

SortTables: A Browser for a Digital Library

William C. Wake* and Edward A. Fox*#
Department of Computer Science* and Computing Center#
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0106
{wakew, fox}@cs.vt.edu

Abstract

Much research in information retrieval has focused more on matching results to queries than on browsing those results. After briefly exploring browsing in physical and electronic libraries, we introduce SortTables, a new system that focuses on support for browsing. We explore the evolution of the system in light of early implementation experience and formative evaluation of the interface. Finally, we briefly review related work, and discuss future directions.

1 Browsing in the Physical Library

In a physical library, we think of browsing as moving among the shelves, looking at items of possible interest. Several characteristics of this type of browsing stand out: access to the actual items, potential access to all the items, a sense of neighborhood, variable focus, and the opportunity for serendipity.

Physical browsing provides access to the actual items. For example, books provide powerful cues about their meaning in a situated way, through physical cues as well as content. A worn-out book might be regarded as more (or less) interesting because of its apparent popularity. A thick book might look like too much work; one with an interesting cover might be chosen instead.

Browsing potentially provides access to **all** the items. One has the sense that one could start at one end and go to the other, looking at each book. This is different from trying to retrieve books by generating queries: we don't have to know all the "query terms" in advance, and we don't have to deal with the same item twice (because it was retrieved by different queries).

The sense of neighborhood is an important reason that browsing is effective. Since books are organized by call number, and call numbers are assigned in a way that keeps related books together, we often find books on related topics close together.

Browsing allows a variable focus: in a promising area, one can systematically examine all items; when items in that area aren't useful, one can make a sweeping movement

through large areas, skipping the uninteresting ones.

Finally, browsing allows for a natural serendipity: one can choose a book at random from a particular area, or from a random area of the library. As with variable focus, one has control over how random one's selection can be.

2 Browsing in the Electronic Library

Browsing in the electronic library can retain many of these characteristics, except for access to the physical items.

Electronic browsing can provide something not feasible in the physical library: multiple orderings of the items. We're not restricted to ordering strictly by call number: we can think of browsing by author or date. (One could imagine a library where the three-dimensional arrangement has significance; for example, the floor of an item could correspond to year of publication or publication type. However, the particulars of the three-dimensional arrangement are usually only accidental, as shelves are usually linearly organized by call number.)

Before computers, card catalogs offered some of this facility: there were often card catalogs organized by author, subject, and title. It would be unusual, however, to find a card catalog devoted to organizing items by publication date or date of acquisition.

Electronic browsing can overcome some of the limitations of the paper card catalogs. Electronic browsers can be much faster to use (as a patron can sit at a terminal, perhaps in his own office, rather than moving around the physical catalog). Browsers can provide greater flexibility in manipulating the items. For example, it is possible to search on both authors and years, and to present the items in a variety of ways (e.g., sorted by publication date).

3 SortTables: A System for Browsing

SortTables is an interface metaphor developed to support browsing. Information retrieval systems have often lacked a browser, thus forgoing the benefits of this type of interaction, or they have provided a simple electronic realization of the card catalog, making little use of the manipulative capabilities of the computer.

SortTables is not in any sense a complete solution to the problems of digital libraries. It only addresses the problem of presenting and manipulating a set of items; as presented here, its search facilities are limited. It doesn't address issues of how documents are stored or retrieved, protocols, security,

To appear in CIKM '95, The 4th International Conference on Information and Knowledge Management, Nov. 28-Dec. 2, 1995, Baltimore, Maryland. Draft: August 14, 1995.

bin/csh (tty1)

_test 98% of about 100000 items

Find:

TITLE	JOURNAL	YEAR	TYPE	AUTHOR	TERMS
Arabic, Persian a...	TUGBoat	1990	Journal Ar	Haralambous, ...	
Getting TeX nical...	TUGBoat	1990	Journal Ar	Hendrickson, ...	
Circular reasonin...	TUGBoat	1990	Journal Ar	Hoenig, Alan	
Just plain Q&A	TUGBoat	1990	Journal Ar	Hoenig, Alan	
A constructed D r...	TUGBoat	1990	Journal Ar	Hoenig, Alan	
TeXoutput devices...	TUGBoat	1990	Journal Ar	Hosek, Don	
Report from the ...	TUGBoat	1990	Journal Ar	Hosek, Don	
The IVRITEX Mail...	TUGBoat	1990	Journal Ar	Hosek, Don	
Lists in TeX's mo...	TUGBoat	1990	Journal Ar	Jeffrey, Ala...	
VMS site report	TUGBoat	1990	Journal Ar	Kellerman, D...	
Some macros to dr...	TUGBoat	1990	Journal Ar	Kelly, B. Ha...	
Virtual Fonts; Mo...	TUGBoat	1990	Journal Ar	Knuth, Donal...	
Exercises for TeX...	TUGBoat	1990	Journal Ar	Knuth, Donal...	
Arthur Lee Samuel...	TUGBoat	1990	Journal Ar	Knuth, Donal...	
Answers to Exerci...	TUGBoat	1990	Journal Ar	Knuth, Donal...	
The future of TeX...	TUGBoat	1990	Journal Ar	Knuth, Donal...	
Additional Hyphen...	TUGBoat	1990	Journal Ar	Kuiken, Gera...	
Textbook publishi...	TUGBoat	1990	Journal Ar	LaFrenz, Mim...	

Figure 1: The SortTables System

or the host of other issues that must be addressed by any full system.

In SortTables, all items are presented in a table, where each row corresponds to a record, and each column to an attribute of that record. At any point in time, one of the columns is marked as the sort key, and all rows are sorted according to it.

There are four essential functions that can be performed on a SortTables table: movement through the table, sorting the items according to one of the attributes, searching for a particular attribute value, and restricting the items according to a range of values of some attribute. The view is updated after each keystroke. The up- and down-arrows and page keys will move by line or page, the left- and right-arrows change the sort key column, alphanumeric characters search incrementally (as each character is typed), and a single keystroke can delete items not in a desired range.

Figure 1 shows the system running on Envision's data. (Envision is a database of computer science literature under construction at Virginia Tech [7] [8].) Note that the current line and the sort column are highlighted; items are sorted according to values in that highlighted column. (The highlighted column appears bold or dark gray on the screen; the '=' divider under the column title marks it as well.) If there were an active search string, it would appear after the text 'Find:'. There is a progress indicator showing approximately where the current line is relative to the list of items (here, 98% through a list of approximately 100K items).

With a few keystrokes, we could restrict our browsing to sets of items such as: "Journal articles only, authored by Knuth, sorted by year."

3.1 Example

Suppose we have the columns Title, Journal, Year, Type, Author, and Terms. To browse items for which "Journal

is 'IBM Journal of Research and Development' and Year ≥ 1980 , sorted by Author," we could proceed as follows. Initially, title is the first column. Press the right arrow key to sort by journal name, then type 'ibm j' to move to the entry for that journal. Typing '=' will restrict the table to those items that match the search string, with results shown in Figure 2.

Next we can move to the year column, and type '198<' to delete items before 1980, with results shown in Figure 3.

Pressing the right-arrow key twice will sort the remaining items by author, with results shown in Figure 4. Notice that we are seeing the same item (the one authored by Adams) in its neighborhood according to the new sort column.

So, with about a dozen keystrokes, we have made a moderately complex restriction, and can browse a useful subset of our data.

4 Design Goals and Early Implementation

The SortTables system was designed with a number of goals in mind:

- Simple metaphor with a high degree of interactivity
- Use of ranges and multiple orderings
- Ability to view the whole database
- Integrated browsing and searching
- Progressive utility
- Engender a sense of progress

The central metaphor is an automatically sorting table that can have parts deleted based on attribute values. The idea of reacting to each keystroke has been present from

bin/csh (tty1)

_test 0% of about 400 items

Find: ibm j

TITLE	JOURNAL	YEAR	TYPE	AUTHOR	TERMS
Multiconic surfac...	IBM J of Res...	1975	Journal Ar	Dimsdale, B....	
Convex cubic spli...	IBM J of Res...	1978	Journal Ar	Dimsdale, B.	
Trimmed surface a...	IBM J of Res...	1987	Journal Ar	Farouki, R.	
Cubic splines wit...	IBM J Resear...	1976	Journal Ar	Inselberg, A...	
Grin; Interactive...	IBM J. Res. ...	1981	Journal Ar	Fitzgerald, ...	I35 i
Average Complexit...	IBM J. Res. ...	1986	Journal Ar	Stone, Harol...	AI03
Color Display and...	IBM J. Res. ...	1983	Journal Ar	Farrell, E. ...	I3m D
Product quality l...	IBM J. Res. ...	1983			PERFO
Moisture solubili...	IBM J. Res. ...	1984			DESIG
Statistical failu...	IBM J. Res. ...	1984			PERFO
Parallel Solution...	IBM J. Res. ...	1974	Journal Ar	Kogge, P.	
Procedural Repres...	IBM J. Res. ...	1976	Journal Ar	Grossman, D....	
Fleshing Out Wire...	IBM J. Res. ...	1980	Journal Ar	Markowsky, G...	graph
Macro Generation ...	IBM J. Res. ...	1980	Journal Ar	Vergnieres, ...	Appli
A Geometric Model...	IBM J. Res. ...	1980	Journal Ar	Wesley, M. A...	
Voronoi Diagram f...	IBM J. Res. ...	1987	Journal Ar	Meshkat, S. ...	
Digital Image Pro...	IBM J. Res. ...	1976	Journal Ar	Bernstein, R...	
Automatic Scaling...	IBM J. Res. ...	1982	Journal Ar	Casey, R. G....	I31 p

Figure 2: Restricted to 'IBM J'

bin/csh (tty1)

_test 0% of about 300 items

Find: 198

TITLE	JOURNAL	YEAR	TYPE	AUTHOR	TERMS
Procedures for th...	IBM Journal ...	1980	Journal Ar	Adams, G. G.	ibm_r
Determining deadl...	IBM Journal ...	1980	Journal Ar	Ahuja, Vijay	ibm_r
The experimental ...	IBM Journal ...	1980	Journal Ar	Allen, F. E....	ibm_r
An overview of ma...	IBM Journal ...	1980	Journal Ar	Ames, I.	ibm_r
Josephson compute...	IBM Journal ...	1980	Journal Ar	Anacker, W.	ibm_r
Structure of tunn...	IBM Journal ...	1980	Journal Ar	Baker, John ...	ibm_r
Estimation of sta...	IBM Journal ...	1980	Journal Ar	Bard, Yonath...	ibm_r
Grammar character...	IBM Journal ...	1980	Journal Ar	Becerril, J....	ibm_r
Digital system fo...	IBM Journal ...	1980	Journal Ar	Beeteson, J....	ibm_r
A system solution...	IBM Journal ...	1980	Journal Ar	Bossen, D. C...	ibm_r
Optimization and ...	IBM Journal ...	1980	Journal Ar	Boyle, D.; M...	ibm_r
Pattern optimizat...	IBM Journal ...	1980	Journal Ar	Braunecker, ...	ibm_r
Effect of process...	IBM Journal ...	1980	Journal Ar	Broom, R. F....	ibm_r
Modeling of chara...	IBM Journal ...	1980	Journal Ar	Broom, R. F....	ibm_r
Fabrication and p...	IBM Journal ...	1980	Journal Ar	Broom, R. F....	ibm_r
An overview of Jo...	IBM Journal ...	1980	Journal Ar	Brown, Alan ...	ibm_r
On the complexity...	IBM Journal ...	1980	Journal Ar	Chung, K. M....	ibm_r
Strength reductio...	IBM Journal ...	1980	Journal Ar	Cocke, John;	ibm_r

Figure 3: Journal = 'ibm j' and Year \geq 1980

```

bin/csh (tty1)
Find: _test          5% of about 300 items
-----+-----+-----+-----+-----+-----
TITLE           : JOURNAL      : YEAR : TYPE      : AUTHOR          : TERMS
-----+-----+-----+-----+-----+-----
An introduction t...|IBM Journal  ...|1984 | Journal Ar|                | ibm_r
Fault-tolerant de...|IBM Journal  ...|1984 | Journal Ar|                | ibm_r
Introduction to t...|IBM Journal  ...|1984 | Journal Ar|                | ibm_r
Storing and evalu...|IBM Journal  ...|1986 | Journal Ar|                | ibm_r
Laser-enhanced pl...|IBM Journal  ...|1982 | Journal Ar|; Acosta, R....| ibm_r
Influence on LSI ...|IBM Journal  ...|1982 | Journal Ar|; Mikhail, W...| ibm_r
Timing analysis o...|IBM Journal  ...|1982 | Journal Ar|; Smith, Gor...| ibm_r
Analysis of linea...|IBM Journal  ...|1982 | Journal Ar|Abdou, Ikram...| ibm_r
Image processing ...|IBM Journal  ...|1985 | Journal Ar|Abrams, M.; ...| ibm_r
Optimizing preven...|IBM Journal  ...|1984 | Journal Ar|Adams, Edwar...| ibm_r
Procedures for th...|IBM Journal  ...|1980 | Journal Ar|Adams, G. G.   | ibm_r
The torus and the...|IBM Journal  ...|1987 | Journal Ar|Adler, R. L.   | ibm_r
Microprocessor im...|IBM Journal  ...|1982 | Journal Ar|Agnew, P. W....| ibm_r
Determining deadl...|IBM Journal  ...|1980 | Journal Ar|Ahuja, Vijay   | ibm_r
A program develop...|IBM Journal  ...|1984 | Journal Ar|Alberga, C. ...| ibm_r
The history of la...|IBM Journal  ...|1981 | Journal Ar|Allen, F. E.   | ibm_r
The experimental ...|IBM Journal  ...|1980 | Journal Ar|Allen, F. E....| ibm_r
An overview of ma...|IBM Journal  ...|1980 | Journal Ar|Ames, I.       | ibm_r

```

Figure 4: Journal = 'ibm j' and Year \geq 1980; sorted by Author

early on: rather than a command being typed, or a form being filled in while the system passively records keystrokes, this system is active while each key is typed.

An electronic replica of a card catalog might allow for range restriction on a single attribute. This metaphor extends that capability with something real card catalogs can't do: easily sort themselves based on *several* attributes, and allow restrictions based on them.

The ability to systematically view the whole database is a characteristic of many browsers. SortTables supports that by allowing one to page through all items based on any of the attributes.

Many systems have distinct modes for searching and for browsing: in search mode, a query retrieves a set of documents; in browse mode, entities (or attribute values) are listed. Some systems, such as the *Computing Archive* [1], integrate both modes, allowing a user to browse an author list while forming a query. The SortTables system is similar: the user is always browsing, but can form queries while doing so.

The system allows for progressive levels of utility: movement through the list, searching, sorting, and range restriction provide increasing levels of sophistication.

Finally, the system tries to engender a sense of progress in its use. This metaphor lets the user systematically restrict a multi-dimensional space by dealing with one dimension at a time. Actions either "look around" or they close in on a particular area of the data.

The system has gone through several versions, with two different concerns: exploring the interface, and developing the underlying implementation. Here is how the successive versions have addressed these issues:

1. Spreadsheet-based prototype. The first version was built on a spreadsheet, by adding buttons that would sort according to the various columns, and using native

spreadsheet facilities for searching and scrolling.

2. Graphical user interface on a NeXT. This version was a prototype designed using the NeXT Interface Builder, and was used to explore implementation techniques in a graphical environment.
3. VT100 interface—initial version. The first VT100-based version was developed to provide a stable interface portable across a variety of platforms. This version has been used as a testbed for formative evaluation of the interface.
4. VT100 interface—current version. The current version has incorporated a number of improvements suggested from formative evaluation, and has been used as a testbed for data structures capable of supporting large data sets.

5 Formative Evaluation

A formative evaluation of the first VT100-based version was performed, to improve the system before over-committing to a particular interface design. The evaluation took two forms: an interface designer (a Ph.D. student in the area of human-computer interaction) critiqued it, and two students served as users in a usability test. At the time, the system performed reasonably well with up to about five thousand items. (The worst case for sorting was less than ten seconds; other operations were faster). The results of this evaluation were used in developing the current version.

The interface critic suggested adding redundant visual coding to indicate the current sort column and a progress indicator showing about how many items were left. He helped tune some of the assignments of functions to keys. Finally, he encouraged future development of a graphical version.

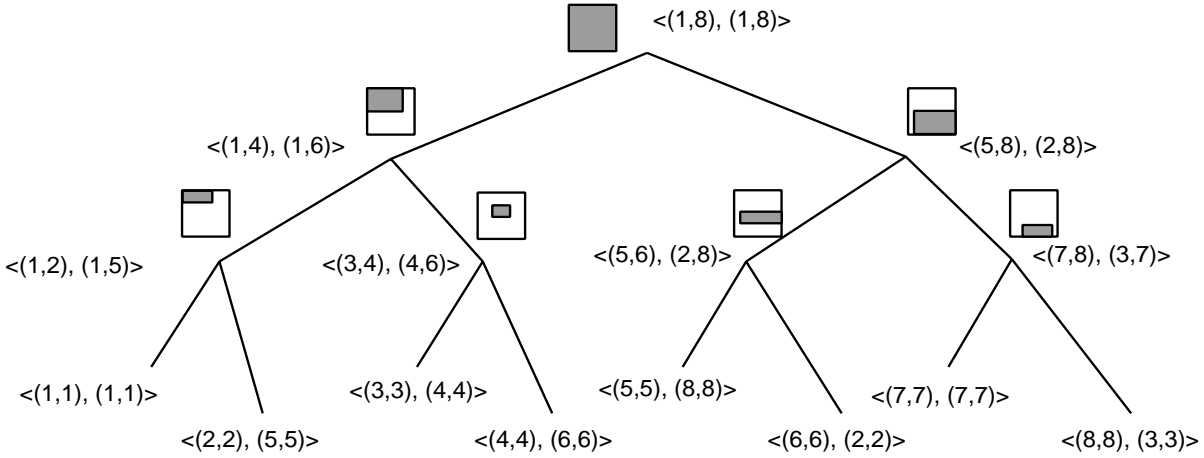


Figure 5: Bounds information tree for the first attribute

The students in the usability test worked with three different data sets: directory information resulting from the Unix `ls -l` listing command (25 items in 7 columns), data extracted from the CIA world fact database (250 items in 6 columns), and a collection of library catalog records (about 5000 items in 5 columns). These were selected to test the system on a variety of types of data sets.

The core of the interface was retained, but several changes were made to ease interaction. Page-up and -down keys were added, to improve navigation. Two functions were removed: undo, and the deletion of items not equal to the pattern. The training material was modified to put increased emphasis on the notion of the current sort column. Finally, sorting performance was identified as a potential bottleneck.

Currently, the system supports on the order of a million records, so the interface evaluation should be revisited with this larger data set.

6 Data Structure

To support the interface, a new data structure is being developed: the thread file. Its key idea is to use bounds information for subsets of the data to avoid searching many records.

6.1 The Basic Idea

The *bounds* of a set of records is a mapping from each attribute to its minimum and maximum values in that set. Suppose we have three records A, B, and C, with the values $\langle 3,4,5 \rangle$, $\langle 0,3,4 \rangle$, and $\langle 2,2,2 \rangle$ respectively. The bounds are: $\langle (0,3), (2,4), (2,5) \rangle$. This says that the first attribute ranges from 0 to 3, the second from 2 to 4, and the third from 2 to 5.

There is a complete binary tree of bounds information for each attribute. Figure 5 shows an example of one such tree (for the first attribute). For simplicity, the tree shows a data set with only two attributes. Nodes are labeled with the bounds information for each attribute, in the form: $\langle (low_1, high_1), (low_2, high_2) \rangle$. Figure 5 graphically shows how each parent's box is the bounding box of its two children. Notice also how successive levels of the tree split the box based on the first attribute.

There is a “known bad” bit associated with each node of the tree, set whenever the (sub-)tree is known not to contain any possibly useful children.

To evaluate a query, the tree for the desired ordering is used. A tree walk compares the nodes to the query:

- If the node is marked “known bad,” return failure.
- If the node is a record (i.e., a leaf) and its bounds are valid, return that record.
- If the bounds don't intersect the query, mark the node “known bad” and return failure.
- Otherwise, walk both children. If upon returning, both children are marked “known bad”, then mark this node “known bad” before returning failure.

The “known bad” bit is set along the way, so a node will not be evaluated again if it has already failed. The nature of the interface ensures that succeeding queries will be tighter than their predecessors, so bad nodes won't suddenly become useful again.

For example, suppose we use the tree to locate items according to the range restriction $\langle (4,8), (7,8) \rangle$. The top node intersects that range (of course), so we continue down the tree. Moving down, the node labeled $\langle (1,4), (1,6) \rangle$ does not intersect the query range, so we mark it “known bad” and ignore its children. On the other side of the tree, $\langle (5,6), (2,8) \rangle$ intersects the range, as do both its children. We will examine the leaf nodes: $\langle (5,5), (8,8) \rangle$ is good, $\langle (6,6), (2,2) \rangle$ is not, $\langle (7,7), (7,7) \rangle$ is good, and $\langle (8,8), (3,3) \rangle$ is not. The good nodes are returned as valid records, and the bad ones are marked “known bad.”

6.2 Refined Data Structure

As described, this structure makes great demands on main memory. To reduce these demands, the “known bad” bits are kept in memory, while the rest of the tree is demand-paged from disk. To reduce space, we want to avoid storing layers of the tree which provide us little information. For example, upper layers of the bounds tree aren't useful: they tend to enclose almost all of the data.

For each attribute, we keep a list of the records, in the order they appear when sorted by that attribute. For the

Bytes	Location	Type of data
$8K^2N/B$	memory	Slice of bounds tree
$KN/(8B)$	memory	Bound bits (“known bad”)
$4KN$	disk	Leaf layer of bounds tree
$KN/8$	memory	Record bits (“known bad”)
$4KN$	disk	Buckets

Table 1: Space required by the thread file. K is the number of columns, N the number of records, B is the number of records per bucket. Each index is assumed to take 4 bytes.

Data Set	Envision	Library
Records (N)	100,000	1,000,000
Columns (K)	6	5
Records/Bucket (B)	16	64
Record space (disk)	22M	167M
Thread file space (disk)	7M	47M
Memory required	2M	4M

Table 2: Actual space overhead for thread file (rounded to the nearest megabyte). “Memory required” does not include cache space for information on disk.

records A, B, and C above, the lists are: $\langle B,C,A \rangle$ (when sorted by the first attribute), $\langle C,B,A \rangle$ (the second), and $\langle A,B,C \rangle$ (the third). When there are no restrictions outside the current sort order, this enables straightforward display of the records in order.

These lists are divided into *buckets*, whose size is chosen to be convenient relative to the disk block size.

The current implementation maintains only three layers of the tree: the level for which bounds information corresponds to a disk block (64 records on our system), the level for buckets of B records, and the bounds information for individual records. The lower layers of the tree provide very fine-grained information, but they are so numerous that they are expensive to track. (The Envision data set has about 100K items; this means 100K position records, plus 10K bounds records per attribute. If the full tree were stored, it would require an additional 90K bounds records for each attribute.)

Table 1 shows the space requirements of this leaner data structure. Items whose location is “memory” reside permanently in main memory; “disk” items are loaded on demand and cached. Table 2 shows these requirements for the two large data sets we have loaded.

6.3 Limitations

This data structure is undergoing refinement and tuning. We plan to add a small auxiliary index in the style of Glimpse [9] to support regular expression queries.

The thread file data structure does not have uniform performance for all queries. When there are very few records left, there can come a point where it spends more time looking at all the records that don’t match the query than at finding those that do. For the Envision data set, this was not a problem, but for the library data set the delay became noticeable when there were fewer than about 5000 records. This problem is solved by switching representations when result sets become small.

7 Current Status

The system has been loaded with the two large data sets described in Table 2 above. The Envision data set consists of bibliographic data extracted from the Envision data base. The library data set consists of information extracted from the university’s online card catalog.

We have not formally assessed performance, but the following figures should give an idea of the interaction, for the system running on a DEC Alpha workstation with either database. Moving by line or page feels instantaneous (the system appears to spend all its time updating the screen). Changing the sort column happens almost as quickly. Searching by typing characters takes up to about two or three seconds. Finally, restricting items by range takes about five to fifteen seconds. These times should improve with further algorithm development and performance tuning, but they suffice to give the system a highly interactive feel.

8 Related Work

Chang and Rice [3] survey browsing from a variety of perspectives. They distinguish browsing from other types of information seeking, and develop a taxonomy of dimensions along which to analyze browsing: contextual, behavioral, motivational, cognitive, and resource-based. The work reported here fits best with what they call the library, information science, and information retrieval traditions: browsing for unplanned discovery and as a problem-solving technique.

Thompson and Croft [12] address the benefits and requirements of browsing, and point out that browsers need a rich set of links, a flexible user interface, and search capabilities. They have a graphical system in which document and concept neighborhoods can be browsed. Crouch et al. [5] discuss a similar system focused on browsing clusters of documents.

Development of the SortTables interface took place in the context of the MARIAN [6] and Envision [7] systems at Virginia Tech. MARIAN uses a vector model, and provides simple lists of ranked results, but lacks a browsing capability. Envision uses MARIAN’s search engine, but adds an unusual tool for viewing results: a “bubble view” of icons, laid out in a flexible two-dimensional arrangement. Envision is limited in browsing abilities: although the display is flexible, it only supports viewing at most a few hundred items from a previously made query. The SortTables interface simplifies Envision’s results display to a textual list, but provides flexible ordering, range restriction, and direct access to all the data.

The data structure to support the interface has two aspects: orthogonal range query, and sorting. (“Orthogonal range query” refers to queries of the form “ $\forall i : \min_i \leq val_i \leq \max_i$ ” as required by the interface.) Many researchers have proposed data structures to support such queries, e.g., k-d trees [2] [13], filtering search [4], and grid files [10]. Unfortunately, none of these address the requirement for sorting. The grid file uses a large multi-dimensional array to store buckets of data, indexed by a set of one-dimensional scales. The others are tree-based. Our data structure was directly inspired by the grid file, although it resembles the others in using trees.

9 Future Directions

Several aspects of this system are targeted for work over the next months:

- Improved algorithms and data structures. The system has reasonable performance on a DEC Alpha workstation, but the performance could be improved. It would be nice to have an algorithm that supported incremental additions to the database, and that allowed multiple simultaneous users in an efficient way. Several users could share trees, but they would need separate “bad” bits and other information.
- Other application areas. SortTables is well-suited for data sets that have a moderate number of interesting attributes. In addition to the bibliographic, geographic, and directory data sets mentioned above, we have explored using it with data from e-mail and calendar applications. These data sets share the characteristic that rapid sorting and range restriction based on attribute values provide useful manipulations.
- Networked interface. The current system is monolithic. It would be useful to split it into a networked client-server implementation, to allow broader use and to facilitate exploration of the thread file data structure with multiple concurrent users.
- Graphical interface. While the early prototypes used a graphical interface, later versions have evolved away from that. The graphical interface would have radio buttons as attribute labels (to choose the sort column) and a scroll bar for navigation. It would retain the single-window design, keeping the search field on the window with the browsing list. A Tcl/Tk version [11] is currently being designed.
- Evaluation. To assess the utility of this approach, it should be evaluated in real use. With the library card catalog data loaded, we hope this will make the system potentially useful to a broad class of users, so it can be effectively evaluated.

10 Acknowledgements

This work was supported in part by the National Science Foundation through CISE Institutional Infrastructure (Education) Grant CDA-9312611. It has benefited from the criticism and discussion of several colleagues, especially Dr. Kevin Mayo of SAIC.

References

- [1] ACM. *Computing Archive: Bibliography and Reviews from ACM*. New York, NY: ACM Press, 1991.
- [2] J. L. Bentley. “Multidimensional Binary Search Trees used for Associative Searching,” *Communications of the ACM*, 18(9), 1975, pp. 509-517.
- [3] Shan-Ju Chang and Ronald E. Rice. “Browsing: A Multidimensional Framework,” in Martha E. Williams, ed.: *Annual Review of Information Science and Technology (ARIST)*, Volume 28, 1993.
- [4] B. Chazelle. “Filtering Search: A New Approach to Query Answering,” *SIAM Journal on Computing*, 15(3), 1986, pp. 703-724.
- [5] Donald B. Crouch, Carolyn J. Crouch, and Glenn Andreas. “The Use of Cluster Hierarchies in Hypertext Information Retrieval,” in *Hypertext '89 Proceedings*, Pittsburgh, PA, ACM, 1989, pp. 225-237.
- [6] Edward A. Fox, Robert K. France, Eskinder Sahle, Amjad Daoud, and Ben E. Cline. “Development of a Modern OPAC: From REVTOLC to MARIAN,” *SIGIR '93: Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Pittsburgh, PA, ACM, 1993, pp. 248-259.
- [7] Edward A. Fox, Deborah Hix, Lucy T. Nowell, Dennis J. Brueni, William C. Wake, Lenwood S. Heath, and Durgesh Rao. “Users, User Interfaces, and Objects: Envision, a Digital Library,” *Journal of the American Society for Information Science*, 44(8), 1993, pp. 480-491.
- [8] L. Heath, D. Hix, L. Nowell, W. Wake, G. Averbach, and E. Fox. Envision: A User-Centered Database from the Computer Science Literature. *Communications of the ACM*, 38(4), Apr. 1995, pp. 52-53.
- [9] U. Manber and S. Wu. “GLIMPSE: A Tool to Search Through Entire File Systems,” Technical Report No. TR 93-34, University of Arizona Department of Computer Science, 1993.
- [10] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. “The Grid File: An Adaptive, Symmetric, Multi-Key File Structure,” *ACM Transactions on Database Systems*, 9(1), 1984, pp. 38-71.
- [11] John Ousterhout. *Tcl and the Tk Toolkit*, Reading, MA: Addison-Wesley, 1994.
- [12] R. H. Thompson and W. B. Croft. “Support for browsing in an intelligent text retrieval system,” *International Journal of Man-Machine Studies*, Volume 30, 1989, pp. 639-668.
- [13] D. E. Willard “New Data Structures for Orthogonal Range Queries,” *SIAM Journal on Computing*, 14(1), 1985, pp. 232-253.