

User-Centric Organization of Search Results

Search engines should organize results to minimize user effort. The authors introduce a user-centric approach to organizing search results for the common ranked-list search interface and the alternative clustering interface, letting users personalize how the results are organized. Such personalized views can be combined to provide an aggregated view as a mass-collaborative way of improving search performance. Two working prototypes, Rants and ClusteringWiki, show how the approach can serve as a complementary solution for effectively organizing search results.

Byron J. Gao
Texas State University

David Buttler
Lawrence Livermore National Lab

**David C. Anastasiu,
Shuaiqiang Wang,
Peng Zhang, and Joey Jan**
Texas State University

How search results are organized and presented directly affects search engine utility. Ideally, the organization should make it easier for users to fulfill their information needs. Currently, the most common search interface is the list interface, in which search results are organized in a ranked list. The clustering interface is an emerging alternative that lets users navigate search results through a hierarchy of meaningful labels (see the “List Versus Clustering” sidebar).

Most search engines use a machine-centric approach for organizing search results. It’s automatic, algorithmic, and involves little or no user intervention. For the list interface, the ranking scheme considers relevancy, term proximity, static quality, and diversity. Commercial search engines typically adopt numerous ranking heuristics. Machine learning techniques are also under intensive research and development. Algorithmic search result clustering

has received increasing attention in recent years from the information retrieval, Web search, and data mining communities (see the “Related Work in Search Result Organization” sidebar). Commercial clustering search engines include Clusty (www.clusty.com), iBoogie (www.iboogie.com), and CarrotSearch (carrotsearch.com).

Our user-centric approach lets users directly edit and manipulate the search result organization, thus creating a query-specific personalized view. Traditional personalized search maintains a profile for each user based on the user’s search history¹ and uses this profile to influence all queries from the user. This influence is limited and indirect because the user can’t arbitrarily and directly manipulate the search result presentation. In user-centric personalization, a set of edits is associated with a particular query (or similar ones). Web queries abide by the power law,² which means there are many rare queries and very

List Versus Clustering

The Boolean model is an early information retrieval (IR) paradigm in which each query represents a Boolean expression. It is a 1–0 model in which documents in the corpus are either relevant – that is, they satisfy the Boolean condition the query specifies – or irrelevant. All the relevant documents are returned unranked.

Most modern IR systems adopt the more user-friendly vector space model. Under this model, search results are presented as a ranked list according to their relevancy to the query, where document-query relevancy is calculated based on the similarity between the document and the query, both represented as vectors of term weights. Most Web search engines present search results as a ranked list, where the ranking scheme typically incorporates additional factors, such as static quality (for example, PageRank).

The flat ranked list search interface is simple and intuitive. It's effective for homogeneous search results or navigational queries that seek a single website or a single entity's homepage. However, queries are inherently ambiguous, and search results are often diverse with multiple senses. In a list presentation, the results on a query's different subtopics will be mixed together. The user must sift through many irrelevant results to locate the relevant ones.

With the Web's rapid growth, queries have become more ambiguous than ever. For example, Wikipedia includes more

than 20 entries for individuals named Jim Gray, including a computer scientist, a sportscaster, a zoologist, a politician, a film director, and a cricketer.

Clustering is an alternative search interface that minimizes users' browsing effort and alleviates information overload by providing additional structure.^{1–3} Clustering organizes objects into groups (*clusters*) that exhibit internal cohesion and external isolation. We can use clustering to categorize a long list of disparate search results into a few clusters such that each cluster represents a homogeneous query subtopic. Meaningfully labeled, these clusters let the user quickly locate relevant and interesting results. Evidence shows that clustering improves the user experience and search result quality.⁴

References

1. M.A. Hearst and J.O. Pedersen, "Reexamining the Cluster Hypothesis: Scatter/Gather on Retrieval Results," *Proc. 19th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, ACM, 1996, pp. 76–84.
2. O. Zamir and O. Etzioni, "Web Document Clustering: A Feasibility Demonstration," *Proc. 21st Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, ACM, 1998, pp. 46–54.
3. C. Carpineto et al., "A Survey of Web Clustering Engines," *ACM Computing Surveys*, vol. 41, no. 3, 2009, pp. 1–38.
4. C.D. Marming, P. Raghavan, and H. Schtze, *Introduction to Information Retrieval*, Cambridge Univ. Press, 2008.

few frequent ones. Queries from individual users would follow the same trend. Therefore, although the user-centric approach is query-specific, in practice it can be applied wisely only to the common queries and wouldn't incur unlimited storage and computation overhead. It provides a complementary solution to profile-based personalization, and the two can be used in combination to handle common and rare queries.

User-centric personalization also lets users share edits. This approach can combine personalized views to form an aggregated view, letting users collaborate at a mass scale and "vote" for the best search result presentation. This approach is in line with current Web trends, such as Web 2.0, Semantic Web, and mass collaboration.³ Whereas general mass collaboration applications feature entity annotation, our approach involves implicitly annotating relationships. Specifically, user editing of ranked lists corresponds to annotation of total orders, and user editing of clusterings corresponds to annotation of partial orders.

We've deployed two prototypes, Rants (dmlab.cs.txstate.edu/rants) and ClusteringWiki

(dm lab.cs.txstate.edu/ClusteringWiki), for the list and clustering interfaces, respectively. Rants (Figure 1a) features enhanced functionalities and makes the first effort to establish a framework and principled solutions for search engines of this kind. For the clustering interface, we introduce and implement personalized and mass-collaborative clustering in the context of search result organization. In contrast to existing approaches that innovate on the automatic algorithmic clustering procedure, ClusteringWiki (Figure 1b) lets users directly edit the clustering results.

Designing and implementing Rants and ClusteringWiki pose nontrivial technical challenges. User edits represent user preferences or constraints that should be enforced the next time the same query is issued. Query processing is time-critical, thus efficiency must receive high priority to maintain and enforce user preferences. Moreover, the dynamic nature of search results brings complications.

Architecture and Principles

Rants and ClusteringWiki share a common architecture (shown in Figure 2) and principles.

Related Work in Search Result Organization

Through SearchWiki (googleblog.blogspot.com/2008/11/searchwiki-make-search-your-own.html) and U Rank (research.microsoft.com/en-us/projects/urank), Web search giants Google and Microsoft have experimented with a new search paradigm that lets users directly and arbitrarily edit the search result ranking. Rants demonstrates a well-formulated framework with extended functionalities,¹ letting users specify both relative and absolute preferences and providing enhanced flexibility in aggregating and sharing user preferences.

Researchers initially proposed document clustering for information retrieval and Web search to improve search performance by validating the cluster hypothesis, which states that documents in the same cluster behave similarly with respect to relevance to information needs.² In recent years, researchers have used clustering to organize search results, creating a cluster-based search interface as an alternative presentation to the list interface. The list interface works fine for most navigational queries (seeking a single website), but can be less effective for informational queries (covering a broad topic), which account for most Web queries.^{3,4} Research has shown that the cluster interface improves user experience and search result quality.⁵⁻⁷

One way to create a cluster interface is to construct a static, offline, preretrieval clustering of the entire document collection. For example, the dmoz (www.dmoz.org) directory tries to establish a hierarchical categorization for the Web. It was manually created by 52,000 editors and covers less than 5 percent of all websites. However, this approach is ineffective because it's based on features that appear frequently in the entire collection but are irrelevant to the particular query.⁸ Query-specific, online, postretrieval clustering (that is, clustering search results) produces superior results.⁵

Scatter/gather was an early cluster-based document browsing method that performed postretrieval clustering on top-ranked documents returned from a traditional information retrieval system.⁵ The Grouper system (retired in 2000) introduced the Suffix Tree Clustering (STC) algorithm, which groups Web search results into clusters labeled by phrases

extracted from snippets.⁹ Carrot2 (www.carrot2.org) is an open source search result clustering engine that embeds STC.

Other related work from the Web, information retrieval, and data mining communities exists. Claudio Carpineto and his colleagues surveyed Web clustering engines and algorithms.⁸ Whereas all these methods focus on improving the automatic algorithmic procedure of clustering, ClusteringWiki¹⁰ adopts a user-centric approach that lets users directly edit the clustering results, leveraging the power of human computation and mass collaboration.

References

1. B.J. Gao and J. Jan, "Rants: A Framework for Rank Editing and Sharing in Web Search," *Proc. World Wide Web Conf.*, ACM, 2010, pp. 1245-1248.
2. C.J.V. Rijsbergen, *Information Retrieval*, Butterworth-Heinemann, 1979.
3. A. Broder, "A Taxonomy of Web Search," *SIGIR Forum*, vol. 36, no. 2, 2002, pp. 3-10.
4. D.E. Rose and D. Levinson, "Understanding User Goals in Web Search," *Proc. World Wide Web Conf.*, ACM, 2004, pp. 13-19.
5. M.A. Hearst and J.O. Pedersen, "Reexamining the Cluster Hypothesis: Scatter/Gather on Retrieval Results," *Proc. 19th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, ACM, 1996, pp. 76-84.
6. A. Tombros, R. Villa, and C.J. Van Rijsbergen, "The Effectiveness of Query-Specific Hierarchic Clustering in Information Retrieval," *Information Processing and Management: An Int'l J.*, vol. 38, no. 4, 2002, pp. 559-582.
7. M. Kaki, "Findex: Search Result Categories Help Users When Document Ranking Fails," *Proc. SIGCHI Conf. Human Factors in Computing Systems*, ACM, 2005, pp. 131-140.
8. C. Carpineto et al., "A Survey of Web Clustering Engines," *ACM Computing Surveys*, vol. 41, no. 3, 2009, pp. 1-38.
9. O. Zamir and O. Etzioni, "Web Document Clustering: A Feasibility Demonstration," *Proc. 21st Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, ACM, 1998, pp. 46-54.
10. D.C. Anastasiu, B.J. Gao, and D. Buttler, "ClusteringWiki: Personalized and Collaborative Clustering of Search Results," *Proc. 20th ACM Int'l Conf. Information and Knowledge Management*, ACM, 2011, pp. 573-582.

The query-processing module takes a query q and a set of applicable user preferences as input to produce a search result presentation T that respects the preferences. The editing module takes T and a user edit e as input to generate preferences with respect to q . These preferences are enforced immediately to refresh T , and are stored and managed by the preference management module.

The framework decomposes search result presentation T into a set of independent editing components. In particular, it decomposes a list into a set of preference pairs in Rants, and it

decomposes a cluster tree into a set of root-to-leaf paths in ClusteringWiki. So, it would decompose the list in Figure 1a into $\{(r_1, r_2), (r_2, r_3), (r_3, r_4), \dots\}$, and the cluster tree in Figure 3 into $\{(All, A, B, P_1), (All, A, B, P_2), (All, A, C, P_3), \dots\}$. Each edit e on presentation T leads to the addition or removal of one or several editing components, reflecting user preferences for T .

A user editing session typically involves a series of edits. Decomposition makes these edits independent of each other, leading to a more responsive and reliable editing experience. Independence of user preferences also enables

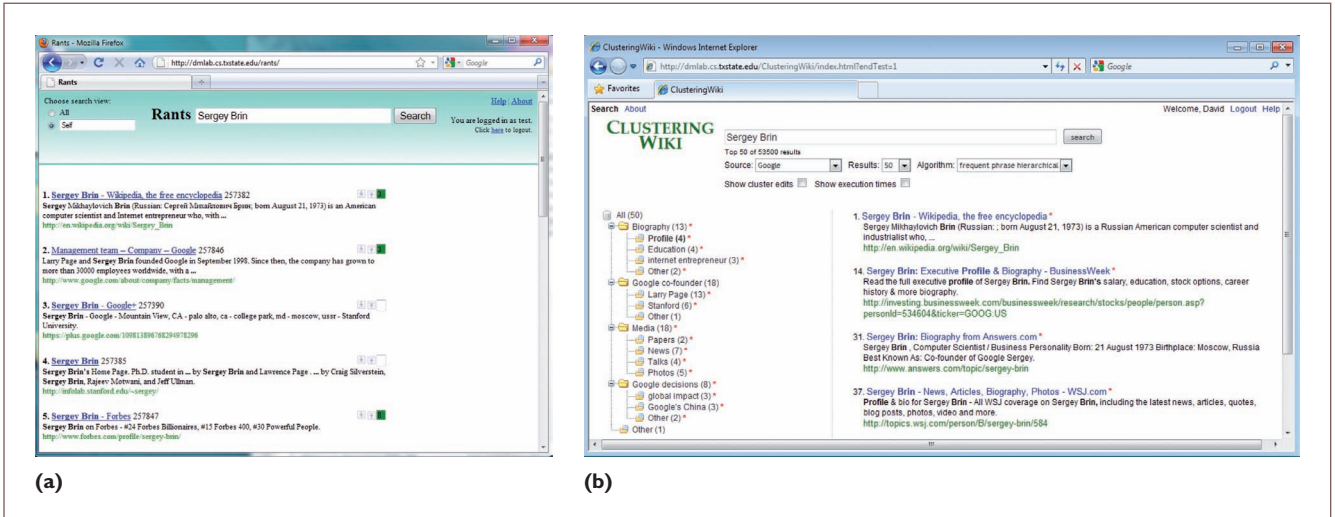


Figure 1. Our user-centric search result organization approach: (a) screenshot of Rants, and (b) screenshot of ClusteringWiki.

their straightforward aggregation and enforcement during query processing. Search results are dynamic in nature. In query processing, we need to enforce only the applicable preferences. For example, for query q , we have a stored preference (r_1, r_2) . If r_2 is missing from the set of returned results, (r_1, r_2) will become inapplicable.

Editing takes user effort. User preferences can be aggregated and shared among users. The preference management module periodically aggregates these preferences offline. With decomposition, we can treat preferences independently and aggregate them separately in a straightforward manner. Let U be the set of users whose preferences are to be aggregated. We add a preference to the set P of aggregated preferences if the percentage of users in U who have specified the preference is beyond a tunable threshold. Preferences in P might conflict. For example, $\{(r_1, r_2), (r_2, r_3), (r_3, r_1)\}$. We use a cycle detection and breaking scheme to resolve the problem. For each user u , let P_u be the set of preferences u has specified through editing. We only need to store the set difference $P_u - P$, which will be updated when P is updated.

User preferences can also be shared among similar queries and reused. For example, a user who has edited the results for the query “David Dewitt” likely wants to reuse the edits for the query “David J. Dewitt.”

We use two similarity measures to decide whether queries q and q' are similar enough to share preferences. The first, $wordSim(q, q')$, compares the keywords of q and q' . The second, $rankSim(q, q')$, compares K_q and $K_{q'}$, the top k

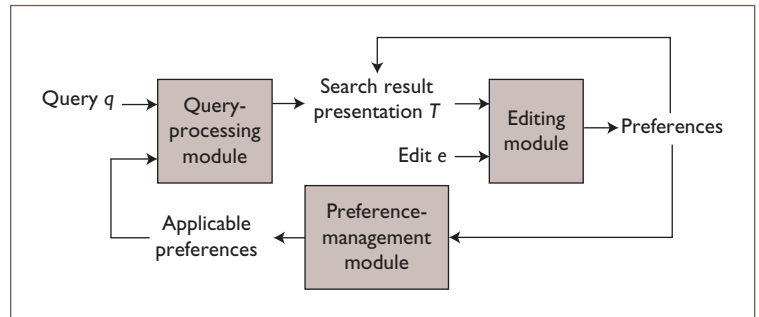


Figure 2. Main components in the Rants and ClusteringWiki architecture. The figure shows a typical workflow of the systems.

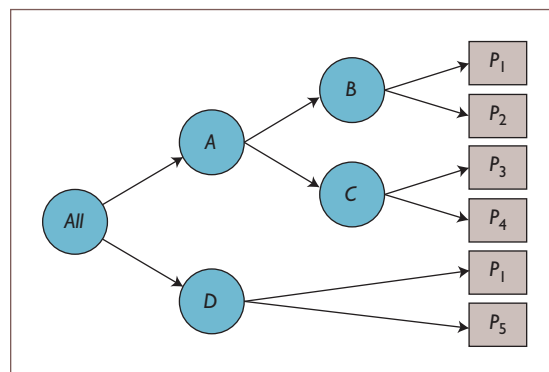


Figure 3. Example cluster tree. The figure exemplifies a hierarchical presentation of search results in ClusteringWiki.

(for example, $k = 10$) search results of q and q' . Both must pass their respective tunable thresholds. Obviously, the bigger the thresholds, the more conservative the sharing. Setting the thresholds to 1 shuts down preference sharing.

To compute $wordSim(q, q')$, we treat q and q' as sets of keywords and use $J(q, q')$, the Jaccard index for q and q' . Specifically,

$$J(q, q') = \frac{|q \cap q'|}{|q \cup q'|}.$$

We have two options for computing $rankSim(q, q')$. We can use $J(K_q, K_{q'})$ – that is, the Jaccard index for the top k results of q and q' . Or, we can use the rank-aware Kendall tau coefficient, a nonparametric statistic for measuring the degree of correspondence between two rankings. Specifically,

$$\tau(K_q, K_{q'}) = \frac{n_c - n_d}{\frac{1}{2}k(k-1)},$$

where n_c is the number of concordant pairs between K_q and $K_{q'}$, and n_d is the number of discordant pairs. The denominator is just the total number of pairs.

Suppose q has no stored preferences, and we want to find a q' whose stored preferences can be shared with q . We first compute $wordSim(q, q')$ to eliminate most of the unqualified candidates. We then compute $rankSim(q, q')$ and select a qualified q' with the largest $rankSim(q, q')$.

Both Rants and ClusteringWiki can reproduce edited presentations. In particular, after a series of user edits on T_{init} (initial query results) to produce T , if T_{init} remains the same in a subsequent query, exactly the same T will be produced after enforcing the stored user preferences generated from the previous user edits on T_{init} .

Rants

Existing systems provide two editing operations, promotion and demotion, in which a user can promote (move up) or demote (move down) a search result r for a query q by one or more positions. Let up and $down$ denote the two operations, where $up(r, 2)$ means to move r up two positions if possible (it might reach the top and not be able to continue).

How does the system interpret a move? The user intention behind a move is unfortunately ambiguous. It might be an assertion for the ranking of all results after the move, or it might indicate several pairwise preferences for the involved results only. In our framework,

we take a conservative approach and make the fewest inferences from a move. For example, if after $up(r, 2)$ by user u for query q , r surpassed r' and r'' , we store two pairs (r, r') and (r, r'') , meaning that for q , user u prefers r to appear before r' and r'' .

We adopt this *least inference principle* for several reasons. First, it generates the least, if any, ambiguity. In the previous example, although different users might intend the same move for different preferences, such preferences would at least include those two pairwise preferences. We don't even infer on the precedence relationship between r' and r'' . Second, it achieves reproducibility.

Based on this interpretation, $up(r, 2)$ is equivalent to two consecutive executions of $up(r, 1)$. Thus, in Rants, we only allow $up(r)$ and $down(r)$, meaning $up(r, 1)$ and $down(r, 1)$, indicated by the \uparrow and \downarrow arrows in Figure 1. This is not a limitation, but an emphasis on primitive functionalities, instead of syntactic sugars, for conceptual clarity. Each move results in specification of one pair (preference). The set of all pairs is maintained as a redundancy-free directed acyclic graph.

Rants also features extended editing facilities. Promotion and demotion specify relative preferences. Users will often want to specify absolute preferences as well. For example, user u might always want to see result r appear among the top three results for query q . This isn't possible through relative preferences because over time, new results for q would take the top three seats whose pairwise preferences with respect to r weren't specified and stored.

User u might want to stipulate that result r must appear among the top k . We use a pair (r, k) to capture this absolute preference. As Figure 1a shows, for each q , k can be entered into the box to the right of the \downarrow arrow.

In processing query q , we first determine a set R of relative preferences and a set A of absolute preferences that are applicable. We consider these preferences as constraints to be enforced. The enforcement adopts a *least modification principle*, in which we use as little modification as possible to enforce the constraints, and we measure the degree of modification by edit distance between the search result rankings before and after the enforcement.

We process R first. In Rants, the preference management module maintains consistency of

relative preferences, so R is completely enforceable. We can enforce a partial order in different ways, which reflects the fact that a directed acyclic graph can have many topological orderings. In graph theory, a topological ordering of a directed acyclic graph is a linear ordering of its nodes in which each node comes before all nodes to which it has outbound edges. It's a total order that's compatible with the partial order. Every directed acyclic graph has one or more topological orderings. To comply with the least modification principle, we compute a topological ordering O for R that's closest to the list of search results, L . Next, we iteratively process the edges in O in order. That is, for each $(r, r') \in O$, if r' is before r in L , move r' down to the position immediately after r . In this process, $(r', k) \in A$ might be violated. But we do nothing about it until the next stage.

Next, we process A . Absolute preferences might conflict with themselves (for example, $(r, 1)$ and $(r', 1)$ might be specified at different times) as well as with relative preferences. Rants adopts a lightweight best-effort heuristic to enforce A . To comply with the least modification principle, we sort the results in A according to their order in L , the list of search results. We then iteratively process each $(r, k) \in A$ by invoking $climb(r)$, which is recursive. If $rank(r) > k$, it moves r up by swapping r and r' . If r' blocks r , it recursively calls $climb(r')$. Result r' blocks r if $(r', k') \in A$ or $(r', r) \in R$ is violated by the planned swapping. When $rank(r) = k$, $climb(r)$ stops, or no swapping can be conducted. In this case, all results above r (including r) are blocked.

Clustering Wiki

In ClusteringWiki, search results are organized in a cluster tree T , as exemplified in Figure 3. The internal nodes contain cluster labels and are presented on the left-hand label panel. Each label is a set of keywords. The leaf nodes contain search results, and the leaf nodes for a selected label are presented on the right-hand result panel. A search result can appear multiple times in T . The root of T represents the query q and is always labeled with *All*. Labels other than *All* represent the various, possibly overlapping, subtopics of q .

A user u can edit T by creating, deleting, modifying, moving, or copying nodes. As discussed earlier, we decompose T into a set of

independent root-to-leaf paths. Thus, each edit leads to the insertion or deletion of one or more paths. Each stored path p can be either positive or negative, representing insertion and deletion, respectively. Two opposite paths, p and $-p$, will cancel each other out. User edits will be validated against a set C of consistency constraints before being stored. The set C contains predefined constraints that are specified on, for example, cluster size, tree height, and label length. These constraints maintain a favorable user interface for fast and intuitive navigation.

For a query q , ClusteringWiki clusters the search results with a default clustering algorithm (for example, frequent phrase hierarchical) to produce an initial cluster tree T_{init} . It then enforces an applicable set P of stored user preferences on T_{init} to produce a modified cluster tree T . The enforcement takes a straightforward combination of the paths in P and T_{init} . In particular, for each positive $p \in P$, if $p \notin T_{init}$, add p to T_{init} . For each negative $p \in P$, if $p \in T_{init}$, remove p from T_{init} .

The system preprocesses the combined titles and snippets of search results and uses them to build T_{init} . ClusteringWiki provides four built-in clustering algorithms: k -means flat, k -means hierarchical, frequent phrase flat, and frequent phrase hierarchical. The hierarchical algorithms recursively apply their flat counterparts in a top-down manner to large clusters. The k -means algorithms follow a strategy that generates clusters before labels. They use a simple approach to generate cluster labels from titles of search results that are the closest to cluster centers. To produce stable clusters, the typical randomness in k -means due to the random selection of initial cluster centers is removed. The parameter k is heuristically determined based on the input size.

The frequent phrase algorithms generate labels before clusters. They first identify frequent phrases using a suffix tree built in linear time using Ukkonen's algorithm. Next, they select labels from the frequent phrases using a greedy set cover heuristic, where at each step a frequent phrase covering the most uncovered search results is selected until the whole cluster is covered or no frequent phrases remain. They then assign each search result r to a label L if r contains the keywords in L . Uncovered search results are added to a special cluster labeled *Other*. These algorithms can generate meaningful labels with

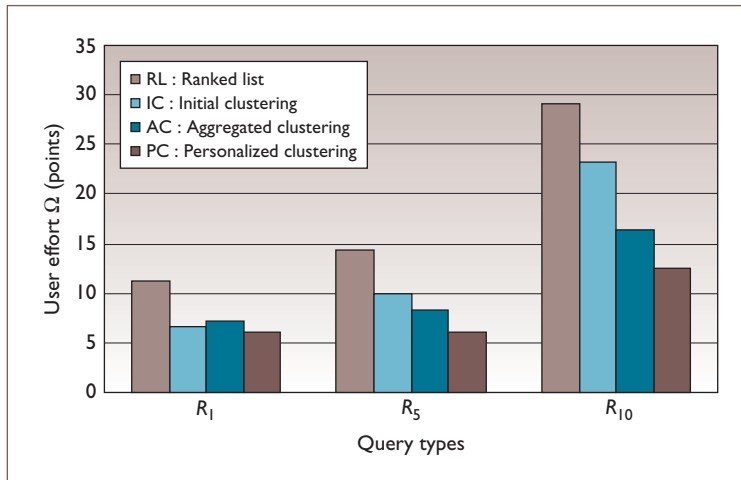


Figure 4. Utility evaluation. The figure shows the utility performance for the comparison partners in our user study.

a couple of additional heuristics – for example, a sublabel can't be a subset of a superlabel, in which case the sublabel is redundant.

To evaluate ClusteringWiki's utility in improving search performance, we conducted a user study with 22 paid participants. We measured user effort for three query types requiring identification of one, five, and 10 relevant results, respectively. The metric for user effort was the weighted number of examined cluster labels and search results assuming top-down scanning. We compared four presentations of ranked list, initial clustering, personalized clustering, and aggregated clustering. We used multiple data sources, including Google's Ajax Search API (code.google.com/apis/ajaxsearch), Yahoo's Search API (developer.yahoo.com/search/web/webSearch.html), the *New York Times* Annotated Corpus dataset (Linguistic Data Consortium), and the Tipster (disks 1–3) and Trec (disks 4–5) datasets.

Figure 4 shows the average user effort for 110 queries from the 22 users for each of the four presentations and each of the query types on the Google data source. We can observe similar trends for other data sources. These trends reveal the following:

- Clustering saves user effort, and personalized clustering is the most effective, saving up to 50 percent of user effort.
- Aggregated clustering also has significant benefits, and unlike personalized clustering, doesn't require user editing effort or login.
- Clustering's effectiveness is related to the depth of the relevant results. The lower the

results' ranking, the more effective clustering is because more irrelevant results can be skipped.

The detailed experiment setting and complete experiment results are available elsewhere.⁴

The ideas presented here can be extended to faceted search,⁵ the current de facto standard for e-commerce. In faceted search, a set of facets, each being a taxonomy, are used to organize information, allowing progressive query refinement and exploratory search. Structure-wise, the only difference between the clustering interface and faceted interface is a single hierarchy versus multiple hierarchies. Faceted search systems are costly to build and maintain. A mass-collaborative solution can greatly benefit not-for-profit community portals such as Craigslist. □

Acknowledgments

This research was supported in part by the US Department of Energy's Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344, the US National Science Foundation (NSF) under grants OCI-1062439 and CNS-1058724, and the Texas Norman Hackerman Advanced Research Program (NHARP) under grant 003656-0035-2009.

References

1. J.-R. Wen, Z. Duo, and R. Song, *Personalized Web Search*, Springer-Verlag, 2009.
2. C.D. Marming, P. Raghavan, and H. Schtze, *Introduction to Information Retrieval*, Cambridge Univ. Press, 2008.
3. A. Doan, R. Ramakrishnan, and A. Halevy, "Crowdsourcing Systems on the World-Wide Web," *Comm. ACM*, vol. 54, no. 4, 2011, pp. 86–89.
4. D.C. Anastasiu, B.J. Gao, and D. Buttler, "A Framework for Personalized and Collaborative Clustering of Search Results," *Proc. 20th ACM Int'l Conf. Information and Knowledge Management*, ACM, 2011, pp. 573–582.
5. D. Tunkelang, *Faceted Search*, Morgan & Claypool Publishers, 2009.

Byron J. Gao is an assistant professor at Texas State University. His research spans several related fields including data mining, databases, information retrieval, and bioinformatics. Gao has a PhD in computer science from Simon Fraser University. Contact him at bgao@txstate.edu.

David Buttler is a research scientist at Lawrence Livermore National Lab. His research interests are in information management systems for distributed data. Buttler has a PhD in computer science from the Georgia Institute of Technology. Contact him at buttler1@llnl.gov.


David C. Anastasiu is a PhD candidate at the University of Minnesota and was a research assistant at Texas State University at the time this research was conducted. His research interests include data mining and information retrieval. Anastasiu has an MS in computer science from Texas State University. Contact him at anast021@umn.edu.

Shuaiqiang Wang is an associate professor at Shandong University of Finance and Economics. His research interests include information retrieval, data mining, and machine learning. Wang has a PhD in computer science from Shandong University. He was a postdoctoral fellow at Texas State University at the time this

research was conducted. Contact him at swang@sdufe.edu.cn.

Peng Zhang is a research assistant professor at Florida Atlantic University. His research focuses on data mining and information security. Zhang has a PhD in computer science from the Chinese Academy of Sciences. He was a postdoctoral fellow at Texas State University at the time this research was conducted. Contact him at zhangpeng@ict.ac.cn.

Joey Jan is a research assistant at Texas State University. His research interests include information retrieval and data mining. Jan has a BS in computer science from Texas State University. Contact him at jj1258@txstate.edu.

 Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.

ANYTIME, ANYWHERE ACCESS

DIGITAL MAGAZINES

Keep up on the latest tech innovations with new digital magazines from the IEEE Computer Society. At **more than 65% off regular print prices**, there has never been a better time to try one. Our industry experts will keep you informed. Digital magazines are:

- Easy to Save. Easy to Search.
- Email notification. Receive an alert as soon as each digital magazine is available.
- Two formats. Choose the enhanced PDF version OR the web browser-based version.
- Quick access. Download the full issue in a flash.
- Convenience. Read your digital magazine anytime, anywhere—on your laptop, iPad, or other mobile device.
- Digital archives. Subscribers can access the digital issues archive dating back to January 2007.

Interested? Go to www.computer.org/digitalmagazines to subscribe and see sample articles.

