in the distal areas of the column, but leaves it unchanged in the central 80 percent of the column. This is an important design decision, since we map the relative frequency of a word to the font size. With the simple clipping technique the clipped edges hint at further words located outside of the visible area just as the focus-and-context technique does. However, it completely clips edges that connect clipped

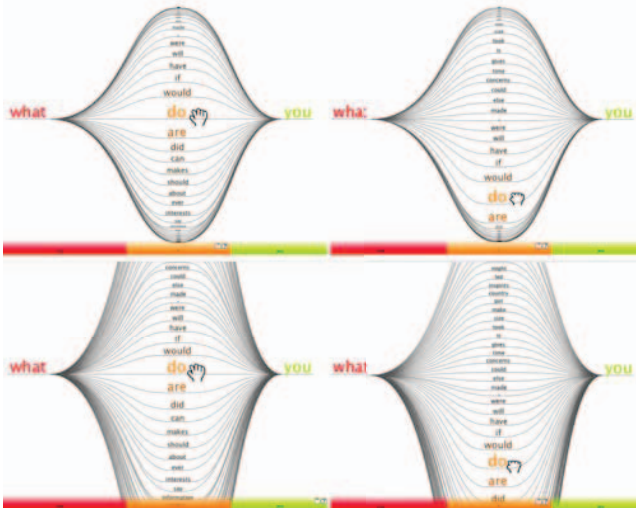

Fig. 7. Vertical navigation: (top) The focus-and-context approach has an aesthetic appearance. (bottom) The simple clipping solution results in a clean separation of the edges connecting to the distal words. The left images show the initial view, while the right images show the resulting view after dragging the word **do** downward for shifting the focus on the words in the upper part of the column.

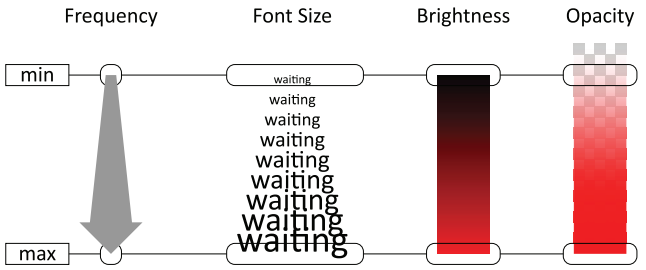

Fig. 8. The accumulated occurrence frequency of the individual words is mapped by utilizing several possibilities to visualize the importance of a single word among all words: (1) the font size, (2) the brightness of the font color, and (3) the opacity of the font color (not default, because it seems to depend on individual preference).

words in two subsequent columns. Both techniques generate overplotting of the edges connecting to the distal words. The overplotting is more pronounced with the focus-and-context technique, which makes it more difficult to estimate the number of words in the context area (Fig. 7). As a result, we generally prefer the clipping technique over the focus-and-context technique.

5 WordGraph Layout Details

This section explains the important design decisions for the layout and rendering in WordGraph. The central concern is the legibility of phrases, and hence the placement of words in subsequent columns is essential. The layout also needs to reflect properties of individual words (e.g., font, size, color, and opacity, see Fig. 8), properties of edges (e.g., path, color, and width), and attributes of n-grams (e.g., absolute and relative occurrence frequency).

The layout process consists of five steps:

1. Horizontal partitioning of available screen space into columns.
2. Vertical ordering within these columns.
3. Exact placement of words.
4. Drawing of edges between (underscored) words.
5. Performing crossing reduction, if possible.

5.1 Screen Partitioning and Word Placement

The initial layout of WordGraph evolves from the submitted query. The longest n-gram returned determines the number of necessary columns. The width of each column takes into account font sizes, word lengths, and additional padding, as shown in Fig. 10. Within a column, each word is horizontally centered, except for the first and last column, respectively.

The vertical arrangement can be done in two ways: one strategy is the *top spread ordering* (Fig. 9, top), which is similar to the text view. The second strategy is the *center spread ordering*, which places words in a column with decreasing font size, starting from the center and alternating the placement above and below (Fig. 9, bottom). The latter strategy is preferred since it places the most important query result in the middle of the screen and facilitates the tracing of alternative phrases without introducing large intercolumn skips.

For the vertical word placement within a column we experimented with two possible layouts, shown in Fig. 10:

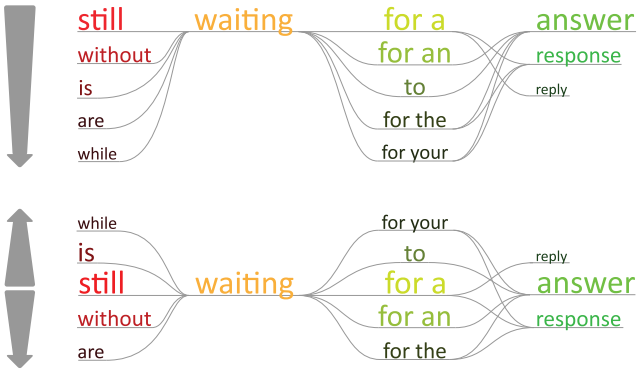

Fig. 9. Arranging words in a column according to their frequency: The *top spread ordering* (top) and the *center spread ordering* (bottom). The relative occurrence frequency of a word in a column is mapped to its font size, color, and brightness.

the *maximal word spreading* (top) uses the entire vertical and horizontal space of a column for equally distributing the words; it is independently applied for each column. The alternative *grid-based word placement* (bottom) is more compact and uses a grid to place the words. The defined cell height for all columns depends on the font size of the most frequent word of all columns. In every column the algorithm starts from the center and places the words above and below, aligned to the defined cell height (Fig. 10, bottom), which minimizes the vertical spread from the center. Horizontally the algorithm is more flexible: in the first and the last column the words are aligned to the inner padding, while in the other columns they are centered. We found that the grid-based vertical partitioning of all columns along with a minimal spread from the center (Fig. 10, bottom) facilitates the readability of the phrase fragments since it resembles a printed page.

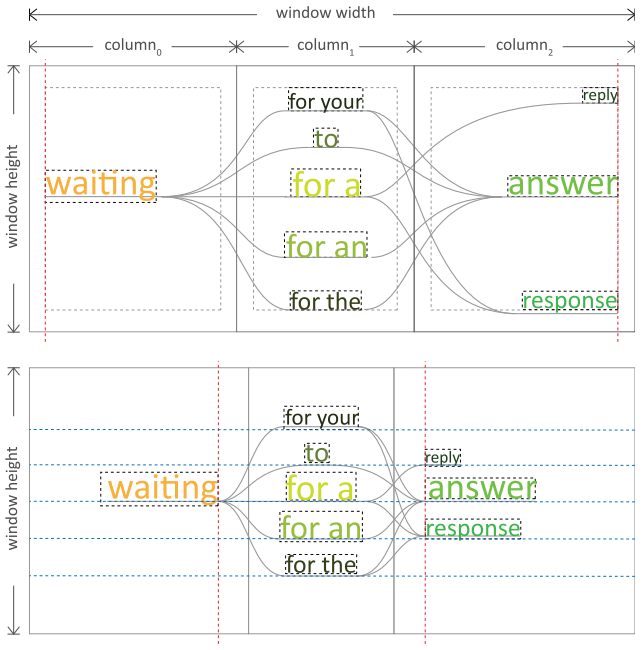

Fig. 10. Column layout and word placement in a column. Maximal word spreading (top). Grid-based word placement of all columns (bottom).

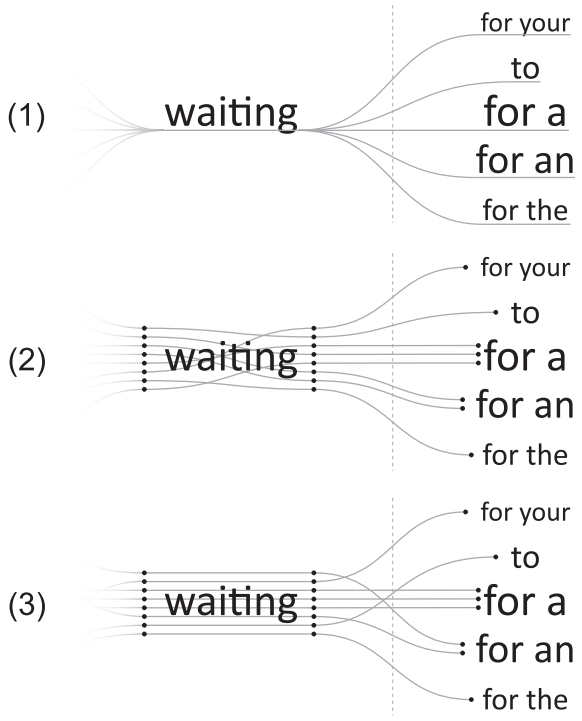

Fig. 11. Possibilities for edge drawing. The edge ports are marked as small dots. (1) *Condensed path view:* all paths between two words are drawn as a single edge and the incoming and outgoing edges are connected by a line passing below the word to improve readability. (2) *Split path view:* each path is drawn independently. Crossings occur in the background of the words. (3) *Split path view:* each path is drawn independently. Crossings occur after the words.

5.2 Edge Drawing

A path represents an n-gram within the graph structure. Therefore, it is also a sequence of words connected by edges. A word only occurs once per column, so many paths could be incident to a node.

As previously mentioned in Section 4, the edges of WORDGRAPH can be rendered in two different ways (Fig. 2). The condensed path view draws a direct representation of the graph with at most only a single edge between words. The split path view shows all n-grams contained in WORDGRAPH by drawing all the edges of the n-grams into the graph. Each edge is defined by a cubic Bézier curve. The start point and end point are located at defined locations (ports) on the source word and the target word. The tangents at these points are always horizontal to allow for a smooth transition from a straight line through the word into the edge.

The *condensed path view* places the port for connecting edges at either end of the baseline of the word. The baseline of the word itself is drawn such that the line passes below the word to the other port and continues to an outgoing edge (Fig. 11(1)). We found that drawing a continuous line below the words, which connects incoming and outgoing edges, significantly contributes to the readability of phrase fragments. Moreover, interrupting the curves by words is recognized as a set of single words without meaning rather than a coherent phrase.

The *split path view* shows all paths defined by the n-grams from the search result set at once. The paths are also drawn in the background of the words such that tracing of an individual path across multiple columns is fully supported. Figs. 11(2) and 11(3) show two different ways of vertically arranging the incoming and outgoing edges of a word. Fig. 11(2) attaches the incoming and outgoing edges of a path to ports at the same vertical position and avoids crossings behind the word, but introduces additional crossing outside the word. Alternatively, incoming and outgoing edges on both sides are attached to appropriate ports depending on their starting position (Fig. 11(3)). In this case, edge crossings occur behind the word, which was generally preferred particularly in combination with the available interaction techniques.

5.3 Edge Crossing Reduction

Edge crossings between columns are introduced when merging multiple occurrences of a word in a column. This is particularly annoying if a node in the upper half of a column is connected to a node in the lower half of the subsequent column. For the *center spread ordering* there is some potential to minimize the number of edge crossings. Our approach is inspired by the classical algorithm for drawing layered graphs which was suggested by Sugiyama [23] for his barycenter-based layer-by-layer sweep.

The WORDGRAPH itself consists of columns which form a horizontally oriented layered graph. Thus, to reduce the number of crossings, each possible pair of layers can be treated by fixing one layer and permuting the other employing a heuristic (often barycentric or median) which reorders the nodes according to the positions of their counterparts in the fixed layer. However, in our case, the order of descending node size away from the center should not be lost, and therefore the nodes cannot be reordered arbitrarily.

Edge crossing reduction algorithm. Let $G = \langle V, E \rangle$ denote a WORDGRAPH. We use a layer-by-layer sweep approach (from left to right) to process the columns of G with respect to their predecessor. Given the ith column $V_i \subset V$, with $V_i = (v_1, \ldots, v_l)$, its preceding or succeeding column V_j, respectively, as well as the edges $E_{ij} \subset E$ between them. Presuming the nodes in V_i have already been ordered according to the center spread layout, say, $v_c \in V_i$ is center node, our crossing reduction algorithm assigns ranks to the nodes in V_i. A mapping $rank_i : V_i \to \{-\lfloor l/2 \rfloor, \ldots, \lfloor l/2 \rfloor\}$ is set up to map the nodes in V_i onto ranks, where a node's rank denotes its distance to the center node v_c and the sign of a node's rank denotes whether it is above or below v_c. Likewise, $rank_j$ assigns ranks to the nodes in V_j (Fig. 12). Then, for each pair of equidistant nodes $(v, v') \in V_i \times V_i$, where $rank_i(v) = -rank_i(v')$, it is determined whether they should be swapped within V_i, which is the case if the barycenter of v' lies above that of v

$$\sum_{\{v',u\} \in E_{ij}} rank_j(u) \quad \geq \quad \sum_{\{v,u\} \in E_{ij}} rank_j(u).$$

We tested different orders to process the WORDGRAPH layers using several graphs from queries containing different numbers, kinds, and positions of wildcards. Altogether, we found that the simplest approach to process the layers from left to right yields the best results on average (mean of 26 percent crossing reduction). For simple graphs our algorithm performs only a few swaps. However,

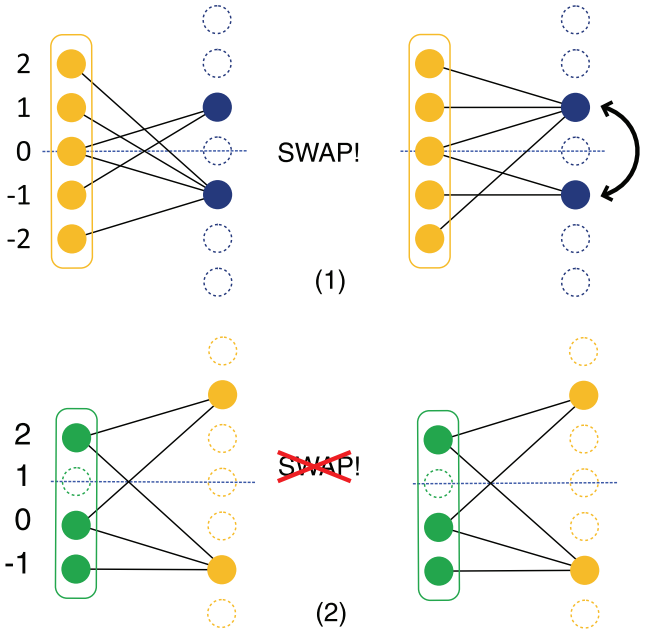

Fig. 12. Edge crossing reduction between two columns. Swapping two equidistant nodes might reduce the number of crossings (1) or it does not (2).

for complex graphs a reduction of crossings of up to 52 percent was observed.

5.4 Layout Guidelines

Based on our experience with alternative implementations of the WORDGRAPH interface we derived the following list of layout guidelines. These guidelines might also be useful for other word-based visualization approaches.

- *Center spread ordering* works better than *top spread ordering*.
- Implement a vertical grid to align words across different columns.
- A minimal vertical word placement starting from the center is preferable.
- Underlining emphasizes the connectivity of a collocation and and significantly improves legibility.
- In the split path view, drawing edge crossings after (instead of behind) words gives a less tangled appearance.
- Crossing reduction between subsequent columns improves legibility.
- Animated transitions are essential for filtering operations, query exploration, and navigation.

6 NETSPEAK'S RETRIEVAL ENGINE

A salient feature of the NETSPEAK phrase search is its efficiency at web-scale. This section introduces the underlying technology to deal with this vast—and still growing—amount of data.

NETSPEAK combines state-of-the-art data structures with original retrieval research in order to answer wildcard queries at the highest possible speed. At its core is a query processor that is tailored to the following task: given a wildcard query q and a set of n-grams D, retrieve those n-grams $D_q \subseteq D$ that

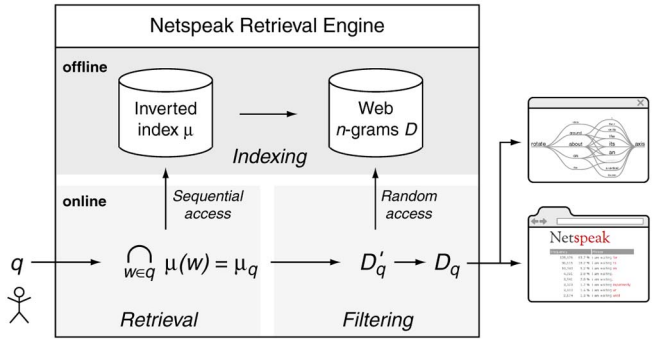

Fig. 13. The NETSPEAK retrieval engine at a glance: Given a query q the intersection of relevant postlists yields a tentative postlist μ_q, which then is filtered and presented as a ranked list or in graph form. The index μ exploits essential characteristics that are known a priori about possible queries and the n-gram set D.

match the pattern defined by q. The query processor addresses the three steps indexing, retrieval, and filtering, as illustrated in Fig. 13.

The indexing step is done offline, once before answering the first query. Let V denote the set of all words found in the n-grams D, and let D' denote the set of integer references to the storage positions of the n-grams in D on hard disk. During indexing, an inverted index $\mu : V \rightarrow \mathcal{P}(D')$ is built that maps each word $w \in V$ to a skiplist $\mu(w) \subseteq D'$ comprised of exactly all references to the n-grams in D that contain w. $\mu(w)$ is referred to as posting list or postlist. Since D is invariant, μ can be implemented as an external hash table with $O(1)$ access to $\mu(w)$. For μ being space-optimal, a minimal perfect hash function based on the CHD algorithm is employed [24].

The two online steps, retrieval and filtering, are taken successively when answering a query q. Within the retrieval step a tentative postlist $\mu_q = \bigcap_{w \in q} \mu(w)$ is constructed; μ_q is the complete set of references to n-grams in D that contain all words in q. The computation of μ_q is done in order of increasing postlist lengths. Within the filtering step, a pattern matcher is compiled on-the-fly from q, and D_q is formed as a set of references pointing to the matching n-grams in μ_q. NETSPEAK exploits the fact that the search in D described above can be significantly narrowed down in our application; the following sections provide an overview of the developed strategies.

6.1 Tailored Indexing

Tailored indexing aims to reduce the filtering effort. The starting point is the distinction of the NETSPEAK queries into fixed-length queries and variable-length queries. The former contain only wildcard operators that represent an a priori known number of words, while the latter contain at least one wildcard operator that expands to a variable number of words. For example, the query `fine ? me` is a fixed-length query since only 3-grams in D match this pattern, while the query `fine * me` is a variable-length query since n-grams of length $2, \ldots, n$ match. Obviously, fixed-length queries can be answered with less filtering effort than variable-length queries: simply checking an n-gram's length suffices to discard many nonmatching queries. The NETSPEAK query processor first reformulates

a variable-length query into a set of fixed-length queries, which then are processed in parallel, merging the results. For example, the aforementioned query `fine * me` is reformulated as follows:

```
fine me
fine ? me
fine ? ? me
  ⋮
```

Since the maximum length of an n-gram in D is small ($n < 8$ for many relevant computer-linguistic applications), the number of fixed-length queries obtained from reformulating a variable-length query is tractable. With k as the number of variable-length wildcard operators in a query q, the size of the respective fixed-length query set is in $O(n^k)$. In the following we assume all queries to be fixed-length queries.

A proper inverted index μ for D enables $O(1)$ access to n-gram sets that fulfill a certain constraint—usually a word w that must occur in all n-grams referred by $\mu(w)$. However, by considering also the position of w an even tighter constraint is imposed. This idea is exploited by the following index μ:

$$\mu : V \times \underbrace{\{1, \ldots, n\}}_{n\text{-gram length}} \times \underbrace{\{1, \ldots, n\}}_{\text{word position}} \rightarrow \mathcal{P}(D'),$$

where the preimage is the Cartesian product of D's vocabulary V, the possible n-gram lengths, and the possible positions of words within n-grams. Given the query $q =$ `fine ? me`, the postlist μ_q is defined as follows:

$$\mu_q := \mu(\text{“}\texttt{fine}\text{”}, 3, 0) \cap \mu(\text{“}\texttt{me}\text{”}, 3, 2).$$

μ_q consists of references to all 3-grams in D that have "fine" as their first word and "me" as their third word. Since this is exactly what the query is asking for, the subsequent step of filtering μ_q can be omitted.

6.2 Postlist Pruning

Postlist pruning aims to reduce set operations. Let $f : D \rightarrow \mathbb{N}$ be a function that indicates the occurrence frequencies of the n-grams in D. Similar to μ, f is implemented as an external hash table. During the indexing step, each postlist $\mu(w, \cdot, \cdot)$ for some $w \in V$ is sorted in decreasing order of the occurrence frequencies of the referenced n-grams, which allows for head pruning and tail pruning.

Head pruning means to start reading a postlist at some entry within, without compromising the recall. Given a query q let τ denote an upper bound for the frequencies of the n-grams in q's result set D_q, i.e., $d \in D_q$ implies $f(d) \leq \tau$. Obviously, in all postlists that are involved within the construction of D_q, all entries whose n-gram frequencies are above τ can safely be skipped. We assess τ as follows:

$$\tau = \min_{d \subseteq q}(f(d)),$$

where d is a maximum, nonterminal n-gram in q. For example, the query $q =$ `sounds fine ? me` contains the two maximum, nonterminal n-grams "sounds fine" and "me" with the frequencies $f(\text{“}\texttt{sounds fine}\text{”}) = 45{,}817$ and $f(\text{“}\texttt{me}\text{”}) = 566{,}617{,}666$. Since no n-gram matching q can have a frequency larger than $\tau = 45{,}817$, all entries of $\mu(\text{“}\texttt{sounds}\text{”})$, $\mu(\text{“}\texttt{fine}\text{”})$, and $\mu(\text{“}\texttt{me}\text{”})$ whose n-grams have a higher frequency than τ can be skipped.

To efficiently determine the first entry of a postlist $\mu(w, \cdot, \cdot)$, $w \in q$, whose frequency drops below τ, an additional meta index μ_f is built during the indexing step, which indexes the postlists of μ. The postlist $\mu_f(w, \cdot, \cdot)$ comprises entries of the form $(f(d), i)$, indicating that the n-gram d referred to at the ith entry of $\mu(w, \cdot, \cdot)$ has frequency $f(d)$. To keep μ_f's memory footprint small, only those postlists from μ are indexed that cannot be read at once into main memory. In addition, only every ith entry of a postlist $\mu(w, \cdot, \cdot)$ is indexed in its corresponding postlist $\mu_f(w, \cdot, \cdot)$, so that $|\mu_f(w, \cdot, \cdot)| = |\mu(w, \cdot, \cdot)|/i$, where in our case $i = 1{,}000$.

Up to this point, the retrieval of n-grams matching a query q is exact—but, not all n-grams that match a query are of equal importance: NETSPEAK users look for n-grams that occur frequently on the web. Taking this fact into consideration, we apply tail pruning on postlists that are too long to be read at once into main memory. As a result, less frequent n-grams that might match a given query may be missed. NETSPEAK employs three tail pruning strategies: 1) stop after a specified number of matching n-grams has been found, 2) stop after a specified number of entries from a postlist has been read, 3) stop after a specified quantile of a postlist has been read. The last strategy is used to define word-class-specific pruning heuristics, since different word classes (stop words, nouns, adverbs, etc.) have a different impact on the construction of D_q. Section 7 reports on effects of these strategies. On demand, a pruned search can be resumed in order to retrieve the complete result set.

7 Evaluation Results and Discussion

In this section, we provide implementation details and evaluate NETSPEAK's components: we report on experiments to assess the retrieval performance of our query processor, conduct a user study, and conclude with a discussion of use cases for the WORDGRAPH interface.

7.1 Implementations Details

The communication between NETSPEAK's interfaces and its retrieval engine is implemented with the Ajax paradigm, using the lightweight JavaScript Object Notation interchange format JSON. The retrieval engine is written in C/C++ and is deployed at our site, accessible through a servlet container. The textual Web interface is implemented using the Google Web Toolkit and it is deployed on the Google App Engine. The visualization client is a standalone application written in Java, deployed at our site. The Java scene graph project Scenario is used to manage and display graphical 2D-elements. Scenario provides convenient methods to handle different kinds of animations [25].

7.2 The Web n-Gram Collection

To provide relevant suggestions, a wide cross section of written text on the Web is required. Currently, we use the Google n-gram corpus "Web 1T 5-gram Version 1" [26], which consists of 42 GB of phrases up to a length of $n = 5$ words along with their occurrence frequency on the web in 2006. This corpus has been compiled from approximately 1 trillion words extracted from the English portion

TABLE 4
Google n-Grams before and after Postprocessing

Corpus subset	Corpus size # n-grams	Space	After post-processing
1-gram	13 588 391	177.0 MB	3.75 %
2-gram	314 843 401	5.0 GB	43.26 %
3-gram	977 069 902	19.0 GB	48.65 %
4-gram	1 313 818 354	30.5 GB	49.54 %
5-gram	1 176 470 663	32.1 GB	47.16 %
Σ	3 354 253 200	77.9 GB	54.20 %

of the Web, totaling in more than 3 billion n-grams. Two postprocessing steps were applied: case reduction and vocabulary filtering. For the latter, a white list vocabulary V was compiled and only n-grams whose words appear in V were retained. V consists of the words found in the Wiktionary and various other dictionaries, complemented by words from the 1-gram portion of the Google corpus whose occurrence frequency exceeds 11,000. After post-processing, the size of the corpus has been reduced by about 46 percent. Table 4 gives an overview.

7.3 Retrieval Performance Evaluation

To evaluate the retrieval performance of the query processor we report on experiments where the retrieval time for a certain recall is measured, dependent on the query processor variant. Here, the retrieval time is quantified as the number of read postlist entries, which is independent from the implementation or underlying hardware. Recall is a standard performance measure in information retrieval. Given an inverted index μ and a query q it quantifies whether a document that belongs to the true result set in fact gets retrieved

$$rec(\mu, q) = \frac{\sum_{d' \in (\mu_q \cap \mu_q^*)} f(d)}{\sum_{d' \in \mu_q^*} f(d)},$$

where μ_q contains the retrieval results obtained using one of the query processing strategies, while μ_q^* contains references to all n-grams that actually match the pattern of q. The measure considers the occurrence frequency $f(d)$ of an n-gram d in order to give n-grams that are more important to the user more weight. We compute recall values at different points during retrieval for a set of 55,702 queries, averaging the results; the queries originate from the query logs of NETSPEAK. Fig. 14 shows the obtained results. As can be seen, tailored indexing has a significant impact on the number of elements to be read in order to achieve a certain recall. Together with postlist pruning, the amount of elements to be read is at least one order of magnitude smaller than with index-based retrieval alone.

7.4 User Study

We performed a user study to assess the usability of our system and to learn about the user acceptance and potential improvements. In particular, we were interested in a comparison of the WORDGRAPH interface and the basic textual interface. Based on feedback from public demonstrations and a pilot study (described in [5]) we derived our main hypothesis: both interfaces perform equally well for

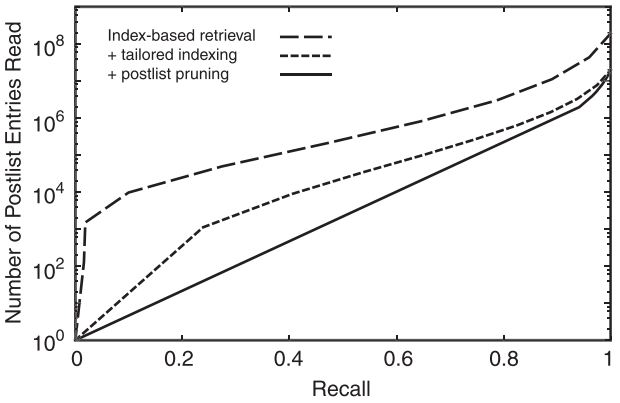

Fig. 14. Average number of postlist entries read during query processing in order to achieve a certain recall.

basic keyword-in-context queries using only a single wildcard. With more than one wildcard, users generally prefer the visual WORDGRAPH interface.

Ten nonnative English speakers with higher English education participated in our study. All of them were volunteers from an English writing course offered by the language center at the university. None of the participants were aware of the NETSPEAK web service or the WORDGRAPH visualization. During a brief introduction of the textual NETSPEAK interface and the WORDGRAPH interface, the participants could enter queries and examine the result sets with the two different interfaces.

The actual study comprised of six pairs of queries. Each pair consisted of two different queries with similar structure and the same number of wildcards. There were two pairs using one, two, and three wildcards, respectively. The level of complexity of the queries increased incrementally. The participants were asked by an instructor to enter one query of every pair in the textual interface and the other one in the WORDGRAPH interface, select the most suitable results, and to assess on a Likert scale how helpful each interfaces was for the task, from 1 (not helpful at all) to 6 (very helpful).

The order of interfaces was counterbalanced into two subgroups. One group requested the first query of a pair with the NETSPEAK web interface and the second one with the WORDGRAPH, and vice versa. Each participant repeated the procedure for each of the six pairs, which resulted overall in 120 assessed requests, 60 per interface.

During the study the instructor observed and ranked for each participant how well the WORDGRAPH interaction patterns were understood. The instructor also noted his impressions about the usage of the different WORDGRAPH interaction patterns. Afterward, the participants were asked in a questionnaire about the general usage and their comments on limitations and desired improvements with respect to interaction and layout.

For each level of difficulty a t-test was conducted to compare the WORDGRAPH interface and the text-interface (Fig. 15). We used an alpha level of 0.05 for all statistical tests. For one-wildcard queries both interfaces achieved similar ratings, the WORDGRAPH interface ($M = 4.75$, $SE = 0.20$) and for the text interface ($M = 4.8$, $SE = 0.23$), $t(9) = 0.15$, $p = 0.88$. For two wildcards, however, the results indicate a significant preference for the WORDGRAPH interface

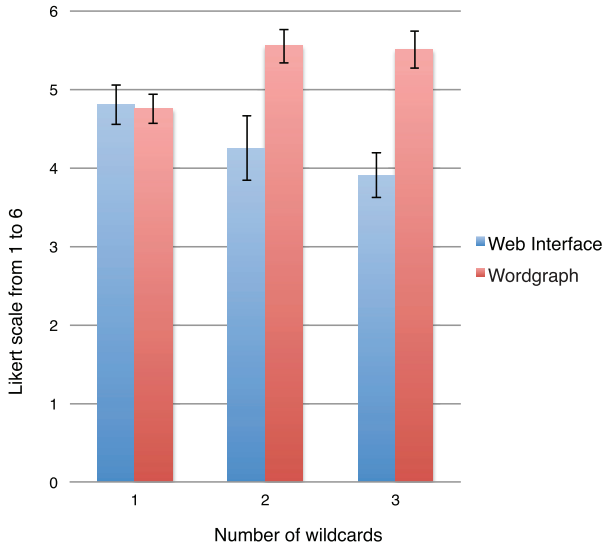

Fig. 15. Mean interface preferences and standard errors according to different numbers of wildcards.

($M = 5.55$, $SE = 0.21$) over the text interface ($M = 4.25$, $SE = 0.41$), $t(9) = 2.94$, $p = 0.016$. An even stronger preference for the WORDGRAPH was revealed for three wildcards: ($M = 5.5$, $SE = 0.23$) versus ($M = 3.9$, $SE = 0.28$), $t(9) = 3.64$, $p = 0.005$. These results support our hypothesis of an increasing preference for the WORDGRAPH interface with an increasing occurrence of wildcards in a query.

The questionnaires reveal that all participants would like to see WORDGRAPH being provided as an additional interface for NETSPEAK. Six of them even considered WORDGRAPH as a substitute for the textual Web interface. They assessed WORDGRAPH as "very intuitive" with an average of 5.1 on a scale from 1 to 6. This positive impression was confirmed by the following observation: the instructors judged the understanding of the interaction patterns by the participants with an average of 5.0. The observation also revealed that seven of the participants applied the subgraph filter predominately to explore the response graph instead of using the mouse-over technique. Only one participant exclusively applied the mouse-over technique during the tests.

Altogether, the answers from the questionnaire and the general feedback we gathered during several public presentations revealed the most appreciated features of the WORDGRAPH:

1. Fluent result filtering.
2. Starting from an overview with the most important information.
3. The possibility of exploring the response set in detail by successive or alternating applications of subgraph filtering.
4. Finally, the improved legibility of the word sequences within the graph by following the edges through the nodes: "It gives the impression of reading from a sheet of lined paper."

Eventually the participants suggested several interesting aspects for improving the WORDGRAPH interface: 1) Most users want to be able to request sentence snippets earlier during the process of exploring the graph and are not willing to wait until only one phrase remains. 2) In cases of two or more words with visually similar frequencies some users would like to see the absolute and relative frequency numbers on demand. 3) The use of thicker edges (in combination with the current coloring) for highlighting a word sequence within the graph was also suggested.

Fig. 16. Word choice with NETSPEAK's Web interface.

7.5 Use Cases and Experiences

Based on our query log analysis, the session types identified and the user study, we identified three practical retrieval tasks related to word choice, which have an increasing level of difficulty:

1. *Phrase Verification.* The most basic retrieval task is to check whether a given phrase is commonly used. As mentioned above, almost 20 percent of all queries come without wildcards. For this task, the textual interface is fully sufficient.
2. *Context-Sensitive Word Choice.* In this retrieval task, a writer is uncertain about what alternative for a word in a given phrase is a good choice, or whether there are in fact any alternatives. This task pertains particularly to second-language speakers who often translate words using a dictionary—the exact translation of many words depends on context. In this respect, NETSPEAK serves as a context-sensitive thesaurus. Choosing the correct adverbs and prepositions is also a common problem. Fig. 16 illustrates how NETSPEAK is used to find the correct collocations between the words `rotate` and `axis`.

   The query language of NETSPEAK is powerful in that it makes it possible to specify rather complex patterns of n-grams to be retrieved. A user who inserts more than one wildcard into a query is less confident about how to write a certain phrase and seeks to generalize the query in order to cover more of the possible alternatives. This, in turn, yields a longer list of results in the textual interface, which may be difficult to overview and which may not always reveal the true picture about which words to choose. Fig. 16 shows an example where `about` appears in three of the n-grams, which indicates that this word should most likely follow `rotate`. The textual Web interface, however, obscures this fact and the user is forced to scan the entire result list several times to grasp the true relationships. By contrast, the WORDGRAPH visualization for the same query as above provides an overview at a glance (see Fig. 17). This is in accordance with our user study, which revealed a preference for the WORDGRAPH interface over the text interface for queries containing more than one wildcard.

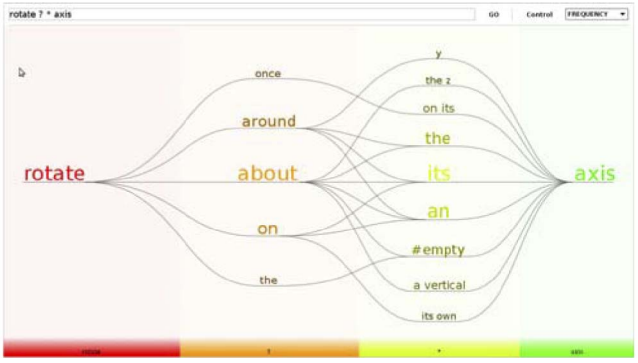

Fig. 17. Word choice with the NETSPEAK WORDGRAPH.

3. *Exploration.* This retrieval task is about writers who want to explore the typical context of a phrase by looking at what comes before, after, or in between the phrase's words. With the NETSPEAK's textual interface, this task is limited to exploring a context of up to four words around a query that comprises, say, only one word surrounded by asterisks. Only by means of additional queries, a user may get a broader view of a phrase's context, having to keep in mind the results of all previous queries. With the WORDGRAPH interface, this task is supported without further ado by means of the query expansion technique (Fig. 4). The results of additional queries, which can be posed interactively, are integrated seamlessly into an existing graph so that users can construct a full picture of a phrase's context. This capability of WORDGRAPH is particularly useful for expert users, including linguists who investigate the characteristics of language use in a given corpus.

Remarks. While writing a text, such as a scientific paper, users often switch back and forth between different retrieval tasks. Phrase verification is the least observed task, which is documented by NETSPEAK's query logs; 80 percent of the queries comprise wildcards. There are two common types of queries: queries asking for the most suitable word in a given context, and queries asking for the typical context of a particular word or, more precisely, which common collocations a particular word has. Thus, it is context sensitivity that is most relevant to the users, which is difficult to express with other commonly available tools. With the textual Web interface, one typically looks at the top results and ignores the rest—similar to the use of a Web search engine. With WORDGRAPH, one explores the results more thoroughly and discovers relationships between words that are not apparent in the textual interface. While the latter often forces a user to formulate a sequence of similar queries, the former provides an effective means for implicit query specification, using filter techniques, query expansion and navigation.

8 CONCLUSIONS AND FUTURE WORK

NETSPEAK answers complex word sequence queries that are formulated in an expressive query language. The system is designed for efficiency and allows for real-time querying of a 42 GB text database. The result set is explored via a textual Web interface or the graphical WORDGRAPH interface. Our analysis shows that the textual interface is sufficient for phrase verification and the comparison of related sentences. The WORDGRAPH interface allows an interactive exploration of the result set and is superior for word choice problems on complex queries. The layout of WORDGRAPH focuses on facilitating legibility, which is achieved by using *center spread ordering*, grid-based word placement, and underscoring edges. Participants of our user study describe WORDGRAPH as very intuitive and appreciate the possibility of graph-based filtering during explorative analyses.

We see NETSPEAK in combination with its visual interface WORDGRAPH as a great educational tool for improving the knowledge of a second language. Additional smart operators for the query language such as antonym wildcards or semantic constraints (e.g., person names, places, dates, and times) and support for further languages besides English would broaden the scope of NETSPEAK. An extension toward domain-specific corpora can help inexperienced authors to become familiar with the appropriate expressions and writing style in a specific field.

The interactive WORDGRAPH interface already allows the user to start with a simple query and visually refine and extend the query. This process generates queries containing various wildcards without the user knowing. We believe that this kind of visual query specification is the right approach since most users (>98 percent) of existing search engines are not aware of the simplest of search operators [27]. Further visual query refinement operations could include constraints to certain word types (e.g., parts-of-speech, location, time, colors) and recently added operators of the NETSPEAK retrieval engine.

Individual documents or even entire corpora can be represented as a wordgraph. An individual node in such a large wordgraph could represent a single word, a common collocation or an n-gram of a certain length. The different levels of granularity allow a tradeoff between the number of nodes and the number of edges in the wordgraph, which a multilayered graph could tie together in a single visualization. These examples illustrate only a fraction of the untapped potential, that is why we believe that the wordgraph is one of the most promising tools for semantic text analytics.

ACKNOWLEDGMENTS

The authors thank the reviewers for their constructive comments and valuable suggestions for improving the paper.

REFERENCES

[1] F. van Ham, M. Wattenberg, and F. Viégas, "Mapping Text with Phrase Nets," *IEEE Trans. Visualization and Computer Graphics*, vol. 15, no. 06, pp. 1169-1176, Nov./Dec. 2009.

[2] M. Wattenberg and F.B. Viégas, "The Word Tree, An Interactive Visual Concordance," *IEEE Trans. Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1221-1228, Nov./Dec. 2008.

[3] GoogleLabs, "Google Scribe," http://scribe.googlelabs.com/, 2012.

[4] T. Park, E. Lank, P. Poupart, and M. Terry, "Is the Sky Pure Today? Awkchecker: An Assistive Tool for Detecting and Correcting Collocation Errors," *Proc. 21st Ann. ACM Symp. User Interface Software and Technology (UIST '08)*, pp. 121-130, 2008.

[5] P. Riehmann, H. Gruendl, B. Fröhlich, M. Potthast, M. Trenkmann, and B. Stein, "The NETSPEAK WORDGRAPH: Visualizing Keywords in Context," *Proc. IEEE Pacific Visualization Symp. (PacificVis)*, pp. 123-130, Mar. 2011.

[6] F. Viegas and M. Wattenberg, "Web Seer," http://hint.fm/projects/seer/, 2012.
[7] W.B. Paley, "Textarc: Showing Word Frequency and Distribution in Text," Poster Infovis, http://www.textarc.org/appearances/InfoVis02/InfoVis02_TextArc.pdf, 2002.
[8] C. Harrsion, "Web Trigrams," http://www.chrisharrison.net/projects/visualization.html, 2012.
[9] C. Collins, M.S.T. Carpendale, and G. Penn, "Visualization of Uncertainty in Lattices to Support Decision-Making," *Proc. EuroVis,* pp. 51-58. 2007,
[10] J. Heer and S.K. Card, "Doitrees Revisited: Scalable, Space-Constrained Visualization of Hierarchical Data," *Proc. Working Conf. Advanced Visual Interfaces (AVI '04),* pp. 421-424, 2004.
[11] C. Plaisant, J. Grosjean, and B.B. Bederson, "Spacetree: Supporting Exploration in Large Node Link Tree, Design Evolution and Empirical Evaluation," *Proc. IEEE Symp. Information Visualization (InfoVis '02),* p. 57, 2002.
[12] C.D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing.* The MIT Press, 1999.
[13] C. Leacock, M. Chodorow, M. Gamon, and J. Tetreault, *Automated Grammatical Error Detection for Language Learners.* Morgan and Claypool Publishers, 2010.
[14] M. Hagen, M. Potthast, B. Stein, and C. Bräutigam, "Query Segmentation Revisited," *Proc. 20th Int'l Conf. World Wide Web (WWW '11),* S. Srinivasan, K. Ramamritham, A. Kumar, M. Ravindra, E. Bertino, and R. Kumar, eds., pp. 97-106, Mar. 2011.
[15] J.-B. Michel, Y. Shen, A. Aiden, A. Veres, M. Gray The Google Books Team, J. Pickett, D. Hoiberg, D. Clancy, P. Norvig, J. Orwant, S. Pinker, M. Nowak, and E. Aiden, "Quantitative Analysis of Culture Using Millions of Digitized Books," *Science,* vol. 331, no. 6014, pp. 176-182, http://www.isrl.uiuc.edu/~amag/langev/paper/michel2011googleBooksSCIENCE.html, Jan. 2011.
[16] W.H. Fletcher, "Web as Corpus," http://www.webascorpus.org/, 2012.
[17] Research and Development Unit for English Studies, "Webcorp Live," http://www.webcorp.org.uk/, 2012.
[18] W.H. Fletcher, "Phrases in English," http://www.phrasesinenglish.org/, 2012.
[19] P. Resnik and A. Elkiss, "The Linguist's Search Engine: an Overview," *Proc. ACL Interactive Poster and Demonstration Sessions (ACL '05),* pp. 33-36, 2005.
[20] M.J. Cafarella and O. Etzioni, "A Search Engine for Natural Language Applications," *Proc. 14th Int'l Conf. World Wide Web (WWW '05),* pp. 442-452, 2005.
[21] P. Chubak and D. Rafiei, "Index Structures for Efficiently Searching Natural Language Text," *Proc. 19th ACM Int'l Conf. Information and Knowledge Management,* pp. 689-698, 2010.
[22] Webis Group at Bauhaus-Universität Weimar, "Netspeak Writing Assistance," http://netspeak.cc, 2012.
[23] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for Visual Understanding of Hierarchical System Structures," *IEEE Trans. Systems, Man, and Cybernetics,* vol. SMC-11, no. 2, pp. 109-125, Feb. 1981.
[24] D. Belazzougui, F. Botelho, and M. Dietzfelbinger, "Hash, Displace, and Compress," *Proc. 17th European Symp. Algorithms (ESA '09),* pp. 682-693, 2009.
[25] Scenario, Project Scene Graph, https://scenegraph.dev.java.net/, 2012.
[26] T. Brants and A. Franz, "Web 1T 5-Gram Version 1," Linguistic Data Consortium LDC2006T13, 2006.
[27] R.W. White and D. Morris, "Investigating the Querying and Browsing Behavior of Advanced Search Engine Users," *Proc. 30th ACM SIGIR Conf.,* pp. 255-262, 2007.

Patrick Riehmann is working toward the PhD degree with the Virtual Reality Systems group at the Bauhaus-Universität Weimar. His research interests include information visualization, graph drawing, and multitouch interfaces.

Henning Gruendl is currently working toward the master's degree in the computer science and media programme at Bauhaus-Universität Weimar. His research interests include information visualization and real-time rendering.

Martin Potthast received the PhD thesis on "Technologies for Reusing Text from the Web" in December 2011. He joined the working group Web Technology and Information Systems at the Bauhaus-Universität Weimar in 2006. His primary research interests include plagiarism detection, writing assistance technologies, misuse detection in social software, and crowdsourcing.

Martin Trenkmann is working toward the master's degree in the computer science and media programme at Bauhaus-Universität Weimar. His research interests include web technology and software engineering.

Benno Stein is the chair of the Web-Technology and Information Systems Group at the Bauhaus-Universität Weimar (www.webis.de). His research interests include intelligent modeling and solving of knowledge-intensive information processing tasks. He has developed algorithms and tools for information retrieval, data mining, knowledge processing, as well as engineering design and simulation.

Bernd Froehlich is the chair of the Virtual Reality Systems group (www.uni-weimar.de/medien/vr) and a full professor with the Media Faculty at Bauhaus-Universität Weimar. His research interests include real-time rendering, visualization, 2D and 3D input devices, 3D interaction techniques, display technology, and support for collaboration in colocated and distributed virtual environments. He is a member of the IEEE Computer Society.