Search++:
More Control than a Simple Search Interface without the Complexity and Confusion of Advanced Search

Alessandro Simone Agnello
Haim Levkowitz

Department of Computer Science
University of Massachusetts
Lowell, Massachusetts
01854-2874
alessandro_agnello@student.uml.edu
haim@cs.uml.edu

***Abstract*—On-line search engines are a key component of all on-line activity and have evolved through the years. Despite search engine advances, users today may need to execute multiple searches prior to reaching their desired search results. However, executing multiple searches in an effort to do so can be time consuming and may not lead to the best results. Leveraging complex search interfaces, including Boolean filters, can exacerbate the problem if the interface is too confusing and unfamiliar to the user. To help overcome these issues, we introduce Search++, a new search interface that reduces the steps for a common user to reach their desired search results, through the use of latest Web technologies and advanced visualizations.**

Keywords–Search Interface; User Decision Pattern; Priority Sorting; Web Visualization.

I. INTRODUCTION

Numerous studies have attempted to characterize the cognitive process of searching for information and model the related series of steps. A common theme amongst the models is a step or steps, where an individual will evaluate results of a search and use the information for a subsequent search to reach, or get closer to, the desired search results. These steps are referred to in many ways amongst the models: “Refinement” [1], “Formulation” [2], and “Query reformulation tactics” [3] to name a few. These models are applicable to searching for information on-line. They have been used to formulate new techniques for on-line search engine interfaces.

Search engine interfaces have evolved throughout the years starting with the first on-line search engine “Archie”, which included many Boolean filter options [4], to today's popular search engines, which include a single input box, button, and complex back-end algorithms to return results. Simplified search interfaces are preferred by common users as advanced search engines can be overwhelming to them. Despite advancements users find themselves executing multiple searches prior to honing in on the desired search results. This effort is time consuming and can lead users to feel discouraged and frustrated [2].

It has been found that a technique that includes summarized hierarchical display of data related to the search results along with the search results themselves can be effective at quickly reaching the final desired search results. However, success depends on how well the summarized data aligns with the search terms [5]. Site-specific search engines, such as Amazon, include Boolean filters with search results. However, the filters are pre-arranged based on the category (e.g., price filter may have a range from lowest to highest priced item within the result set). Furthermore, Boolean filters omit search results, which may further delay a user from honing in on the desired search results. Users have been known to enter broad search terms at first in an effort not to exclude search results that may help them further define the desired search results [6]. This suggests that a user's awareness of all available data helps her/him reach the desired search results.

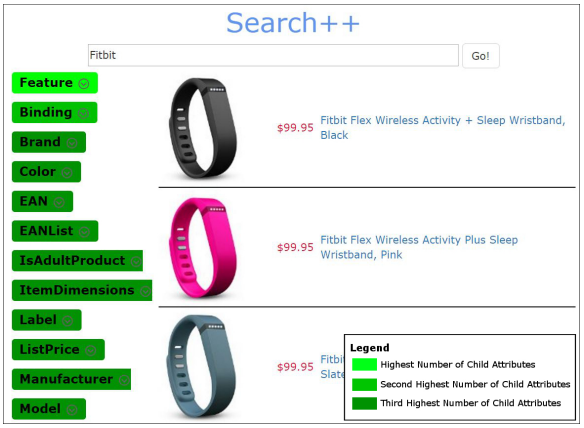

Figure 1. Subset of a result shown, when a user typed in Fitbit as a search term.

To help accommodate users, refining their search and to alleviate their struggling using advanced interfaces, we introduce a new technique, Search++. Search++ bridges the gap between the familiar but simplistic standard search interface and the complex advanced search interface. Search++ utilizes an expandable data source plugin utility that allows users to search for attributes of single or multiple source result sets. Search++ allows the user to rank these attributes, including their sub-attributes. The ranking informs Search++ what the user is more interested in (e.g., the user has a higher interest in the price of an item than in its weight). Colors are used to help depict which attributes have the largest number of sub-attributes. Search++ associates a base light green color with

Copyright (c) IARIA, 2016. ISBN: 978-1-61208-465-7

the attribute with the highest frequency of sub-attributes. Light green was chosen to show contrast against the background and green creates a relaxing effect for people [7]. We felt that attributes with the highest frequency of sub-attributes, shows the most likely sorted set for the common user. These attributes are most common throughout the returned result set. Attributes are darkened in color proportional to their decrease of sub-attribute frequency. (E.g., Features: 20 sub-attributes, Price: 10 sub-attributes, thus Features is light green, and Price is darker). This allows the user to quickly visualize the attributes with greatest sub-attribute frequency. In Figure 1, we provide an example result set of Search++. Upon initial data retrieval or user interaction our query processing modules will be triggered. These modules retrieve, normalize, sort, and visualize user's search input and interaction (discussed in Section IV-B). The main contributions of Search++ are:

- A new search interface that provides more control than the standard stripped down interface. Without overwhelming the user with advanced search interface components.
- User defined and controlled relevance ranking.
- A framework that aggregates multiple Web Server/Client technologies.
- An interactive and visual interface that lets the user control the process and view results.

To help evaluate the capabilities of Search++, we constructed a case study using Amazon Web Services (AWS) [8]. We asked a group of ten individuals who described themselves as "non-technical" to search for six unique products they are seeking to purchase; three searches using Amazon's search interface and three using Search++. Section VI describes this in further detail.

In Section II, we discuss the past and present implementations of search interfaces. Section III define the components of Search++. Section IV describes how the various components work among each other and user interaction. Section V define the processing modules upon initial load and user interaction results. Section VI shows our case study using AWS as our data repository. Section VII recaps our findings of Search++.

II. Related Work

The goal of any search engine is to find information that matches or is relevant to some criteria [9]. In the early days of on-line computing, it was common to have complex input criteria that may not be readable/understandable to today's common user.

Complex interfaces are only usable by a few very technically savvy users who are able to negotiate the interface's complexity to their needs. They often cause confusion and frustration for the common user. Various approaches, such as WISE-Integrator [10], try to bridge this gap by giving the user more fields to fill out (e.g., price range, author name, etc.), but this assumes the user knows more about what they are looking for, which is often not the case. Common e-commerce based search engines [11] have filter-down mechanisms to help users identify some of their needs (e.g., price, brand, etc.). However, the user cannot explicitly state what is their highest relevance. These filtering mechanisms intentions are to remove entries rather than resort them. Complex, advanced search engine interfaces [12] [13] are not designed for the common user. They show powerful results, but they lack in ease of control and instant understanding/intentions of use.

Our Search++ approach is designed to support a common (i.e., not particularly technical) user, offering a little more control than just an input field, without overwhelming the user with a tremendous amount of additional fields that may or may not be coherent. We define a dynamic attribute as a parent level descriptor of a given item. A dynamic attribute instance may have child attributes. Child attributes are child level descriptors of a single dynamic attribute. It is possible that a child attribute may also be a dynamic attribute and have child attributes. Our case study (Section VI) does not show any child attributes also being a dynamic attribute. Dynamic attribute instances are obtained from the search result set and grouped in a way that shows only distinct results. Our case study (Section VI) demonstrates various forms of this scenario. We color code each attribute to show frequency of sub-attributes (discussed more in Section V-D). Additionally, the user can sort these attributes based on her/his priorities. Upon resorting attributes, Search++ will display results based upon their attribute ranking.

In Section VI, we demonstrate an application of Search++ on data exploration of e-commerce products. We explore the usage of ranking of dynamic and child attributes along with color scheme. We demonstrate the usefulness of Search++, and display the value of ranked attribute and sub-attribute results.

III. Components

The current fundamental set of Search++ components includes three base views (Dynamic Table, Input Box, and Dynamic Attributes), Session State Management, Data Source Plugins, and four Query Processing Modules (Frequency Based Priorities, Resorting Priority, Selection, and Color Code). Additional libraries provide data organization and presentation. We now describe each component in more detail.

A. Dynamic Table

Search++'s dynamic table provides a spreadsheet-like view of the data. Each table may have up to i rows and j columns. A user may select n row(s) to view the dynamic and child attributes related to the selection. Upon selecting a row, it will be highlighted. The dynamic and child attributes related to selected row(s) will be displayed. Upon deselection of the row(s), the entire list of dynamic and child attributes are displayed. This technique provides an on-demand multiview linked visualization that helps the user learn what dynamic and child attributes make up the related dynamic table entries. Figure 2 show the various states of a dynamic table.

B. Input Box

Search++'s input box is the initial visualization and first point of interaction for the user. This component is common in most search interfaces.

C. Dynamic Attributes

Search++'s dynamic attributes are created from data associated with each of the search results. A dynamic attribute may have n child attribute(s) (e.g., "Feature" being a dynamic attribute having child attributes of: "16 Megapixels", "5x

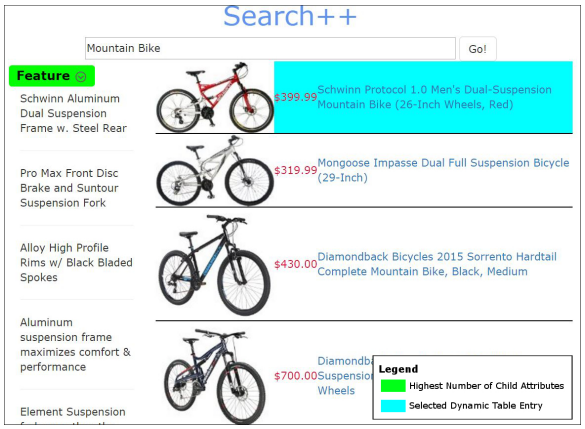

Figure 2. User has clicked on a single dynamic table entry. Only the associated dynamic and child attributes are displayed.

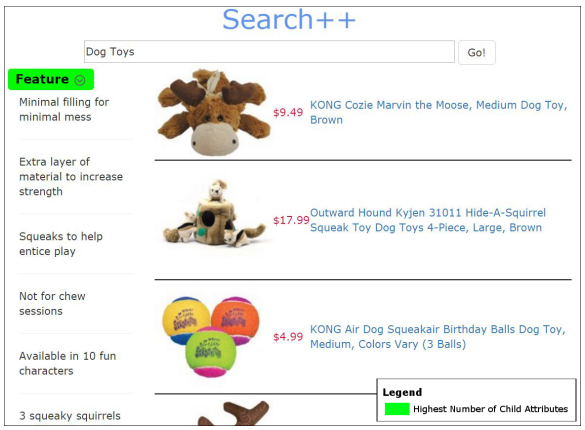

Figure 3. User has expanded dynamic attribute “Feature” and associated child attributes are now displayed.

optical zoom”, “20x digital zoom”, etc.). Figure 3 shows various child attributes.

Search++'s interface allows users to sort dynamic and child attributes according to their priority. Upon sorting, our query processing modules will be engaged (discussed in Section III-F). The dynamic attributes are color coded by their frequency of child attributes (discussed in Section V-D). This allows the user to learn how common an attribute is. Dynamic and child attributes may be hidden when the user is interacting with the dynamic table row(s). This technique allows users to view the dynamic and child attributes related only to their selection.

D. Session State Management

There are three common techniques to store session state: client, server, database [14]. Search++ utilizes server based session state. Each user has one session, which are not shared between users. Sessions allow the server to automatically garbage collect un-utilized data [15]. Session state management in Search++ allows the system to retain critical information that was normalized and processed upon the initial and subsequent execution of the search. This information is referenced upon UI interaction within the same result set. This technique eliminates the need for re-access to the data source.

E. Data Source Plugins

We define a data source plugin as a utility to retrieve and normalize external data requests. Traditional and pure plugin architectures [16] were examined. We chose traditional plugin architecture as it allowed us to retrieve data with less complexity compared than pure. Search++'s data source plugins allows for various API's to be called and normalized to a single result set. This technique allows Search++ to add and configure plugins without impacting the downstream processes. In Section VI, we demonstrate the usefulness of this technique.

F. Query Processing Modules

Search++ has four primary query processing modules (frequency based priority, resorting priority, selection, and color coding). Each module is responsible for a single task (e.g., retrieve data, organize data). This technique allows other modules to be added or removed during run-time without affecting the entire system. Each module is discussed in detail in Section V.

G. Implementation

Search++ is implemented using commonly-used libraries (Microsoft MVC5, Angular-JS, Bootstrap and jQuery-Sortable [17], [18], [19], [20]) for user interactions, design of each component, session management and algorithm runtime. The combination of these technologies is common and has been shown to provide the necessary performance capability when used in conjunction.

IV. UI, INTERACTION, AND WORKFLOW

Search++'s interface displays a single view. This view allows a user to search for criteria and rank attributes according to their priority. Search++'s query processing modules will leverage the attribute priority to obtain a list of prioritized search results. In this section, we describe the workflow that a user needs to carry out in Search++ to accomplish a goal.

A. Initial Search

Upon initial search, Search++ retrieves data to render the visualization. Upon retrieval of data from our data source plugins (Section III-E) Search++'s query processing modules will engage to normalize the data set for visualization. The result set includes dynamic attributes and tables. The dynamic attributes are presented in the default sort order in accordance to the dynamic and child attribute frequency (described in Section V-A). The dynamic table will include the sorted search results based upon the dynamic and child attribute priority (described in Section V-C).

The following is the exchange that takes place between client and server for initial data acquisition.

- Client : HTTP GET request for data
- Server : Receives request
  - Queries data within plugin architecture
  - Performs normalization and engages algorithms
- Client : Receives Data

Copyright (c) IARIA, 2016. ISBN: 978-1-61208-465-7

B. Manipulating Data

Manipulating details within a data set to further derive insight is important. Search++ allows for manipulations within dynamic and child attributes, dynamic table, or re-searching for criteria. Upon any of these manipulations our dynamic table will update (discussed in Section IV-B2). In the following subsections we will discuss each manipulation and the reactions that occur.

1) Dynamic and Child Attributes: A user can manipulate dynamic and child attributes. A dynamic attribute can be reordered amongst other dynamic attributes. A child attribute can be reordered among other child attributes within the particular dynamic attribute. A user can reorder a dynamic or child attribute by selecting and dragging the attribute above or below other attributes. Upon reordering our query processing module is engaged and the dynamic table is updated. Figure 4 demonstrates the result of a user reordering.

2) Dynamic Table: Search++ has a simplistic view for the user to quickly understand what is available. The dynamic table entries may be reordered depending on the user's interaction. If the user changes their search criteria the dynamic table entries will be replaced. To help users understand what dynamic and child attributes are associated with a particular row in the dynamic table, Search++ allows the user to select n row(s). Upon selection of a given row, Search++ displays the dynamic and child attributes related to the selection. If a user deselects all rows previously selected, the dynamic and child attributes related to the entire result set will reappear. Figure 2 demonstrates this action. This creates a multiview linked visualization that helps the user learn what attributes comprise a particular set of entries.

3) New Search: On a new search, Search++ will reinitialize all data to the begin state. Search++'s visualization will update the result set to a similar format of all previous searches. This consistency allows users to be more effective conducting searches as the components become familiar to the user.

V. QUERY PROCESSING MODULES

Search++ has four query processing modules that work together to provide a normalized result set. We will discuss each module in further detail below.

A. Frequency Based Priority

The frequency based priority module is run when the user conducts a new search. The input to this algorithm is the output dynamic and child attributes from the originating data source. The output is a sorted occurrence list of dynamic and child attributes (i.e., attributes that are most common are earlier in the list and tail down to least). Table I shows an example input and output. This algorithm sorts dynamic attributes in descending order according to the dynamic attributes with the greatest number of distinct child attributes. This allows the user to quickly ascertain the attributes with the greatest amount of variation.

B. Resorting Priority

Resorting dynamic and child attributes allows the user to identify search results that most closely align with their ranked ordered priorities. This technique is preferable to that of complex interfaces that filter criteria as result sets are not removed but reordered.

TABLE I. THE FOLLOWING TABLE SHOWS AN EXAMPLE INPUT AND EXPECTED OUTPUT FROM THE FREQUENCY BASED PRIORITY ALGORITHM.

Input	*Output*
Fragile (5)	Feature (30)
Weight (10)	Weight (10)
Feature (30)	Warranty (8)
Warranty (8)	Fragile (5)

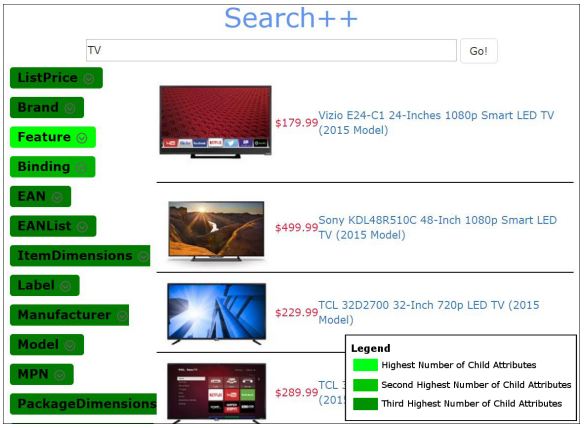

Figure 4. User has prioritized “ListPrice” and “Brand” above “Feature”.

Upon sort of dynamic or child attributes, the Web client will notify the server of change. Search++ has an optimal approach of sending this data, by sending only the unique identifiers of the new order, rather than the organized full data set. With the use of session state management (discussed in Section III-D) we are allowed to quickly manipulate the resorting and allow our selection algorithm to engage.

C. Selection

Search engines typically provide results in a table. Common users have grown accustomed to such an interface. Search++ additionally displays the results as such referred to as a dynamic table (described in Section IV-B2). The selection algorithm is designed to create entries in a table manner. The priority sort order of dynamic and child attributes define the sort order of the dynamic table entries. The selection algorithm maps the dynamic and child attributes to the data source entries to provide the dynamic table.

The selection algorithm first identifies the search results with parent and child attribute pair ranked highest in the priority. These search results are designated to be first on the list in the dynamic table. The selection algorithm will then identify search results that were not used in the first iteration and contain the parent and child attribute pair ranked second in priority. These search results are designated to be second on the list of the dynamic table. This process is repeated until all search results have been identified and designated to be included in the dynamic table. This algorithm will be triggered upon new search creation or user interaction of dynamic or child attributes.

D. Color Code

We color code dynamic attributes with the highest frequency in light green and darken the color of each subsequent

Copyright (c) IARIA, 2016. ISBN: 978-1-61208-465-7

dynamic attribute with lower frequency than the previous. We felt that attributes with the highest frequency of sub-attributes, shows the most likely sorted set for the common user. If two or more dynamic attributes have the same number of child attributes, their color will be identical. We only color these dynamic and child attributes once, upon initial data retrieval.

Each dynamic attribute following the base attribute is shaded using the following calculation: previous dynamic attribute color + (((current dynamic attribute child count - previous dynamic attribute child count) / previous dynamic attribute child count) * base light color).

VI. CASE STUDY

To demonstrate Search++'s functionality and utility, we conducted a user case study. In order to facilitate the case study, a data source plugin for Search++ was developed to leverage AWS as a repository. We asked a group of ten individuals who described themselves as "non-technical", five male, five female, aging from 25 – 62, having highest education degree: graduate (2), undergraduate (4), and high school diploma (4) to search for six unique products they are seeking to purchase; three searches using Amazon's search interface and three using Search++. We compared the number of steps taken by participants to find the desired search result, level of confidence participants had with the search result, and overall experience using each (Amazon, Search++). In the following sections we describe the results and conclusions of this case study.

A. Results

We captured data from ten participants and sixty search attempts; thirty using Amazon's search interface and thirty using Search++. We counted the number of steps each participant took in each search attempt to find the desired search result. A step is defined as any and each manipulation that a participant performs on the view. On average we found each participant took 7.8 steps using Amazon's search interface compared to 7.3 steps using Search++ to find the desired search result. Table II shows summarized results of participant steps. Table IV shows summarized results of Amazon's search interface. Table V shows summarized results of Search++.

In general, we found participants using Search++ required fewer steps compared to Amazon's search interface to find the desired search target. Search++ required fewer steps except for filtering/re-ranking. We believe participants in this study re-ranked items more than necessary as they were experiencing the Search++ technique for the first time. In the next section we will discuss user feedback of this case study.

TABLE II. THE FOLLOWING TABLE SHOWS SUMMARIZED RESULTS OF PARTICIPANT STEPS

	Amazon	*Search++*
Average Number of Steps	7.8	7.37
Median	7	7
Mode	6	7
Standard Deviation	2.98	1.45

At the conclusion of each participant's search, we asked the participant: "How confident do you feel that your selected search result is best result that could be found? Rate 1–5 (1 being the lowest level of confidence and 5 being the highest)". Participant confidence with the selected search result was found to be higher using Search++. Table III shows the summarized results of participant confidence with the selected search result. We believe the Amazon's search interface filters, which omit results from the result set, led participants to have a low level of confidence. In contrast, Search++ re-prioritization technique does not omit results, but re-orders the result set.

TABLE III. THE FOLLOWING TABLE SHOWS USERS SUMMARIZED CONFIDENCE LEVEL

	Amazon	*Search++*
Average Confidence Level	3.53	4.37
Median	4	4
Mode	4	4
Standard Deviation	0.73	0.49

TABLE IV. THE FOLLOWING TABLE SHOWS SUMMARIZED RESULTS OF PARTICIPANT STEPS USING AMAZON'S SEARCH INTERFACE

	Back Button	*New Search Term*	*Filtering*	*Product Detail View*	*Next Page*
Average	0.6	1.37	3.33	1.6	1.33
Median	0.0	1.0	3.0	1.0	1.0
Mode	0.0	1.0	3.0	1.0	1.0
Standard Deviation	0.81	0.56	0.48	0.81	0.55
Count	18	41	100	48	40

TABLE V. THE FOLLOWING TABLE SHOWS SUMMARIZED RESULTS OF PARTICIPANT STEPS USING SEARCH++

	Back Button	*New Search Term*	*Rank Change*	*Product Detail View*	*Next Page*
Average	0.27	1.2	3.93	1.27	1.33
Median	0.0	1.0	4.0	1.0	1.0
Mode	0.0	1.0	4.0	1.0	1.0
Standard Deviation	0.45	0.41	0.64	0.45	0.55
Count	8	36	118	38	40

In addition to the confidence rating, we asked participants: "What was your experience using Search++" and "Would you like to use the Search++ technique in other areas of search on-line?" Eight out of the ten participants responded favorably to their experience using Search++.

Nine out of the ten participants would like to see Search++ in other areas of search on-line. "I like how I can just continuously re-shuffle my results without having to search over and over again."

One more observation: We collected all dynamic and child attributes that were used to present the returned Search++ fields. Table VI shows a summary of the result set. Note that the selection pool of dynamic and child attributes listed in the table provides plenty data to priority-sort search terms. This seems to offer more power to the user than the prevailing filtering approaches, which tend to remove potentially relevant data from the analysis.

B. Conclusion

The case study yielded favorable results for Search++, showing that it is easily adopted by users, can reduce the number of steps to obtain search results, and might be preferred over existing search interfaces. We believe that participants that did not respond favorably to Search++ may have factored

Copyright (c) IARIA, 2016. ISBN: 978-1-61208-465-7

TABLE VI. THE FOLLOWING TABLE SHOWS DYNAMIC AND CHILD ATTRIBUTES RETREIVIED DURING SEARCH++ SEARCHES

	Dynamic	*Child Attributes*
Average	32.8	289.12
Mode	30	288
Min	25	233
Max	51	357
Standard Deviation	5.64	32.37
Total	820	7228

Amazon's other offerings above and beyond search results when making the comparison (e.g., "Customer Reviews", "Customers also bought"). The technique allows users to find results that they may not have known existed using other search interfaces. "I found a school supply package using Search++ including the crayons I was looking for as well as other markers and paper which I assumed I'd have to buy separate, I didn't see this on Amazon's search."

Search++ yielded only slightly better results in terms of steps to find desired search results; 7.3 steps on average compared to that of 7.8 using Amazon however, Search++ rated hire in participant confidence. However, it is worth considering that subjects had had more experience searching with Amazon's search interface than with Search++; it is reasonable to expect that with a little more experience, their performance and satisfaction using Search++ would improve.

VII. CONCLUSIONS

It has been shown that better search tools are needed for better exploration [21]. Oblinger and Oblinger [22] argue that the "Net generation" (those who learned to read after the Web) are qualitatively different in their informational behaviors and expectations; they multitask and expect their informational resources to be electronic and dynamic. The Net generation expects to be able to use Web resources to lookup, learn, and investigate tasks with fluid user interfaces. To the best of our research, even now, ten years after that observation was made, search tools still cannot meet those expectations.

In this paper, we have introduced Search++, a technique that allows users to easily execute a search against one or many sources, identify the attributes that may be most important to them, prioritize attributes according to importance, and find the search results that most closely align with their priorities. We have demonstrated that additional search options are needed. We have shown various ways to explore these integrated results within the interactive visualization paradigm, by the three UI components described in Section III (Dynamic Table, Input Box, and Dynamic Attributes). We have provided methods to efficiently organize these results without interfering with cognitive workflow.

We provided a case study that explored the value in Search++, by asking ten "non-technical" individuals to search for six unique products they are seeking to purchase; three searches using Amazon's search interface and three using Search++. We found that Search++ offered a slight improvement over Amazon's search interface. Our research was limited with our case study of ten users. We plan on advancing Search++ by introducing a collaboration feature, which will allow multiple searchers to work towards a a common goal. Search++ may be adapted to solve other decision based real world problems, by taking into considerations features inherent to each problem. (For example, consider the problem of seeking a new job, and imagine the various job-related features that might impact the search.)

REFERENCES

[1] B. Shneiderman, D. Byrd, and W. B. Croft, "Clarifying search: A user-interface framework for text searches," D-lib magazine, vol. 3, no. 1, 1997, pp. 18–20.

[2] C. C. Kuhlthau, "Inside the search process: Information seeking from the user's perspective," JASIS, vol. 42, no. 5, 1991, pp. 361–371.

[3] M. J. Bates, "Information search tactics," Journal of the American Society for information Science, vol. 30, no. 4, 1979, pp. 205–214.

[4] A. Halavais, Search engine society. John Wiley & Sons, 2013.

[5] M. W. Newman and J. A. Landay, "Sitemaps, storyboards, and specifications: a sketch of web site design practice," in Proceedings of the 3rd conference on Designing interactive systems: processes, practices, methods, and techniques. ACM, 2000, pp. 263–274.

[6] J. Teevan, C. Alvarado, M. S. Ackerman, and D. R. Karger, "The perfect search engine is not enough: a study of orienteering behavior in directed search," in Proceedings of the SIGCHI conference on Human factors in computing systems. ACM, 2004, pp. 415–422.

[7] K. Naz and H. Helen, "Color-emotion associations: Past experience and personal preference," in AIC 2004 Color and Paints, Interim Meeting of the International Color Association, Proceedings, vol. 5. Jose Luis Caivano, 2004, p. 31.

[8] E. Amazon, "Amazon elastic compute cloud (amazon ec2)," Amazon Elastic Compute Cloud (Amazon EC2), 2010.

[9] J. Y. Kim, M. Cramer, J. Teevan, and D. Lagun, "Understanding how people interact with web search results that change in real-time using implicit feedback," in Proceedings of the 22nd ACM international conference on Conference on information & knowledge management. ACM, 2013, pp. 2321–2326.

[10] H. He, W. Meng, C. Yu, and Z. Wu, "Wise-integrator: An automatic integrator of web search interfaces for e-commerce," in Proceedings of the 29th international conference on Very large data bases-Volume 29. VLDB Endowment, 2003, pp. 357–368.

[11] Q. Peng, W. Meng, H. He, and C. Yu, "Clustering e-commerce search engines," in Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters. ACM, 2004, pp. 416–417.

[12] M. Wilson, A. Russell, D. A. Smith et al., "mspace: improving information access to multimedia domains with multimodal exploratory search," Communications of the ACM, vol. 49, no. 4, 2006, pp. 47–49.

[13] J. Zhang and G. Marchionini, "Evaluation and evolution of a browse and search interface: Relation browser++," in Proceedings of the 2005 national conference on Digital government research. Digital Government Society of North America, 2005, pp. 179–188.

[14] M. Fowler, Patterns of enterprise application architecture. Addison-Wesley Longman Publishing Co., Inc., 2002.

[15] B. C. Ling, E. Kiciman, and A. Fox, "Session state: Beyond soft state." in NSDI, vol. 4, 2004, pp. 22–22.

[16] D. Birsan, "On plug-ins and extensible architectures," Queue, vol. 3, no. 2, 2005, pp. 40–46.

[17] A. Leff and J. T. Rayfield, "Web-application development using the model/view/controller design pattern," in Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International. IEEE, 2001, pp. 118–127.

[18] N. Jain, P. Mangal, and D. Mehta, "Angularjs: A modern mvc framework in javascript," Journal of Global Research in Computer Science, vol. 5, no. 12, 2015, pp. 17–23.

[19] M. Otto and J. Thornton, "Bootstrap," Twitter Bootstrap, 2013.

[20] B. Bibeault and Y. Kats, jQuery in Action. Dreamtech Press, 2008.

[21] M. Johnson, I. Zaretskaya, Y. Raytselis, Y. Merezhuk, S. McGinnis, and T. L. Madden, "Ncbi blast: a better web interface," Nucleic acids research, vol. 36, no. suppl 2, 2008, pp. W5–W9.

[22] D. Oblinger, J. L. Oblinger, and J. K. Lippincott, Educating the net generation. Boulder, Colo.: EDUCAUSE, c2005. 1 v.(various pagings): illustrations., 2005.

Copyright (c) IARIA, 2016. ISBN: 978-1-61208-465-7