Gumshoe Quality Toolkit: Administering Programmable Search

Zhuowei Bao[*] zhuowei@cis.upenn.edu
Benny Kimelfeld[†] Yunyao Li[†] {kimelfeld,yunyaoli}@us.ibm.com
Sriram Raghavan[‡] sriramraghavan@in.ibm.com
Huahai Yang[†] hyang@us.ibm.com

ABSTRACT

Enterprise search is challenging due to various reasons, notably the dynamic terminology and domain structure that are specific to the enterprise, combined with the fact that search deployments are typically managed by domain experts who are not necessarily search experts. To address that, it has been proposed to design search architectures that feature two principles: *comprehensibility* of the ranking mechanism and *customizability* of the search engine by means of intuitive runtime rules. The proposed demonstration operates on top of an engine implementation based on this search philosophy, and provides an administrator toolkit to realize the two principles. In particular, the toolkit provides a complete visualization of the provenance (hence ranking) of search results, embeds an editor for programming runtime rules, facilitates the investigation of (the cause of) missing or low-ranked desired results, and provides suggestions of rewrite rules to handle such results.

Categories and Subject Descriptors: H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval

General Terms: Human Factors, Management

1. BACKGROUND

While search engines are very successful in retrieval on the Web, enterprise search remains an important living challenge with various sources of difficulties highlighted by the retrieval community [2,4,6,7]. Among those are the sparseness of link structure and anchor text, low economic incentive of content owners to promote easy search access, and a strong presence of dynamic terminology and jargon that are specific to the domain. Another important difficulty is the fact that enterprise search deployments are typically managed by administrators who are *domain* experts but not *search* experts—although they well understand the specific content and user needs in the domain, translating that knowledge into tuning the underlying retrieval model is nontrivial, sometimes practically impossible.

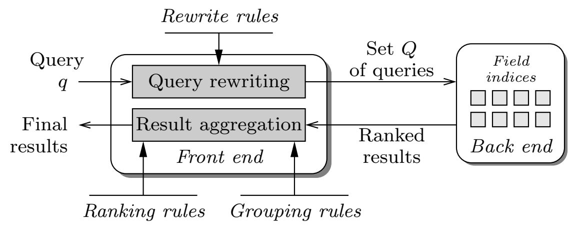


Figure 1: Search-engine architecture

Facing those challenges, research and industrial groups [5, 10] advocate the design of *programmable search*—a search paradigm that features two principles: *comprehensibility* (i.e., transparent ranking mechanisms), and *customizability* by intuitive *runtime rules* (to program domain knowledge). This demonstration is based on such a system, namely Gumshoe,[1] and is aimed to provide search administrators with the proper tooling to fully realize the two principles.

Gumshoe Search Architecture

We now outline the search architecture underlying the toolkit. Figure 1 depicts the conceptual runtime flow.

Categories, metadata fields and matchings. At backend analysis, each document is associated with a *category* and multiple *metadata fields*. The category is semantically meaningful in the domain; examples include employee directories, software pages and wiki pages. The metadata fields are extracted to leverage structural Web-page information (e.g., HTML title, actual title, headers, HTML meta fields and URL components) in retrieval, similarly to existing search systems [8, 11]. For each type of metadata we store in a (conceptually) separate index, called *field index*, the corresponding field values along with their container documents. Thus, the search index is a collection of field indices, where each document belongs to one or more of them. Given a search query and a field value, there are different types of *matchings* with various *strengths*; examples are "coverage"—exact matching up to normalization, "n-gram"—the query is an n-gram of the field, and "person name"—variants of the same person name (e.g., the first name is replaced by its first letter). Other actions taken at backend analysis (e.g., *global analysis* [11]) are not discussed here.

Top results and ranking vectors. To allow a search administrator to phrase simple and intuitive rules, the engine

[*]University of Pennsylvania, Philadelphia, PA, USA. Work done while at IBM Research – Almaden.
[†]IBM Research – Almaden, San Jose, CA, USA
[‡]IBM India Research Lab, Bangalore, India

[1]This engine is deployed in IBM intranet search [10]. More details are in the IBM page of *Infrastructure for Intelligent Information Systems* `http://www.almaden.ibm.com/cs/disciplines/iiis`.

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.
ACM 978-1-4503-1156-4/12/10.

uses the notion of *top results* as follows. Various combinations of metadata fields and matching types (implying strong matching) are specified in advance. For a given query, each document obtained by one of the specified combinations is marked as a top result. All top results are ranked higher than all non-top results. Among the top results, ranking is determined by *ranking vectors*. Specifically, the engine is configured with a collection of document *features* such as the number of top matchings for the query, the document last update time and URL length, and so on. Two top matches are ranked by comparing their ranking vectors in a lexicographic manner. But this order can be overridden by *rules*.

Rewrite Rules. Rewrite rules program the *query rewriting* component of Figure 1, and are in the spirit of the *query-template* rules [1,5,9]. Rather than a precise specification of the rule language, we give examples in a simplified language. The following rule fires when the query is of the form x info, where x belongs to a dictionary of products; it introduces a new query with the term info removed.

$$\textit{EQUALS:}\ \ x[\mathsf{in\ PRODUCT}]\ \mathsf{info} \rightarrow x$$

The next rule, involving a regular expression, fires when the query *contains* the word lotus followed by either presentations or spreadsheets; it introduces a new query with the two words replaced by lotus symphony.

$$\mathsf{lotus}\ x(\mathsf{presentations}|\mathsf{spreadsheets}) \rightarrow +\mathsf{lotus\ symphony}$$

The plus sign assigns *preference* to the new query over the original query, meaning that *among the top results*, the ones that match the new query are ranked higher than those that match the old one. These are the only rules we mention here, so we use $q_1 \rightarrow q_2$ as a shorthand notation of $x(q_1) \rightarrow +q_2$.

Other runtime rules. Runtime rules are in the form of *query pattern* $\rightarrow$ *action*. Two additional types of rules are *grouping rules* and *ranking rules*. In a grouping rule, results of specified categories are clustered together. In a ranking rule, results of specified categories are clustered together and ranked higher than results of other categories.

2. TOOLKIT DESCRIPTION

The demonstrated toolkit is aimed for use by search administrators in the enterprise, with the goal of realizing the two principles of a programmable search engine: comprehensibility and customizability. The toolkit operates alongside the underlying search server and communicates with it (through a REST API) to realize its functionality. A session begins with a simple (typical) search interface: the administrator poses a keyword-search query, and in return the toolkit displays pages of ranked results as obtained from the search server. Beyond that, the functions of the toolkit are classified into three categories that we explain next.

2.1 Provenance Tracking

The *provenance* of a result entails the reformulations, field indices, ranking-vector values, and the rules that were involved in the process of retrieving the result by the search engine. The toolkit visually presents the provenance of selected search results through several display components. For that, attached to each search result is a "Track" check button that, when turned on, activates the provenance display for the result. In Figure 2 the search query is "employee skills," and the provenance of four results is displayed.

Trellis graph. The *trellis graph* shows how a result is obtained from reformulations and field indices, and it has four vertical layers. The first layer contains the query reformulations obtained by invoking the rewrite rules (at runtime). In Figure 2, the rule skills $\rightarrow$ expertise fired, so we get two reformulations: employee skills and employee expertise. The second layer contains the field indices in which the considered results were found. A reformulation is connected by an edge to a field index if the index contains a match for that reformulation. In Figure 2, for example, the index "Title Content" contains employee expertise while "Title Name" contains employee skills. It may be the case that the number of such indices is too large to display, and then the indices of lower quality will be blurred to avoid overwhelming the administrator (e.g., "H1s" of Figure 2). The third layer in the trellis graph contains the actual relevant entries in the field indices, represented by snippets. Each relevant entry has an incoming edge from each containing index; that edge is labeled with the type of match identified (i.e., n-gram or coverage). The last layer contains the result themselves, each with incoming edges from their fields (in the third layer). Figure volumes are used for visualizing importance: larger circles (e.g., that of employee expertise) represent reformulation that are ranked higher, larger ovals (e.g., that of "Title Name") represent higher-quality fields, and thicker edges (e.g., that of "coverage") represent higher-quality matches.

Ranking vector and rules. In addition to the trellis graph, the toolkit visualizes the ranking vector and specifies the fired rules (see the left pane in Figure 2). The layout of the ranking vector is by a column graph that allows for easy comparison among results. When hovering over a specific result, the fired rules relevant to (i.e., needed to fire in order to obtain) that result are highlighted.

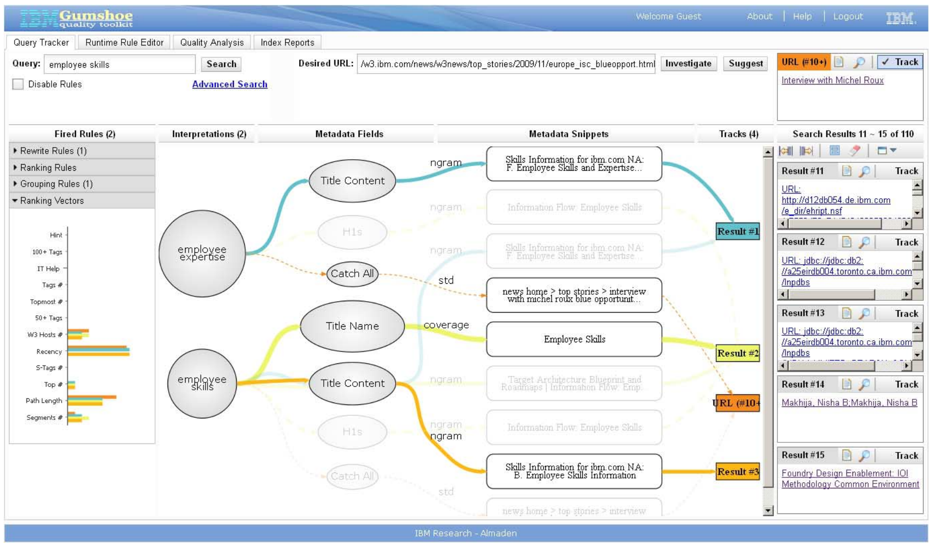
Result investigation. An important functionality of the toolkit is the ability to paste a new URL and investigate that URL w.r.t. the existing (top) results. This functionality is crucial for understanding why a specific desired result is ranked as it is compared to the top results. The administrator pastes the URL of the desired result in the "Desired URL" field (see the top of Figure 2) and, upon clicking on "Investigate" the desired result is tracked alongside the other results. For illustration, the darker orange result in Figure 2 (denoted "URL(#10+)" to specify that it is ranked lower than the tenth result) is an investigated URL.

2.2 Rule Editing

A *rule editor* embedded in the toolkit allows the administrator to introduce new runtime rules and edit/delete existing ones. Once the rules are *saved*, the editor signals the search server (through the REST command) to reload the rule files, thereby avoiding the need to manually restart the server outside the toolkit. Observe that the rules are placed in various files (corresponding to various kinds of rule semantics); to ease the access to the rules, the editor has filtering fields in the form of substring matching.

2.3 Rule Suggestion

Once a desired result is investigated, the administrator can ask the toolkit to suggest rules. In the current version of the toolkit, the suggestions are restricted to those of the form $s \rightarrow t$. We refer the reader to Bao et al. [3] for the details of the method deployed to produce suggestions. In essence, a suggested rule should satisfy two properties. First, it should



IBM Gumshoe quality toolkit
Welcome Guest
About | Help | Logout
Query Tracker
Runtime Rule Editor
Quality Analysis
Index Reports
Query: employee skills
Search
Disable Rules
Advanced Search
Desired URL: /w3.ibm.com/news/w3news/top_stories/2009/11/europe_isc_blueopport.html
Investigate
Suggest
URL (#10+)
Track
Interview with Michel Roux
Fired Rules (2)
Rewrite Rules (1)
Ranking Rules
Grouping Rules (1)
Ranking Vectors
Interpretations (2)
Metadata Fields
Metadata Snippets
Tracks (4)
Search Results 11 ~ 15 of 110
employee expertise
employee skills
Title Content
H1s
Catch All
Title Name
ngram
std
coverage
Skills Information for ibm.com NA: F. Employee Skills and Expertise...
Information Flow: Employee Skills
news home > top stories > interview with michel roux blue opportunit...
Employee Skills
Target Architecture Blueprint and Roadmaps | Information Flow: Emp.
Skills Information for ibm.com NA: B. Employee Skills Information
Result #1
Result #2
URL (#10+)
Result #3
Result #11
Result #12
Result #13
Result #14
Result #15
Makhija, Nisha B;Makhija, Nisha B
Foundry Design Enablement: IOI Methodology Common Environment
IBM Research - Almaden

Figure 2: Screen shot: provenance tracking

be *effective* in the sense that it pushes the desired match up into the top-k matches, where k is a fixed parameter (5 in our implementation). The second property is informal: the rule should be *natural* in the sense that it corresponds to a semantically justified rewrite (so that it makes sense to the administrator). For example, skills $\rightarrow$ expertise is natural as the two sides have similar meanings, and so is download $\rightarrow$ issi in IBM as ISSI is a prominent download/installation tool inside IBM. To rank the extent to which a rule is natural, we use a supervised machine-leaning approach.

2.4 Implementation

The toolkit is built as a Web application. On the server side the implementation is by Java (Servlet) technology carried by Jetty WebServer.[2] We use Graphviz[3] to compute the layout of the provenance trellis graph. On the client side the toolkit uses the Dojo Toolkit[4] for the general layout. Data is communicated between the client and the server mainly in JSON format.

3. DEMONSTRATION

The goal of the demonstration is to illustrate how the principles of programmable search can be realized and facilitated by the proper tooling and visualization. Our plan is for the audience to experience using the toolkit as it is intended to be used by search administrators in the enterprise (and as it is currently being used by them in IBM). Specifically, following a brief description of the background and underlying search paradigm we will demonstrate various use cases.

Provenance visualization. We will consider various search queries and explain the provenance of the top results for these queries (including the corresponding runtime rules, reformulations, field indices and ranking vectors).

Desideratum investigation. For one or more search queries, we will consider desired search results that are low ranked, and use the provenance visualization to explain their relative low ranking compared to other results.

Runtime programming. We will use the embedded rule editor to insert, edit and delete rules, and observe their immediate effect on the search results. In particular, we will show how desired results are pushed to the top by phrasing appropriate rewrite rules.

Rule suggestion. We will consider cases of low-ranked desired results and ask the toolkit to suggest rewrite rules. We will further discuss the quality of these rules and explain the underlying learning machinery.

Acknowledgments

We are deeply grateful to Nicole Dri and Gayle S. Borge for providing valuable feedback on the toolkit. We also thank Shivakumar Vaithyanathan for insightful discussions.

4. REFERENCES

[1] G. Agarwal, G. Kabra, and K. C.-C. Chang. Towards rich query interpretation: walking back and forth for mining query templates. In *WWW*, 2010.

[2] O. Alhabashneh, R. Iqbal, N. Shah, S. Amin, and A. E. James. Towards the development of an integrated framework for enhancing enterprise search using latent semantic indexing. In *ICCS*, 2011.

[3] Z. Bao, B. Kimelfeld, and Y. Li. Automatic suggestion of query-rewrite rules for enterprise search. In *SIGIR*, 2012.

[4] P. A. Dmitriev, N. Eiron, M. Fontoura, and E. J. Shekita. Using annotations in enterprise search. In *WWW*, 2006.

[5] R. Fagin, B. Kimelfeld, Y. Li, S. Raghavan, and S. Vaithyanathan. Rewrite rules for search database systems. In *PODS*, 2011.

[6] R. Fagin, R. Kumar, K. S. McCurley, J. Novak, D. Sivakumar, J. A. Tomlin, and D. P. Williamson. Searching the workplace web. In *WWW*, 2003.

[7] D. Hawking. Challenges in enterprise search. In *ADC*, volume 27 of *CRPIT*, 2004.

[8] Y. Li, R. Krishnamurthy, S. Vaithyanathan, and H. V. Jagadish. Getting work done on the web: supporting transactional queries. In *SIGIR*, 2006.

[9] I. Szpektor, A. Gionis, and Y. Maarek. Improving recommendation for long-tail queries via templates. In *WWW*, 2011.

[10] S. Vaithyanathan. Building search systems for the enterprise, 2011. SIGIR'11 industrial track keynote.

[11] H. Zhu, S. Raghavan, S. Vaithyanathan, and A. Löser. Navigating the intranet with high precision. In *WWW*, 2007.

[2] http://jetty.codehaus.org/jetty/

[3] http://www.graphviz.org/

[4] http://dojotoolkit.org/