Textual Difference Visualization of Multiple Search Results utilizing Detail in Context

Edward Suvanaphen and Jonathan C.Roberts
Computing Laboratory,
University of Kent,
Canterbury, Kent, UK, CT2 7NF
{es45,J.C.Roberts}@kent.ac.uk

Abstract

While searching documents on the web users often refine their query terms, to generate new result lists and explore different pages, thus they end up with several result sets. Experience suggests that users compare these different sets, either implicitly or explicitly. Comparison between these sets (a) allows users to observe similarities and differences that are not innately viewable by viewing the sets separately, (b) reduces the cognitive effort of switching from one result set to another and (c) enables them to browse more effectively. In this paper we investigate comparison visualization and describe a prototype search-engine similarity tool (SES), which visualizes the textual difference of multiple web searches using a combination of multiple views and visual bracketing.

Keywords: *Search result visualization, comparison visualization, detail in context, multiple linked views.*

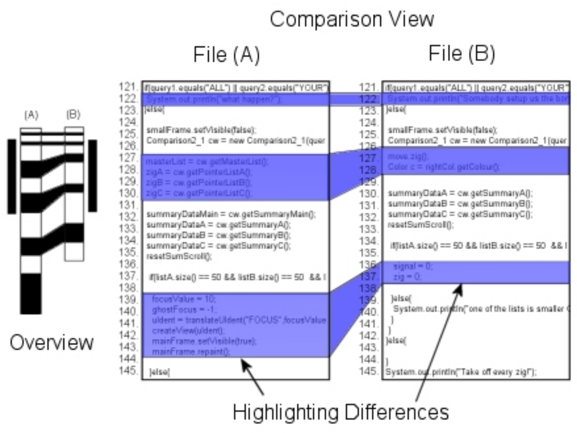
1 Introduction

Web searching is now an everyday activity. Traditional search engines display the results in multiple long lists; the user can scroll the list to view new information or select ‘next’ to display additional items. For the user to progress they must decide whether to open the linked information in the current view (the default operation) or to open the information in a ‘new window’. After viewing several pages they may change the keywords and perform another (semantically related) search. The users subconsciously start to realize similarities and differences between searches.

Thus, for a user to (explicitly) compare multiple search results: they have to use two web-searches displayed side-by-side in two separate windows and visually inspect the two results. Consequently, it is difficult for the user to easily see and understand the similarities and differences between multiple searches.

In this paper we explore the use of a ‘comparison view’. Such a view can pre-attentively visualize the associations between the multiple datasets by explicitly demonstrating and annotating these features. Through such comparisons the user can gain a better understanding of the searched information and they can more effectively browse, because their attention is drawn towards significant features in the results. As these observations are explicitly represented the cognitive overhead of switching from one result-set to another is reduced. The cognitive load can be further exacerbated as the results are traditionally displayed in a long list, which utilize a large amount of space. There are visualizations that address the limited screen space that use abstraction, summarization, multiple views and focus+context methodologies. Our comparison tool uses several of these techniques. Indeed, the tools summary view reduces the consumed screen real-estate by producing a comparison from an intersection of the different search-result lists.

In this paper we focus on the comparison and display of textual data. Although graphical techniques are useful, and by mapping the data to a graphical representation it may allow the user to view the full data set on his screen, but he/she will often lose the subtle details and context information that is provided by a textual representation of the results. Moreover, everyday users are inherently more familiar with text-based browsing: Textual visualizations of the search result data is both important and necessary. Thus it is important to design a solution that utilizes traditional representation methodologies but allows the user to easily compare and contrast different search results. This paper (a) describes principles behind textual difference (section 2) (b) presents a design and implementation for search engine similarity (SES) that uses multiple and detail+context views (sections 3and 4) (c) reviews current difference visualization systems (section 6).



```
if(query1.equals("ALL") || query2.equals("YOUR"
 System.out.println("what happen?");
}else{

 smallFrame.setVisible(false);
 Comparison2_1 cw = new Comparison2_1(quer
 masterList = cw.getMasterList();
 zigA = cw.getPointerListA();
 zigB = cw.getPointerListB();
 zigC = cw.getPointerListC();

 summaryDataMain = cw.getSummaryMain();
 summaryDataA = cw.getSummaryA();
 summaryDataB = cw.getSummaryB();
 summaryDataC = cw.getSummaryC();
 resetSumScroll();

  if(listA.size() == 50 && listB.size() == 50  && l

   focusValue = 10;
   ghostFocus = -1;
   uIdent = translateUIdent("FOCUS",focusValue
   createView(uIdent);
   mainFrame.setVisible(true);
   mainFrame.repaint();

  }else{
```

```
if(query1.equals("ALL") || query2.equals("YOUR"
 System.out.println("Somebody setup us the bon
}else{

 smallFrame.setVisible(false);
 Comparison2_1 cw = new Comparison2_1(quer

 move.zig();
 Color c = rightCol.getColour();

 summaryDataA = cw.getSummaryA();
 summaryDataB = cw.getSummaryB();
 summaryDataC = cw.getSummaryC();
 resetSumScroll();

  if(listA.size() == 50 && listB.size() == 50  && l

   signal = 0;
   zig = 0;

  }else{
   System.out.println("one of the lists is smaller
  }
 }
}else{

}
System.out.println("Take off every zig!");
```

Figure 1. Model of Code Comparison Tool. Two files (A & B) are compared, differences are highlighted in color and an overview of the two documents is shown on the right. Colour highlighting is used to link similar results (Similarity principle) and Synchronized scrolling is used to align the two files while they are being viewed (Symmetry principle).

2 Background & Analysis

The data to form the comparison can come from various sources. Obviously, the user may wish to compare the results from a range of search engines, this may be achieved to observe popularity of different sites or to find particular results that one search engine may not display. Alternatively, the user may wish to compare the results from multiple searches using different keywords, to find links between different subjects or to help the user consolidate towards some ideal search terms. This may involve the use of intersection ($A \cap B$), union ($A \cup B$) and difference ($A - B$) operations. Additionally, meta or statistical information may be compared. For example, appearance, link structure or file size information can be compared. These abstract forms would need to be visualized in graphical techniques, such as bar charts or scatter plots.

One of the earliest text-based comparison tools could be the English Hexapla New Testament [9]. This was printed in 1841 and displays the original Greek at the top of a page with six different English translations underneath in parallel columns (three on the left page and three on the right). This demonstrates the simplest form of comparison, which is side-by-side. We describe four further categories:

The **'side-by-side with cues'** category describes the visualizations that utilize annotations (or various cues) as well as the parallel views. For example, similar information can be concurrently highlighted in the same color, or circles and arrows connecting the objects together could highlight similar objects.

Overlaying multiple visualizations will generate one layered view; much the same way as overlaying a series of overhead transparency foils would generate a single view. This enables a visual comparison of the multiple views and does not alter the format of any visualization. Additionally, various parts of the information could be 'shaded out' to bring to the foreground (say) similarities between datasets; see Figure 1.

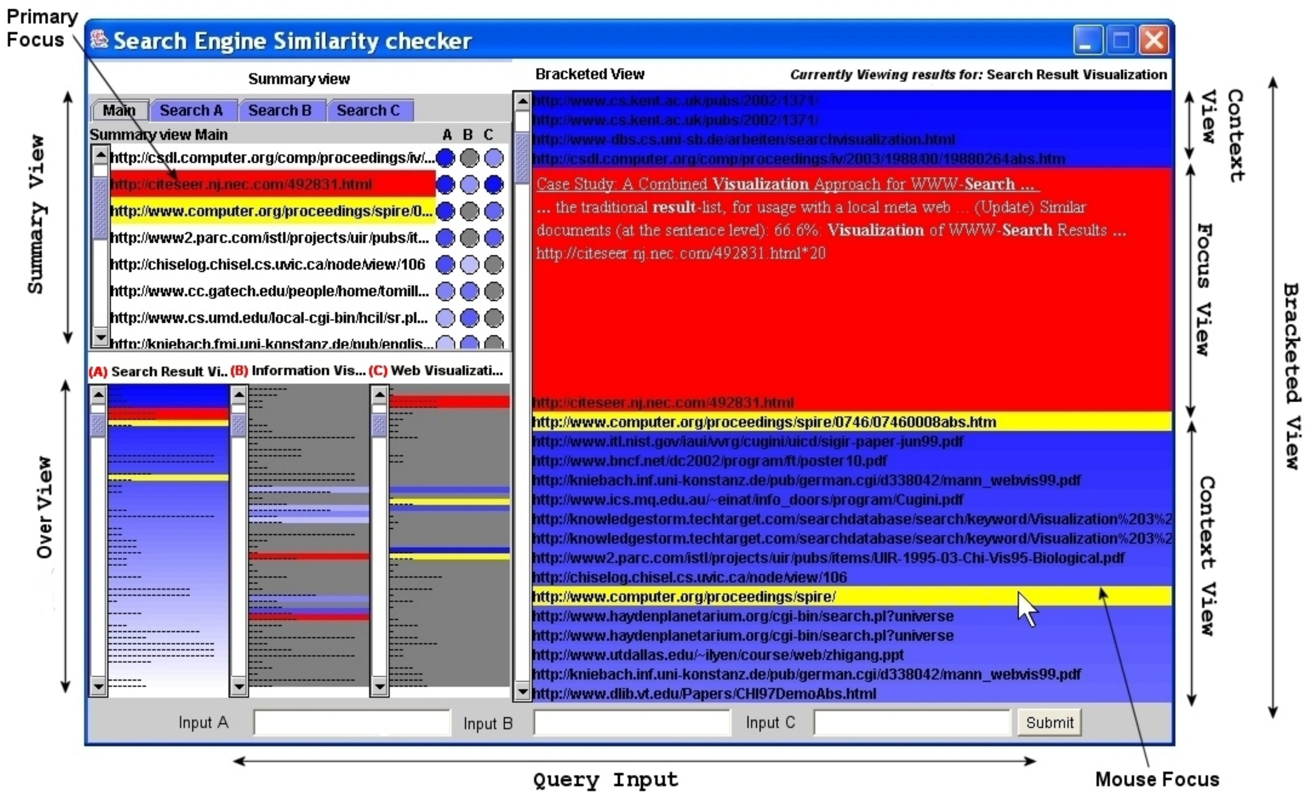
A **merged view** shows multiple datasets in one view. Each of the differences are shown in context with surrounding information. For example, Microsoft Word^{TM} provides a 'merge document' facility that aggregates different versions of the same document together. Sentence or word differences are highlighted in color and by striking through words. The user is given the responsibility to accept or reject individual changes.

Abstract comparison is dissimilar to the aforementioned categories. Traditional methods display the text information as text, albeit various characters may be inserted or highlighted in some form: but it is still a one-to-one mapping. However, it is possible to compose and abstract statistical information from this original data. For example, the structure of a sentence can be represented by a parse tree (noun, verb phrases etc), these parse trees represent an abstract version of the original sentence; hence it is possible to compare and visually depict the difference between several parse trees. Although the comparison can be achieved at the abstract level the visualization can be displayed at either the abstract (such as the parse trees) or translated back to the non-abstract (the original text view). Havre et al [18] utilize abstract views in the SPARKLER visualization; in their tool a dot represents a search result, a group of dots represent a list of search results and multiple groups of dots represent several searches.

3 Design & the Gestalt Principles

In this paper we wish to visualize textual elements that are similar or dissimilar between various lists, thus elements in one list need to be associated with elements in another. This can be achieved through the use of the Gestalt principles (from the German word for 'pattern'). These are based on work by a group of German psychologists in 1912 who detail theories of perceptual organization [22]. Although, many details vary between sources, there is a set of core principles that remain throughout. From experience and evaluating other visualizations we can ideate different designs that utilize the Gestalt principles.

The principle of *proximity* states that objects closer together will be considered as belonging together. An exam-



Summary view Main A B C

http://csdl.computer.org/comp/proceedings/iv/...
http://citeseer.nj.nec.com/492831.html
http://www.computer.org/proceedings/spire/0...
http://www2.parc.com/istl/projects/uir/pubs/it...
http://chiselog.chisel.cs.uvic.ca/node/view/106
http://www.cc.gatech.edu/people/home/tomill...
http://www.cs.umd.edu/local-cgi-bin/hcil/sr.pl...
http://kniebach.fmi.uni-konstanz.de/pub/englis...

(A) Search Result Vi.. (B) Information Vis... (C) Web Visualizati...

Currently Viewing results for: Search Result Visualization

http://www.cs.kent.ac.uk/pubs/2002/1371/
http://www.cs.kent.ac.uk/pubs/2002/1371/
http://www-dbs.cs.uni-sb.de/arbeiten/searchvisualization.html
http://csdl.computer.org/comp/proceedings/iv/2003/1988/00/19880264abs.htm

Case Study: A Combined **Visualization** Approach for WWW-**Search** ...
... the traditional **result**-list, for usage with a local meta web ... (Update) Similar documents (at the sentence level): 66.6%: **Visualization** of WWW-**Search** Results ...
http://citeseer.nj.nec.com/492831.html*20

http://citeseer.nj.nec.com/492831.html
http://www.computer.org/proceedings/spire/0746/07460008abs.htm
http://www.itl.nist.gov/iaui/vvrg/cugini/uicd/sigir-paper-jun99.pdf
http://www.bncf.net/dc2002/program/ft/poster10.pdf
http://kniebach.inf.uni-konstanz.de/pub/german.cgi/d338042/mann_webvis99.pdf
http://www.ics.mq.edu.au/~einat/info_doors/program/Cugini.pdf
http://knowledgestorm.techtarget.com/searchdatabase/search/keyword/Visualization%203%2
http://knowledgestorm.techtarget.com/searchdatabase/search/keyword/Visualization%203%2
http://www2.parc.com/istl/projects/uir/pubs/items/UIR-1995-03-Chi-Vis95-Biological.pdf
http://chiselog.chisel.cs.uvic.ca/node/view/106
http://www.computer.org/proceedings/spire/
http://www.haydenplanetarium.org/cgi-bin/search.pl?universe
http://www.haydenplanetarium.org/cgi-bin/search.pl?universe
http://www.utdallas.edu/~ilyen/course/web/zhigang.ppt
http://kniebach.inf.uni-konstanz.de/pub/german.cgi/d338042/mann_webvis99.pdf
http://www.dlib.vt.edu/Papers/CHI97DemoAbs.html

Figure 2. The SES (Search Engine Similarity) comparison tool.

ple is the SQWID [19] visualization where distance is used from Nodes determines their relevancy to particular search terms.

The principle of *symmetry* states that by placing objects in symmetry the user can more easily see associated patterns. A design example is Dynamic scrolling lists, where selection of a result that could be found in more than one list, scrolls the lists until the similar results were aligned. Ware [20] describes an application that allows users to align data in different time-series plots to identify patterns.

The principle of *similarity* states that objects sharing visual characteristics (e.g. shape, colour, texture, orientation) are understood as belonging together. Colour highlighting can be used to link similar results in different result sets. This is utilized in the SPARKLER Visualization, by Havre et al [18].

The Principle of *foreground/background* states that their is an implicit understanding of the object and its background, that being the object will stand out from its background. This effect can be caused by the relative size difference between these entities, where the uniform background space is usually perceived as the larger space, thus drawing attention to the smaller areas. color, texture or other appropriate delimiters can also cause this effect. By highlighting relevant/similar results in bright colours, and the remainder in grey, the visualization creates a division which the user will perceive as 'layers' (the foreground, and background)

The principle of *chronology* states that things that move or change in synchrony seem together. This can be applied in various ways. For example, items that are animated and move in unison will seem grouped together. Dynamic Synchronized scrolling is used in our design to this effect. This will link the users current result across several different views, and will modify each view in concordance with any changes in the current selected result. This technique is used in a number of tools, for example, Vdiff provides synchronized scrolling between two textual difference views, and North and Shneiderman use this in their Snap-together tool [3].

It is our aim to explicitly visualize the textual difference of the various search results, thus, we have identified the following design criteria that will allow users to quickly identify results that are both similar and interesting.

(1) Link Similar results between different views, so that the user remains coordinated. The Similarity principle should be used to select suitable variables to link the results.

(2) Provide a method of displaying the large amounts of data returned by search engines in the limited amount of

screen space. Techniques which should be considered include Detail and Context views, and multiple views.

(3) Provide a summary view of the data which displays an intersection of results from different search terms. A summary of the data is created dynamically from the master repository of data, this in turn can be displayed separately through the use of multiple views.

4 SES: Search engine Similarity

A screen shot of the system is displayed above in Figure 2, in this section we detail how a user operates the tool, then discuss the three sub-components of the GUI: the Summary view, Overview and bracketed view, detail the use of coordination between views, and finally describe how the tool is implemented.

User Operation. The user begins their interaction by inputting two or three sets of search terms, each of the keyword groupings should be similar in some way, for instance, each word may be a synonym of another. These terms are then submitted to Google using the Google Web API [8]. Two or three sets of search result elements (SRE's) are returned (depending on the number of terms submitted). Each SRE represent an individual url, and contains information concerning nine variables. In this work we focus on four variables: title, URL, paragraph of text and page size. The information is then visualized in three coordinated views: summary, overview and bracketed view. A user can open a webpage by clicking on the title of the result.

The summary view, shown top left of Figure 2, displays only the websites that appear in at least two search-result lists. This view consists of a tabbed pane. The main tab shows a summary view, that contains a list of URLs each with three circle-glyphs A/B/C. The circles are shaded in a value equivalent to there ranking (a darker shade of blue represents a higher ranking – see Figure 3), unless the particular result is not present in that search result, when it is colored gray. The results from each search (A, B or C) are aligned in columns. Each of the other three tabbed lists show detailed information for each specific search, including page title, fragment of the text, URL and page size. This tab-pane is shown in Figure 4).

The overview panel (bottom left of Figure 2) provides a view of the whole dataset, affording context for the other views. Comparing lists of search results in context allows the user to identify 'interesting' results in both the immediate surroundings and within the macro view of the list. This is achieved using greeking. In this view, the length of the line represents the page size and a parallel column represents each search result list. The information within each column can be ordered in several ways, such as, by page size or rank order. In our implementation each of the columns are ordered by rank. When the user selects one column (e.g. column A) this becomes highlighted and the intersections with the other lists (columns B and C) are shown in a shade equivalent to the rank of the selected overview (column A). Thus, for example, a highly ranked hit that is also contained within the other search results, but at a lower rank, will appear dark blue in each to denote its high ranking, but will appear higher up in column A, but lower down in column B and C.

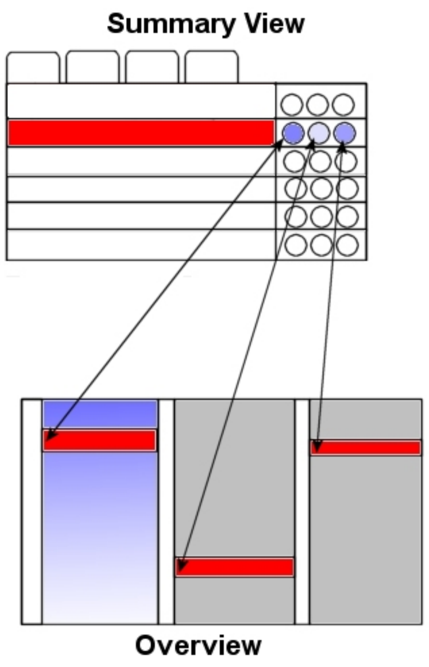
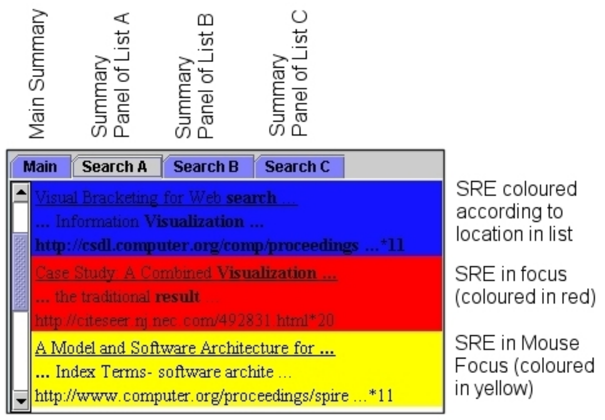


Figure 3. Circle Glyphs. Each of the circles are shaded in a value equivalent to there ranking (A darker shade of blue represents a higher ranking).

The bracketed view, depicted at the top right of Figure 2, is based on the visual bracketing principles in [10]. This is a detailed view of the currently selected list in the overview panel. The bracketing principle is a detail-in-context view, which allows a user to first focus on one search result element in detail, while viewing the surrounding context, which is achieved by reducing the detail of the surrounding text.

Coordination. When dealing with multiple views it is crucial to coordinate between these views [14]. Coordination is an important part of comparison visualization, for example when viewing a result in one search result list the user needs to be aware of the location of similar occurrences of that result in other lists. This is used in several ways in SES. First, when the user selects the focus of the summary view (bottom left) the bracketing view updates to depict the selected information, see Figure 5. Second, dynamic brush-


Main Summary
Summary Panel of List A
Summary Panel of List B
Summary Panel of List C
Main
Search A
Search B
Search C
SRE coloured according to location in list
SRE in focus (coloured in red)
SRE in Mouse Focus (coloured in yellow)

Visual Bracketing for Web **search** ...
... Information **Visualization** ...
http://csdl.computer.org/comp/proceedings ...*11
Case Study: A Combined **Visualization** ...
... the traditional **result** ...
http://citeseer.nj.nec.com/492831.html*20
A Model and Software Architecture for **...**
... Index Terms- software archite ...
http://www.computer.org/proceedings/spire ...*11

Figure 4. Summary view (tab view). Each tab contains a detailed view of each specific search, including page title, fragment of the text, URL and page size.

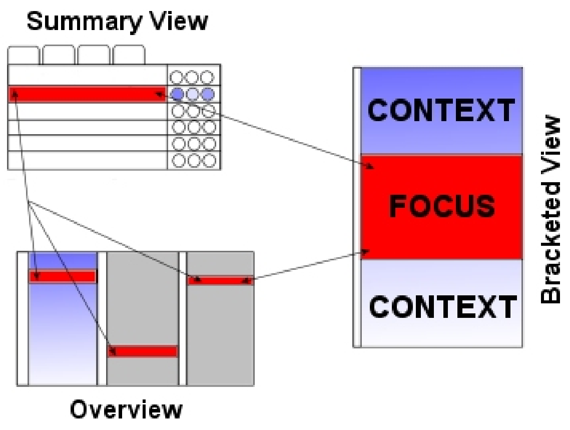


Figure 5. Coordinated views. When the user selects a result in one view, all the other views are updated and the result is highlighted in the other views.

ing is used to highlight every URL that has the same host-name (highlighted in yellow). As the user moves the cursor and brushes over a SRE every other occurrence of that SRE is highlighted, wherever it is located. Finally, every occurrence of the hostname that is focused in the bracketed view is highlighted (in red) in every other view.

Implementation. The Program was implemented using Java 1.2. Data was obtained through the Java bindings to the Google Web API [8]. The program uses a model-view-controller design pattern where the results are stored in a central data-model and the windows represent *views* on this data. Manipulating either the mouse or the scrollbars will trigger a set of Listener objects which will manipulate the visualization accordingly. The Listeners also control the colour-shading, which highlights results as the mouse cursor moves over it.

5 Results & Analysis

We have performed a preliminary user trial; each user was given a brief demonstration of the program, encouraged to interact with the visualization, and observed throughout. Afterwards, each user completed a questionnaire and was informally interviewed.

We discovered that some of the users found the controls hard to comprehend at first, but after using the 'help panel' the users understood how to control the program. The summary panel and the dynamic brushing (in yellow) were both found to be extremely useful in the comparison process. The summary panel enabled users to quickly observe relationships between similar results. Moreover, users stated that the focus highlighting (in red) enabled them to locate relevant results quickly.

The user trial revealed new problems with the GUI, Some users entered combinations of search terms which were so far removed from each other that results existing in multiple lists could not be found. A similar situation also ensued if the users searches did not return any results or if the user did not input two or three searches. One solution, that we plan to investigate in future work, would be to use synonyms based on the search terms.

Another problem that the user study highlighted was one of scalability. Ie., how does the system cope with large volumes of search results, or more than three queries.

Currently, the system, by default, displays a maximum fifty search results for each input term A/B/C. If the user requires to view more than fifty results then the major challenge is to accommodate these extra results in the overview-panel. Both the summary and bracketed view naturally expand to the number of results presented, however overview-panel displays a complete 'overview' of the data in one view. A solution could be to place the greeked lines closer together to fit more data in. But, this would not make much space saving. Alternatively, the overview could be generated using an alternative design such as pixel based techniques [5], or even a mixture of greeked lines and pixel based techniques in a bracketed view [10].

Comparing more than three query terms is perhaps the more interesting design challenge. For instance, it would be feasible to imagine that a user would wish to compare their last four or five searches (from a search history). Solutions to this are ongoing work. One of our ideas uses non-textual similarity (an abstract graphical realization) alongside textual-difference visualization, where the graphics show the overview and the text view shows the detail.

6 Related Work

Textual difference visualization has been used in a number of applications, from general file-differencing tools to more specific applications such as program code analysis, literature comparison, plagiarism detection, and finally search result visualization.

Most of the file-difference tools can be classified as 'side-by-side with cues' techniques, as shown diagrammatically in Figure 1. These tools typically emphasize the differences between the versions rather than emphasizing similarity. Often the differences form a smaller set, and thus can be more easily pre-attentively visualized. For instance, Vdiff [6] displays two parallel, synchronized views; colors emphasize the parts that are dissimilar, and similar areas of the text are enclosed in a bounding box, the boxes in each of the columns that are similar are joined by explicit lines. Both parallel views are scrollable and they align to keep identical parts adjacent. These tools can also be used to analyze program code.

Code development is often done in incremental stages by many developers. Thus, as parts of the code are written, tested, enhanced and extended many versions of the code are generated. These are often stored in a version-control repository (such as CVS). Consequently the textual difference tools enable a developer to better understand incremental changes, identify missing code or errors in the code. There are many tools available including [1, 2, 7, 21]. Most of the tools share the same basic textual format where the viewing window is divided into two (or three) principal sections: one for each file. Usually the differences in each of the files are highlighted; different colors represent added lines, deleted lines, and changed lines. Some tools such as WinDiff [21] merge the information into one view [23], so different colors are used to represent similar and different text. SeeSoft visualizes program text and version control using greeked lines [17]. Greeking exchanges the characters of the text with straight lines and hence is useful for providing summary information about a document. Robertson and Mackinlay use greeking as part of their Document Lens [15]. Some of the code visualization tools also provide additional facilities such as to allow the user to edit the files that are being differenced or highlighting the location of minor differences such as letter changes or capitalizations.

As well as visualizing differences in files and program code, textual difference can be used to investigate corpus information. ItLv (Interactive Timeline Viewer) [12] by Monroy et al. is a tool that can be used to visualize variations among documents. They use a timeline viewer, where the X-axis corresponds to pages in the book, and the Y-Axis displays multiple rows (one for each edition being compared). Small bars along each progress line depict the differences of the editions; the height of the bar denotes the number of differences. Monroy et al. use this to examine various editions of Don Quixote.

Plagiarism detection is another use for textual difference tools, something that is useful in an educational environment. Websites such as Turnitin.com and wordchecksystems.com offer plagiarism software, which utilizes textual difference to show where students have copied work from Internet sites. The distinction with these tools is that, although they adopt a similar side-by-side approach to difference multiple files, they can compare against versions stored in a work repository, or against information on the web, thus deterring students from retrieving texts from homework repositories or merely copying text from the web. Abstract comparison and abstract visualization is also used in plagiarism detection [4, 13].

There are a number of tools which compare search result data without using text, these abstract comparison tools, such as SQWID [19] and SPARKLER [18], plot objects within a two-dimensional axis, instead of representing the text. SQWID, for example, visualizes search results by submitting a query, and then locate 'interesting' words from the returned set of results. These words then become term nodes, and each of the results returned, represented by page nodes, will gravitate towards different term nodes depending on their relevance to the term. Sparkler displays its results as dots along multiple lines drawn from the center of the view.

Some tools utilize objects within a 3D space to substitute the text. VR-VIBE [16], for example, creates visualizations of bibliographies. The user specifies keywords that they wish to use to generate the visualization and place these keywords in 3D-space. Representations of each document are then positioned in the space according to how relevant each document is to each of the keywords. Scatter-plot graphs are another non-textual method of comparison. The 3D scatter plot was part of a series of visualizations for the NIRVE [11] project. Users enter three search terms that become the three axis of the scatter plot; the results are placed within the scatter-plot according to the number of times that the each of the search words appears in each of the results.

All of the abstract search-result visualizations described use the Gestalt's similarity principle. The use of color is a prominent characteristic shared between nearly all the visualizations. Non-textual visualizations also take advantage of characteristics such as distance to show similarity (as used by [18] and [19]).

7 Future work & conclusion

We have developed and implemented a difference visualization that compares multiple search result lists. Our system is based on the 'side-by-side with cues' category, and uses Gestalt principles to implement the comparison. Indeed, we have used multiple views, dynamic brushing and highlighting techniques to aid in result comparison, and have implemented a detail-and-context view to aid in viewing large data sets. Our user study has shown that users consider the summary view and color highlighting to be particularly useful tool for locating similar results.

We believe that such textual comparison is useful, and that it should be readily available for everyday users. Currently this is not the case. There is much to improve and many other ideas to explore. For instance, it would be useful to apply the other comparison categories of overlaying, merging and abstract views to search-result comparison. There has been some research in the application of abstract views to search-result visualization (as detailed in section 6), and some work has been done on overlay and merged views for other application areas.

Apart from further improvements of SES, such as improving the greeked panel and allowing two, three or more comparisons, employing other (exact or fuzzy) matching criteria's (such as hostname, filename, page size or content), or using different search engines, we envisage that it would be useful to have a search result difference visualization that is fully flexible and integrated into the browser. For example, it would be favorable to continually add searches, remove searches from the comparison, compare subsets of returned results, and to do this natively (within the browser). Ideally, the comparison would be achieved silently and historically, such that a user can compare any previously searched elements (even from previous days) without explicitly requesting such an action.

Acknowledgements

We thank all those people who have tested the software especially for their comments and input into this work.

References

[1] Araxis Merge Tool. www.araxis.com/merge, February 2004.

[2] Beyond Compare, advanced file and folder comparison utility. www.scootersoftware.com/, February 2004.

[3] C.North and B.Shneiderman. Snap-together visualization: A user interface for coodinating visualizations via relational schemata. In *Advanced Visual Interfaces*, pages 128–135, 2000.

[4] F. Culwin and T. Lancaster. Visualising intra-corpal plagiarism. In *5th International Conference on Information Visualization (IV '01)*, pages 289–296. IEEE, July 2001.

[5] D.A.Keim, M.C.Hao, U.Dayal, and M.Hsu. Pixel bar charts: a visualization technique for very large multi-attribute data sets. In *Information Visualization (InfoVis)*, pages 20–34. IEEE, March 2002.

[6] D.J.Barnes and M.T.Russell and M.C.Wheadon. Developing and adapting UNIX tools for workstations. In *EUUG Conference Proceedings*, pages 321–333, 1988.

[7] ExamDiff visual file comparison tool. www.prestosoft.com/ps.asp?page=edp_examdiff, Feb 2004.

[8] Google Web API. http://www.google.com/api, Feb 2004.

[9] English Hexapla new Testament, 1841.

[10] J.C.Roberts and E.Suvanaphen. Visual bracketing for web search result visualization. In Ebad Banissi et al, editor, *Proceedings Information Visualization (IV03)*, pages 264–269. IEEE Computer Society, July 2003.

[11] J.Cugini, S.Laskowski, and M.Sebrechts. Design of 3d visualization of search results: Evolution and evaluation. In *Proceedings of IST/SPIE*, Visual Data Exploration and Analysis, pages 198–210, January 2000.

[12] C. Monroy, R. Kochumman, R. Furuta, and E. Urbina. Interactive Timeline Viewer (ItLv): A tool to visualize variants among documents. *Lecture Notes in Computer Science*, 2539:39–49, 2002.

[13] R. Ribler and M. Abrams. Using visualization to detect plagiarism in computer science classes. In *2000 IEEE Symposium on Information Visualization (InfoVis)*, page 173. IEEE, October 2000.

[14] J. C. Roberts. On Encouraging Coupled Views for Visualization Exploration. In R. F. Erbacher and A. Pang, editors, *Visual Data Exploration and Analysis VI, Proceedings of SPIE*, volume 3643, pages 14–24. IS&T and SPIE, January 1999.

[15] G. G. Robertson and J. D. Mackinlay. The document lens. In *Proceedings of the 6th Annual Symposium on User Interface Software and Technology*, pages 101–108. ACM Press, November 1993.

[16] S.D.Benford and D.N.Snowdonand and C.M.Greenhalgh and R.J.Ingram and I.Knox and C.C.Brown. VR-VIBE: A virtual environment for co-operative information retreival. *Computer Graphics Forum*, 14(3):349–360, 1995.

[17] S.G.Eick, J.L.Steffen, and E.E.Sumner. Seesoft - a tool for visualizing line oriented software statistics. *IEEE Transactions on Software Engineering*, 18(11):957–968, November 1992.

[18] S.Havre, E.Hetzler, K.Perrine, E.Jurrus, and N.Miller. Interactive visualization of multiple query results. In *2001 IEEE Symposium on Information Visualization (InfoVis)*, pages 105–112. IEEE, October 2001.

[19] S.McCrickard and C.Kehoe. Visualizing search results using SQWID. In *Proceedings of the Sixth International World Wide Web Conference*, Apr. 1997.

[20] C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers, 2000.

[21] WinDiff Tool, microsoft sdk tools, February 2004.

[22] W.Wertheimer . Laws of Organization in Perceptual forms (Untersuchungen zur Lehre von der Gestalt II). *Psycologische Forschung* , 4:301–350, 1923.

[23] W. Yang. How to merge program texts. *The Journal of Systems and Software*, 27(2):129–141, November 1994.