BasketLens: interface for document visualization and exploration

Darya Filippova [dasha.filippova@gmail.com]
University of Maryland, College Park
May 17, 2007

Final report
Independent study conducted with Ben Shneiderman and Catherine Plaisant

Abstract

BasketLens is a novel tool that allows users to visualize and browse through the document collection. Users can perform word search and search for user-defined groups of words, and then visualize search results in a way that helps uncover the patterns within the document collection. BasketLens was developed as an aiding tool for the researchers at Maryland Institute for Technology in the Humanities (MITH) to explore the eroticism in Emily Dickinson's poems. This paper describes the design and implementation of the BasketLens, users' feedback on the tool, and future work.

1. Motivation

A substantial effort has been put into digitizing books, poetry, historical documents. Some literary and historical collections are accessible through Internet, for example [1, 13, 14]. While it is usually possible to search for a particular document (by title, author, date, language), only few collections support searching across several documents. Researchers can as well study the documents one by one and take notes, but notes can get disorganized, paper notes can get lost. BasketLens offers a convenient way to search across a collection of documents.

2. Introduction

BasketLens grew out of the Emily project [8]. Emily started as a tool developed by Nitin Madnani at the University of Maryland. A year later, Emily was brought back to life by Ben Shneiderman and Catherine Plaisant.

MITH envisions itself as an "applied think tank for the digital humanities". MITH researchers working with Emily Dickinson's poetry and supporting Emily Dickinson Electronic Archives [1] wanted to have a tool that would help the researchers to explore the collection of Emily Dickinson letters-poems. They had particular expectations: the tool should provide search

capabilities and visualize the results in a clear and concise manner; there should be a way to incorporate the data from the XML tags associated with the poems. A question of Dickinson's sexual orientation was one of the many the MITH scholars wanted to find answers for, so they requested that a tool has a way of searching for groups of word that are unified by a common idea (for example, eroticism or sexuality). One of the important aspects of the work was the level of users' computer education which could vary from unexperienced to proficient. Some of the researchers were reluctant to use software for the purpose of literary research. The goal of the project was to come up with an easy design that meets the expectations and encourages people to use the tool.

Since Emily project can be potentially used with collections other than Dickinson's poetry, it was renamed to BasketLens.

3. Related work

This paragraph explores the projects that share similarities with BasketLens in goals and implementation.

Nora Visualization Tool was developed as a result of research efforts of the University of Maryland and University of Illinois scholars [2, 4]. The Nora Visualization Tool rates Dickinson's poems based on the training set of data. Nora aids the discovery of the "erotic" words and phrases and can be used for document analysis. Nora, however, does not specialize in word search within the poems.

The Compus project resembles BasketLens in several aspects [5]. Like BasketLens, Compus works with a collection of manuscript letters available in XML format and offers an opportunity to examine all the documents at once. Compus' central idea is the visualization of the manuscripts. However, Compus does not support word search.

The GRIDL project [6] employs an idea of using two axes to represent information. BasketLens uses similar approach for the table view: the document names form one axis and search terms form another axis. Unlike fixed axes in GRIDL, the search term axis in BasketLens is adjustable.

The InfoMagnets software [7] uses a Latent Semantic Analysis (LSA) for deriving the documents' topics and rates the documents according to the created categories. User can break the existing categories into several ones, but each new category can only use the same words as

in the initial category. Unlike BasketLens, InfoMagnets does not have an option of creating user categories.

Other tools like PaperLens [15] provide complicated views of statistical data which scholars might find difficult to understand. Systems similar to [16] offer capabilities for manuscript analysis, but they do not support search.

Search by a category such as a search by state, symptom, author or topic are widely represented in the online search databases and offline complimentary tools. BasketLens goes beyond this approach and offers an opportunity to search for user-defined categories of words [3, 12].

4. Definitions

This section defines the terms used in the article:

word basket – a group of words unified by some concept. A basket can be either user-defined – user picks the words that become basket contents and the name for the basket; or a basket can be predefined – imported from an outside source.

search term – a word or a word basket that the user wants to search for.

search, or a *bSearch* – the user can think of the BasketLens search as a process of finding the matches to the search terms within the documents. The result of such a search is a collection of search terms matched with the list of poems and poem lines the term occurred in. When the user searches for a word basket(s), the engine finds references for every word from the basket in every document and sums up the number of matched words for each document. This way, if there were a DocumentA, a DocumentB, and a basket `"house"` consisting of `"home house room door stair"`, then the bSearch will tell that DocumentA and DocumentB each matched three times (see Fig 1).

DocumentA	**DocumentB**
With one light on in one room,	My lover's gone, his boots no longer by my door,
I know you're up when I go home	He left at dawn, and as I slept I felt him go.
With one small step upon the stair,	Returns no more, I will not watch the ocean
I know your look when I get there	My lover's gone, no earthly ships will ever
	Bring him home again, bring him home again.

Figure 1 Basket search

grouping basket contents – when the user searches for the word basket, she has the option of combining the search results altogether, or viewing them on a word per word basis, or *ungrouping* (more about grouping - below).

5. BasketLens interface

The UI for BasketLens had undergone several UI changes, but the tool's simplicity and discoverability of its features was always a prime objective. I will describe the elements of the UI in the order they would be used by a literary scholar. Let's assume that the goal is to find out how often the subject of pleasure is mentioned in the documents.

First, the user loads the data. using the load panel in the right top corner. After the user selects the file and BasketLens processes it, the panel updates to show statistics about the collection of documents and the original data file. The location of the data file is saved as a parameter and thereafter that data file will be loaded automatically on startup with every run of BasketLens.

Following the data load event, BasketLens updates the tabs responsible for visualization of data. In the table view, the user finds a new column "Documents" that lists the documents names (in the case of Emily Dickinson poems "Documents" column contains first lines of the poems since Dickinson did not give names to her letter-poems). The document view shows all the documents at once sorted in alphabetical order. The user can access the documents by scrolling up and down in either view. If the user clicks on a document or its name, the document contents will be loaded into the preview window (on right side in the middle).

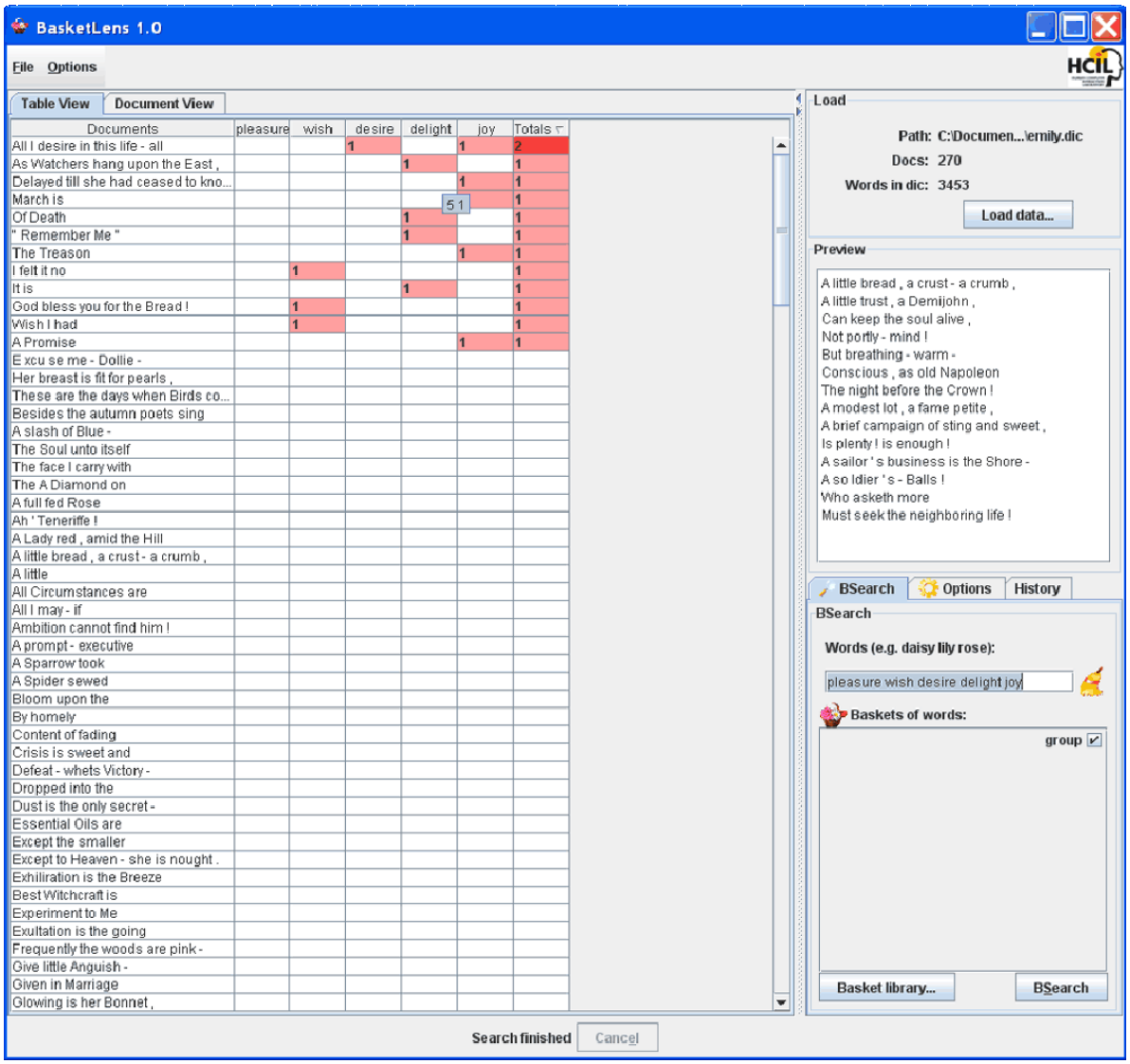

Figure 2 Search for pleasure words: table view creates columns for each search word, summarizing the results in "Totals" column.

Now that the data is loaded, the user would like to look for words referring to pleasure, for example "`pleasure wish desire delight joy`". The user clicks "BSearch" and in an instance the results appear in the table view (Fig 2). "Totals" column shows a sum of the number of matches for all search terms within one poem. The results are sorted in descending order by the "Totals" column. The user can tell that while there was no mentioning of the `pleasure` itself, the word `wish` occurred in three different poems, there were references to `desire`, `delight` and `joy`. The user goes through the poems and finds other words that occur near the search terms and that refer to pleasure: `rapture`, `heaven`, `paradise`, `love`, and `loving`. Adding these words to the search field and searching again yields another picture (Fig 3).

In the Basket Library, there is a predefined[1] `pleasure` basket. For the user, it might be interesting to compare his results for pleasure words and the results of searching for the predefined `pleasure` basket. To compare search results of the `pleasure` basket and user-chosen words, the user goes to the basket library, saves his search words as `my pleasure basket`, selects the predefined `pleasure` basket and clicks "BSearch" (Fig 4).

The user can see that while his basket matched 4 word in the poem "`All I desire in this life – all…`", the Inquirer basket has matched only one word.

If the user becomes curious what that word was, he should uncheck the "group" checkbox in the search panel.

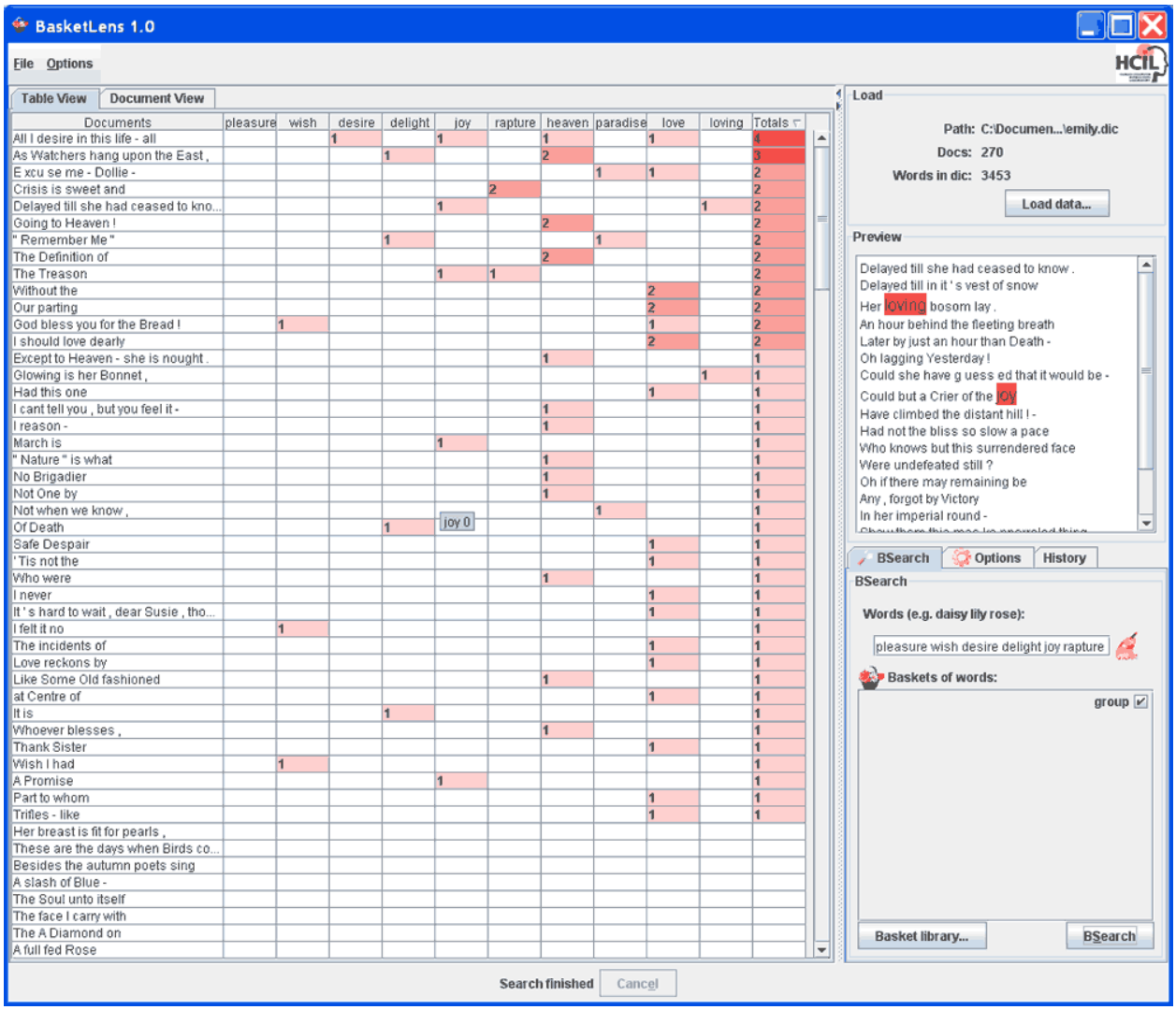

Figure 3 Searching for pleasure words: the user has added "rapture heaven paradise …" to the search words. For each new search word, table view adds a column that reflects the number of matched words in each document. "Totals column", right, shows the matches per document.

[1] All predefined baskets were borrowed from the Inquirer [9] project.

Unchecking the “group” checkbox tells BasketLens that the user wants the contents of the basket to be represented as multiple columns, one per each basket word. Since the `pleasure` basket contained 80 words, there will be 80 columns for the basket. To fit them all on the screen, the user switches to the “Options” tab (right bottom side) and checks the table view flags. “Shrink columns” and “Shrink rows” try to fit the columns/rows in a given space by recalculating the columns width/row height. “Hide empty columns” removes the columns that have no matches within (i.e. the corresponding search term was not found in any document). “Hide lines” makes the table view clearer by hiding the borders between rows and columns. This option is useful when the columns/rows are too narrow and borders obscure the view. “Hide numbers” will hide the cell weights. The option does not affect the color coding or the results of the search and is purely “cosmetic”. The “Color range” panel offers two options for calculating the brightest/dimmest color: if the user selects “Maximum per column”, the BasketLens will calculate the maximum number of matches per column (i.e. per search term) and will assign an individual color range to every column based on its maximum.. If the user selects the “Maximum per table” option, then the absolute maximum of matches will be found (usually it is a cell value in “Totals” column) and the color intensities will correspond to the values 0 through the maximum per table.

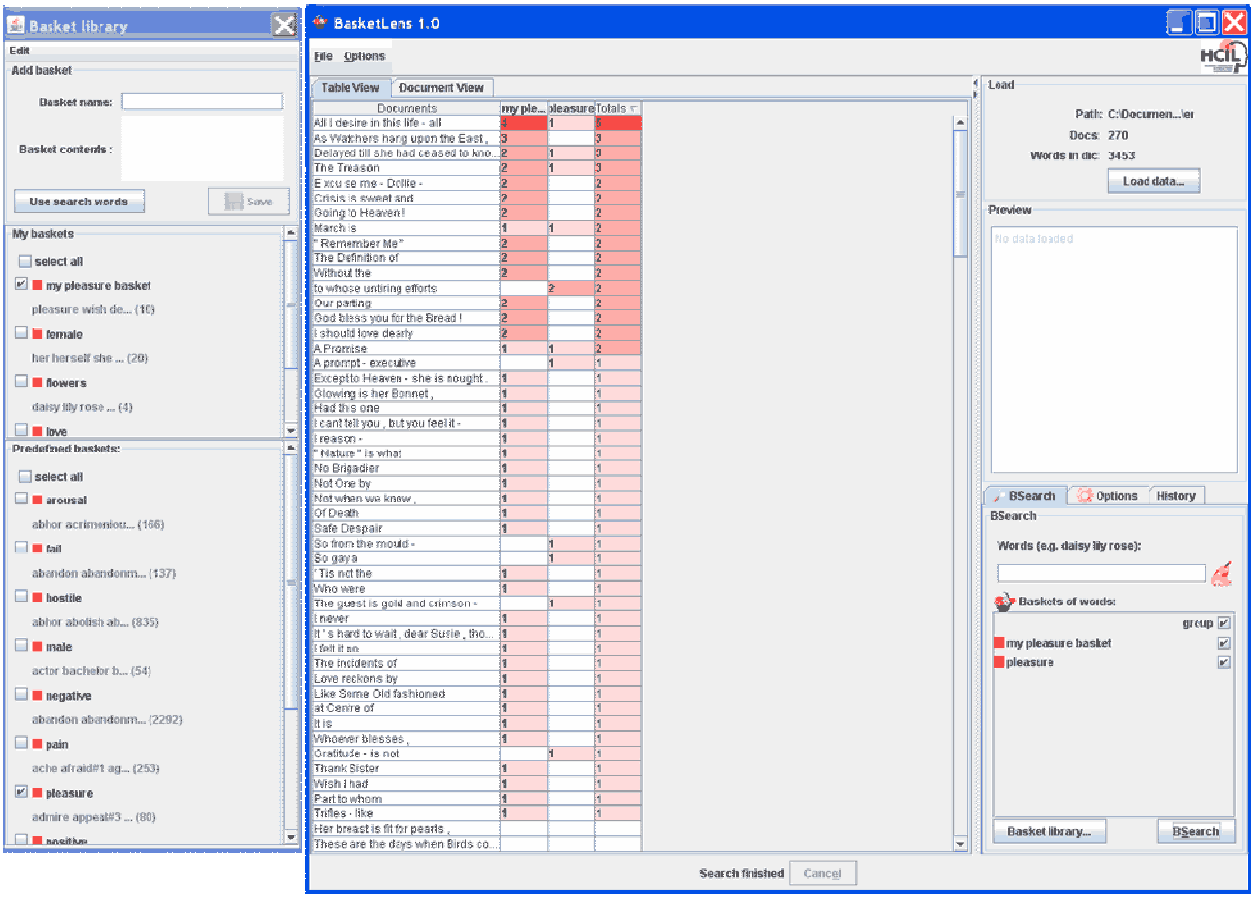

Figure 4 Comparing predefined and user basket: the table view shows one column corresponding to "my pleasure basket" and one column corresponding to "pleasure" predefined basket.

Switching to document view will adjust the "Options" panel respectively: while working with the document view, the user can see the document view options only. In this view, the user can scroll through the full texts of the documents. The documents with the most references will be brought up to the top. The user has the option of reordering the documents alphabetically or by the document length (lines).

An option shared by table and document views, "hide documents with no matches", does what its name suggests: it hides the documents that have no matches in the document view and shrinks the appropriate rows to the minimum height in the table view.

If the user is interested in viewing additional information about documents, she can load a CSV[2] data file. For the data in CSV file and the corpus of documents to be aligned correctly, the CSV file must include a column with the same names for the documents that were used in the documents' data file. To load CSV file, the user goes to the `File->Load CSV file...` and

[2] CSV – comma-separated values file format that stores tabular data.

chooses the appropriate file. Once the file is loaded into the system, the user will see the list of column tags available. Checking/unchecking the tags sends a message to the table view to create a new column and fill it with the data from CSV file (Fig 5). Just as other columns in the table view, the user can sort the CSV data columns by clicking on a column header. Cells containing dates will be compared numerically; cells containing strings will be compared alphabetically.

A click on an icon of a broomstick next to the search field will erase the results of the previous searches and refresh the screen. It also clears the search fields.

The history panel (right bottom corner) shows a list of all previous search queries made in the current run of BasketLens. Clicking on either entry will load the contents of the history entry to the search panel and the user can restart the search. Clear button will clear the history, but will not erase any other data.

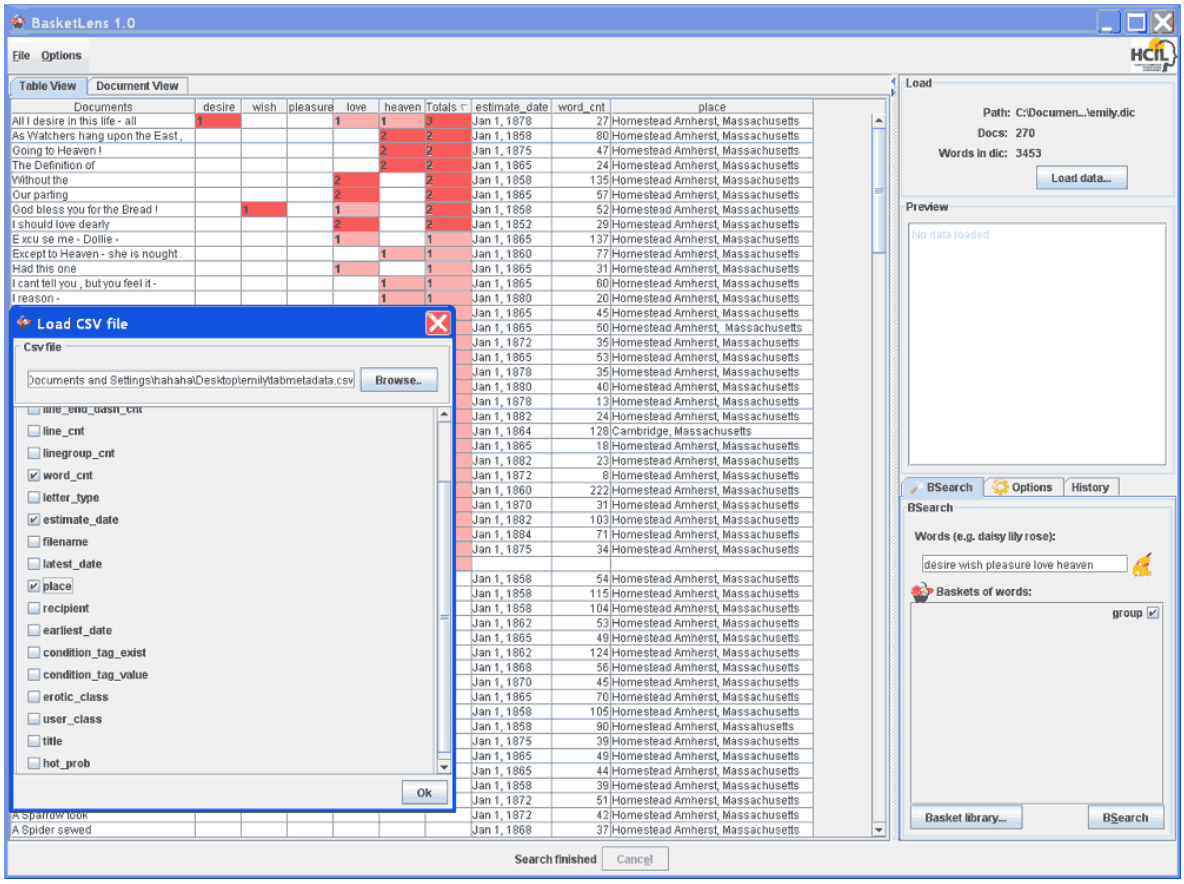

Figure 5 Loading CSV file: selected tags force new columns to appear in table view filled with data from the CSV file.

Basket Library window (Fig 5) views two lists of baskets: a list of modifiable user-created baskets and a list of unmodifiable predefined baskets borrowed from Inquirer project [9]. By selecting/ deselecting a basket entry, the user can add/remove the basket to the bag of search

terms in the search panel. Right-click on the basket entry opens a popup menu with available basket operations. “Copy to My baskets” options creates a new instance of the basket, assigning it the same contents and appending “copy” to the end of name. “Copy to My baskets” makes sure there is no basket with such a name; otherwise BasketLens offers the user to rename the basket. “Color” brings up a color chooser and updates the user’s choice of color for the current basket. “Edit” is enabled only for user baskets. Predefined baskets cannot be modified, but their copies can. Edit command brings up a window with the contents of the basket already loaded. The user can add or remove terms just as if she was using a text editor. Like “Edit”, “Rename” is enabled only for the user baskets; the command brings up a window asking for a new name. Deletion will work only for the user baskets and delete operation erases the basket from the collection permanently. The user and predefined baskets are read in on the BasketLens startup; the user baskets are saved from one run of a program to another in the `baskets.save` file.

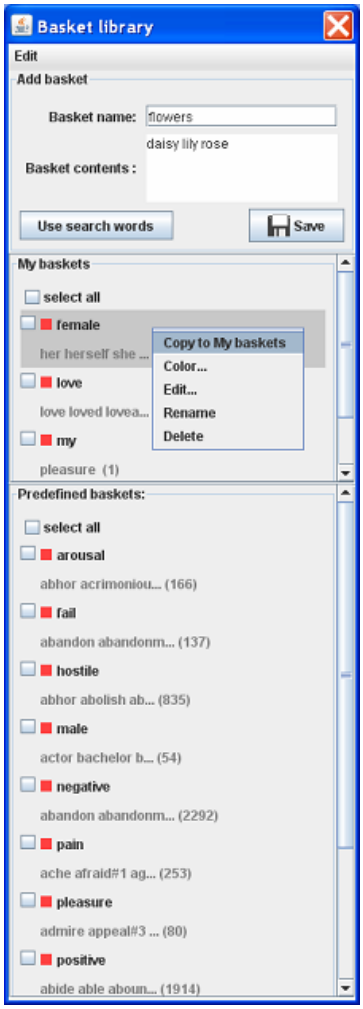

Figure 6 Basket Library window lists all user-defined baskets and all predefined baskets. The user can create a new basket typing the search words in the "Words" field or by loading the search words (button "Use search words")

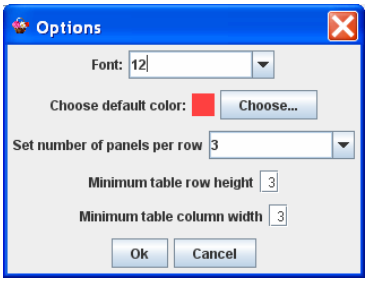

Figure 7 In "Options" window, the user can change document view font, default color that is used to highlight matched words, document view layout and more.

By selecting `Options->Options` in the menu at the top of the screen, the user discovers a variety of options (Fig above). The font option sets the font size for the full text documents in document view. The default color sets the color used in the views to represent the matched words. “Number of poems per column” is a debugging option which controls the layout in the documents view. Minimum values for the height and width control how compact the table in table view can be (see table view options above).

6. User evaluation

For testing purposes and to gather users’ feedback, several users worked with BasketLens. Three of them worked at MITH and were among the intended users. They were computer-educated and open to using computers as an aiding tool in literary research. Two other users worth mentioning were computer science undergraduate students.

The first user from MITH was not familiar with the project and tried using it for the first time during the presentation of the BasketLens. He had some trouble figuring out the sequence of actions that produces the result in BasketLens. He advised to make the tool more animated so that the users follow the flow of actions and data. The user also suggested that drag and drop feature would make the tool more attractive for him. He liked the general idea of BasketLens software, its ability to search for word within a set of documents. He also appreciated the introduction of the word basket which he said was a novel and a very useful idea.

Another user from MITH, a professor of English literature at University of Maryland specializing in Emily Dickinson’s poetry, had tried the tool during a test session. She was first given a BasketLens demo, and then she tried the tool herself. She reported that she liked the idea of word baskets – it was the feature her colleagues and she were missing in other tools. She

appreciated that there were predefined baskets and that the user could perform some operations on those. She was glad that BasketLens could load additional data through a CSV file. She found the table views an easy-to-understand way of visualizing search results. She had some suggestions about predefined baskets, such as offering the "female" basket to the users, and helped in testing the tool. In general, she enjoyed her experience and said that the tool was not too complicated.

Yet another user from MITH, a graduate student in Library Sciences program, who works closely with computers and digitizes literary documents, tested the tool during a demo session. She was given a short presentation and then she was given an opportunity to try the tool herself. She reported that the tool was easy to use and did not require her to spend too much time thinking about the next move. She found the interface simple and easy to understand. She was curious about using sets of documents other than the collection of Emily Dickinson's poems. She made some suggestions on the future features, for example, the phrase search. In general, she was very impressed with the tool and said that she though it highly useful for literary research.

A first undergraduate computer science student was suspicious towards software that helps in the literary analysis. Nevertheless, he tried BasketLens, tested the interface, and followed the instructions to find references to flowers mentioned in the poem. He was not impressed with the interface calling it too simple and, being unfamiliar with the needs of literary researchers, he expressed doubts in BasketLens's usefulness.

A second undergraduate computer science student was disposed towards trying the software and was interested in discovering the features and their effects on the data set. She examined every tab and button, then went to "Load" panel and asked for an appropriate data file and its location. She initiated several searches herself trying switching between different options. She then performed a task of searching for pleasure words. After several iterations, she asked if she could save the words and followed to the basket library. She tested baskets functionality and found it useful. She enjoyed her experience, especially the viewing options for the table view.

Overall, users were interested in trying BasketLens and experimenting with it. Users' feedback has given some ideas for further exploration, such as introducing the phrase search and manuscript visualization. Users' reaction also helped to outline the most useful/wanted features, such as baskets and basket operations and the ability to tune the search results view. The performance of unexperienced users shows that some of the features are underdeveloped and

need further investment of time, for example, it is worth to pay attention to discoverability of all the options.

7. Limitations

BasketLens project is in the process of developing new data import functions so that various data formats could be supported. For now, BasketLens can work with short (1-10 lines) poems and poems of a medium length (11-50 lines). Poems that are longer were not part of the Emily Dickinson collection and BasketLens was not tested on those. Apart from document length, document format is a cause for concern as well. Usually poems' lines are narrow or at least of the same length from one line to another, while documents in general are compose of paragraph of various lengths. Viewing such collections in the document view can be problematic.

8. Future work

In the future, BasketLens would benefit if it uses an external tool, like WordNet [10], to retrieve the word baskets given a base term. User evaluation suggests that phrase search and manuscript viewing were among the available options. It also can be a good idea to develop the set of operations on the basket and build an online repository accessible for everyone.

9. Acknowledgements

I would like to thank Catherine Plaisant and Ben Shneiderman for their guidance in the development of the project, fascinating ideas and encouragement. I also would like to thank MITH faculty for testing BasketLens and providing feedback on the tool.

References

1. Dickinson Electronic Archives. See: http://www.emilydickinson.org, 1994-2007.
2. Matthew Kirschenbaum. Poetry, Patterns, and Provocation: The Nora Project. The Valve – a Literary Organ. See: http://www.thevalve.org/go/valve/article/poetry_patterns_and_provocation_the_nora_project/, 2006.
3. Deborah Parker. The World of Dante. See: http://www3.iath.virginia.edu/dante/, 2006.
4. Catherine Plaisant, James Rose, et. al. Exploring erotics in Emily Dickinson's correspondence with text mining and visual interfaces. *Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries*, Chapel Hill, NC, United States, 2006. 141-150.
5. Jean-Daniel Fekete, Nicole Dufournaud. Compus: visualization and analysis of structured documents for understanding social life in the 16th century. *Proceedings of the fifth ACM conference on Digital libraries,* San Antonio, United States, 2000. 47-55.
6. David Feldman, Xavier Ferre Grau, et. al. Visualizing Digital Library Search Results with Categorical and Hierarchical Axes. *Proceedings of the fifth ACM conference on Digital libraries,* San Antonio, United States, 2000. 57-66.
7. Jaime Arguello, Carolyn Rose. InfoMagnets: making sense of corpus data. *Proceedings of the 2006 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: companion volume: demonstrations,* New York, United States, 2006. 253-256.
8. Nitin Madnani. Emily: A Tool for Visual Poetry Analysis. See: http://www.umiacs.umd.edu/~nmadnani/emily/emily.pdf, 2005.
9. Inquirer Home Page. See: http://www.wjh.harvard.edu/~inquirer/, 2002-2007.
10. WordNet 3.0. See: http://wordnet.princeton.edu/perl/webwn?s=word-you-want, 2006-2007.
11. The PLANTS Database. See: http://plants.usda.gov, National Plant Data Center, Baton Rouge, LA 70874-4490 USA.. See also: http://plants.usda.gov/characteristics.html. 10 May 2007.
12. National Digital Library Program. See: http://memory.loc.gov/ammem/dli2/html/lcndlp.html, 10 May 2007.
13. Einstein Archives Online. See: http://www.alberteinstein.info/, 17 May 2007.

14. The Oxford Text Archive. See: http://ota.ahds.ac.uk/, 17 May 2007.
15. Bederson, Ben B., Czerwinski, Mary, Lee, Bongshin, Robertson, George. Understanding research trends in conferences using PaperLens. *Conference on Human Factors in Computing System. CHI '05 extended abstracts on Human factors in computing system,* Portland, United States, 2005. 1969 – 1972.
16. E. Lecolinet, L. Likforman-Sulem, L. Robert, F. Role, J-L. Lebrave. An integrated reading and editing environment for scholarly research on literary works and their handwritten sources. *International Conference on Digital Libraries. Proceedings of the third ACM conference on Digital libraries,* Pittsburgh, United States, 1998. 144-151.

Appendix A.

Implementation details

BasketLens code is written in Java and consist of approximately 4000 lines. The program was run on a PC computer with 128MB of RAM and 2.00GHz processor. The code is divided into two parts: the code responsible for the UI and the code responsible for the operations with data. I will first describe the data structures supporting the basket library and search, and then I will give an overview of the UI classes and interactions between them.

A1. Dictionary

The Dictionary class is responsible for loading, parsing, and storing the documents' data. It can read in and write out the dictionary file format. This class can read in XML files, parse the data within and store the processed data in the dictionary format.

A2. Basket

Basket class unites such elements as a basket name, basket contents, and various flags into one data structure. It also offers convenient methods for operating with basket: adding/removing words, renaming the basket, saving basket color, etc.

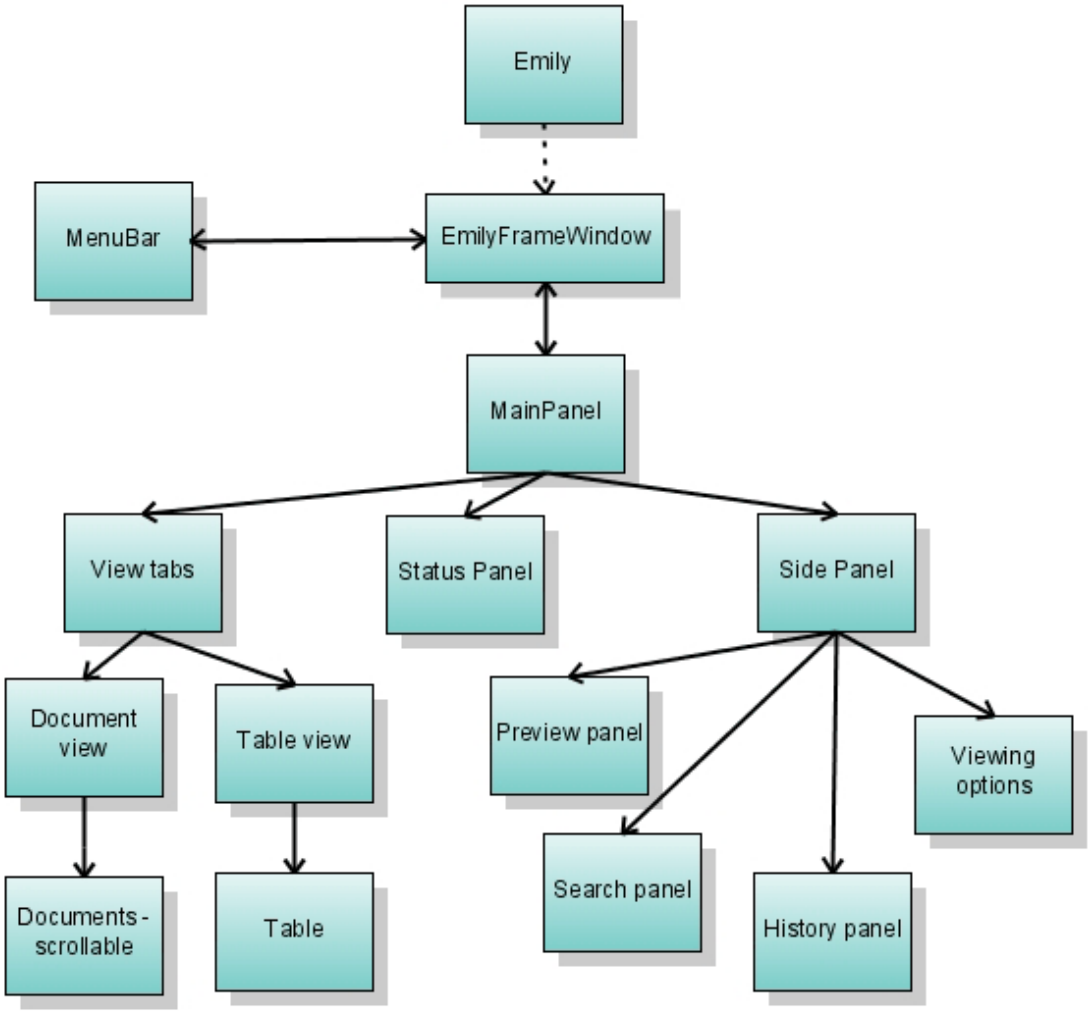

Figure 8 UI diagram

A3. BasketCollection

BasketCollection class maintains a set of baskets, mapping basket names to the basket entities. BasketCollection is responsible for creating new baskets, updating basket's contents, name, color, and flag states, deleting baskets, retrieving the contents of the basket, copying baskets. BasketCollection is an intermediary between the BasketLens classes and individual baskets. When a new basket is created, it is assigned the default color.

A4. SearchTerms

SearchTerms data structure maintains a collection of search terms and baskets along with their colors. Instead of maintaining separate lists, as was the case in Madnani's implementation, each term is bundled with its descriptive elements. The lists can easily get out of sync while the SearchTerms structure stores the data together. When a new search term, whether a word or a

basket name, is added to the SearchTerms structure, the term is automatically assigned the default color.

A5. UI classes

BasketLens's UI is written using Java Swing classes. Each class in the system is responsible for some part of UI such as a document view or a side panel (Fig 6). It is a big change from the Madnani's implementation: in the early versions, one class was responsible for all the functionality and UI which made it very difficult to navigate and add changes. I started with refactoring the code, but later decided it would be easier to rewrite the UI part from the beginning. Now the BasketLens code consists of multiple classes with divided responsibilities.