Automatic Suggestion of Query-Rewrite Rules for Enterprise Search

Zhuowei Bao
University of Pennsylvania
Philadelphia, PA 19104, USA
zhuowei@cis.upenn.edu

Benny Kimelfeld
IBM Research – Almaden
San Jose, CA 95120, USA
kimelfeld@us.ibm.com

Yunyao Li
IBM Research – Almaden
San Jose, CA 95120, USA
yunyaoli@us.ibm.com

ABSTRACT

Enterprise search is challenging for several reasons, notably the dynamic terminology and jargon that are specific to the enterprise domain. This challenge is partly addressed by having domain experts maintaining the enterprise search engine and adapting it to the domain specifics. Those administrators commonly address user complaints about relevant documents missing from the top matches. For that, it has been proposed to allow administrators to influence search results by crafting query-rewrite rules, each specifying how queries of a certain pattern should be modified or augmented with additional queries. Upon a complaint, the administrator seeks a semantically coherent rule that is capable of pushing the desired documents up to the top matches. However, the creation and maintenance of rewrite rules is highly tedious and time consuming. Our goal in this work is to ease the burden on search administrators by automatically suggesting rewrite rules. This automation entails several challenges. One major challenge is to select, among many options, rules that are "natural" from a semantic perspective (e.g., corresponding to closely related and syntactically complete concepts). Towards that, we study a machine-learning classification approach. The second challenge is to accommodate the cross-query effect of rules—a rule introduced in the context of one query can eliminate the desired results for other queries and the desired effects of other rules. We present a formalization of this challenge as a generic computational problem. As we show that this problem is highly intractable in terms of complexity theory, we present heuristic approaches and optimization thereof. In an experimental study within IBM intranet search, those heuristics achieve near-optimal quality and well scale to large data sets.

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

General Terms: Algorithms, Performance, Theory

Keywords: Query reformulation, query-rewrite rules, enterprise search

SIGIR'12, August 12–16, 2012, Portland, Oregon, USA.

1. INTRODUCTION

While search engines on the Web are highly successful in content retrieval, enterprise search remains an important living challenge [2, 11, 12]. The retrieval community identified various sources of difficulty [5, 7, 10, 11], including the sparseness of link structure and anchor text, low economic incentive of content providers to promote easy search access, and a strong presence of dynamic, domain-specific terminology and jargon. Another practical difficulty lies in the fact that enterprise search deployments are typically managed by administrators who are domain experts but not search experts. Hence, although these administrators well understand the specific content and search needs of the domain, translating that knowledge into tuning the underlying retrieval model is a nontrivial (and sometimes impossible) task.

To address the above challenges, research and industrial groups [8, 9, 21] advocated the design of search architectures that are *(1)* easily comprehensible (featuring fairly transparent ranking mechanisms), and *(2)* easily customizable through intuitive *runtime rules* to program domain knowledge. One important type of runtime rules is that of *query-rewrite rules* (or *rewrite rules* for short) that, in essence, introduce human-controlled *query reformulation* [18, 19] at runtime. By augmenting or modifying the posed search query, rewrite rules enable administrators to easily satisfy a wide range of maintenance needs such as: adaptation of terminology (e.g., produce a query where green card is replaced with permanent residency), query specialization (e.g., h5n1 instead of seasonal flu, or issi,[1] the company's software installation tool, instead of download), query generalization (e.g., intranet account management instead of intranet password change), noise removal (e.g., elimination of web page from a query like james allan web page), and so on.

This work is motivated by a deployment of the above architecture in the IBM internal search. There, the type of tasks that dominate the effort of administrators is that of pushing up relevant documents for specific search queries. Such a task is typically triggered by an employee complaint mentioning a specific query and a relevant document (URL) missing from the top results (e.g., the first page). For example, such a complaint can be that http://issi.ibm.com/lnotes[2] is missing from the top results for lotus notes download. In turn, the administrator can phrase a rewrite rule to accommodate the potential mismatch between the query and the document. Those rewrite rules are in the spirit of (actually, a generalization of) the "query template" rules [1, 20]. In

[1]IBM Standard Software Installer.
[2]We replaced each example URL with a simplified variant.

particular, a rule is usually applicable to multiple queries (rather than to just one), and it usually results in *augmentation* (rather than *replacement*) of the query by the newly generated query. For example, the administrator can introduce the rule download → issi that, in effect, incorporates the results for lotus notes issi into those for lotus notes download whenever the latter query is posed. In Section 2 we describe our engine architecture and rules in more detail.

Manually formulating rewrite rules to handle complaints is an extremely tedious and time consuming process. First, the administrator needs to inspect the specified document in order to come up with relevant rules. For each relevant rule, a rough estimation is made on its effect on other queries (and for that, query logs may be consulted). Next, the administrator tries out her rule of choice (in a sandbox environment). If the desired effect (pushing up the desired match) is not realized, this process is repeated with alternative rules, sometimes until the conclusion that the problem lies in the ranking algorithm and/or the backend analytics. In this work, we seek to aid administrators of enterprise search by automatically producing rewrite-rule suggestions that are already validated as problem solvers. Towards this goal, we need to overcome two major challenges, which are the subject of this paper. These challenges are generic to essentially every search system that features administration by rewrite rules, and we confront these challenges in a level of abstraction that departs from our specific testbed.

Challenge 1: Generating Intuitive Rules. A straightforward approach towards suggesting rewrite rules is to generate candidates from the desired document (or specific parts thereof), and filter out the candidates that fail to achieve the desired effect. However, the number of possible rules obtained in this way can be overwhelmingly large. For example, in our experimentation over real data (within IBM intranet search), our automation often reaches around 100 suggestions, and in some cases around 1000. More importantly, the vast majority of the suggested rules do not make sense to the administrator; that is, they are not rules of the kind she would devise or perceive as intuitive. For illustration, Figure 1 shows some of the suggestions we obtained. In the top part of the table in the figure, relating to seasonal flu, the top two are reasonable but the other hardly make sense. Note, however, that determining what "reasonable" or "sense making" means may require domain knowledge. As an example, IBM uses SCIP for consulting on organizational reconstruction; thus, the top rewrites for change management in the bottom part of Figure 1 make sense. Yet one can hardly accept the other rules in that part.

We refer to a rule that makes sense as *natural*. Formulating natural rules is necessary, as administrators should be confident in the semantic justification of each rule. Put differently, comprehensiveness, which is at the heart of the architecture philosophy, is violated if rules are inconsistent with human judgment. (This problem does not arise in online query reformulation/expansion [4] that affect only the internals of the search process.) Thus, crucial to realizing automatic rule suggestion is a component that classifies rules into natural and unnatural ones. In Section 3 we present a machine-learning approach to realizing such a component.

A related problem is that of recommending query alternatives to end users [3, 13, 22]. However, a query alternative applies just to the user-posed query, hence is different from a rewrite rule. Furthermore, the techniques there are heavily based on query logs and/or anchor text, which may be sparse in the enterprise Intranet. The same holds true for the work of Kraft and Zien [14], aiming to find reformulations that are "closely related" to the original query. Moreover, in these works a recommendation (or reformulation) has no effect beyond the specific engine-user interaction at hand; it is not the case here, as we discuss in the next challenge.

s	t
seasonal flu	avian flu
seasonal flu	h5n1
seasonal flu	flu employee
seasonal flu	and IBM h5n1
seasonal	avian
seasonal	h5n1
seasonal	flu
seasonal	you and IBM
...	...
change management	scip
change management	strategy & change internal practice
change management	welcome to strategy
change management	welcome strategy
change management	strategy change & internal
change management	to strategy change internal practice
management	internal
management	index pages
management	internal practice scip
...	...

Figure 1: Automatically generated rules $s \rightarrow t$

Challenge 2: Cross-Query Effect. As a rewrite rule can affect multiple queries, its inclusion may actually reduce the overall search quality. Further, a rule can cancel the positive effects achieved by previous rules. To illustrate, in the example of Figure 4 (which we discuss in detail throughout Section 4) the rule spreadsheets → symphony rewrites spreadsheets download into symphony download, resulting in the document d_2 being a top match. Following a complaint on the query lotus notes download, we introduce the rule r_1: download → issi. However, r_1 also affects our previous spreadsheets download by pushing the desired d_2 below d_1.

To address the above problem, the search administrator maintains a representative benchmark of queries and desired matches (possibly weighted by popularity) that includes those indicated in past complaints. To estimate the effect of a candidate rule, search quality is evaluated (using a quality estimator of choice, e.g., DCG) by running the engine against the benchmark. If a negative effect is detected, the administrator can avoid the rule, consider a new rule, or just accept the loss in quality. But here, automation can contribute significantly in terms of both effort and quality. We can allow the system to automatically suggest the selection of a subset of the rules to optimize quality (w.r.t. the underlying estimator). Moreover, the administrator can then choose *multiple* rules (instead of just one) to address a specific complaint, and thereby enrich the search space for the system. In the above example, for instance, if the system had the rule notes download → notes issi at its disposal, it could suggest to use it instead of r_1, and thus maintain the top results for both spreadsheets download and lotus notes download. In Section 4, we formalize this idea as a combinatorial optimization problem. Unfortunately, this problem turns out to be computationally hard (NP-hard), and even hard to approximate within any nontrivial ratio. Nevertheless, we propose greedy heuristics and optimizations thereof.

A related optimization problem has been recently studied by Ozertem et al. [19], where one is given a collection of

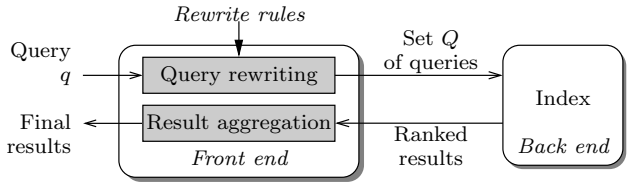

Figure 2: Search-engine architecture

query reformulations (each applying to one query) to select from, and the goal is to maximize the probability of a click in a user model built from the query log.

Organization. The rest of the paper is organized as follows. In Section 2 we briefly describe our search architecture. We study the above two challenges in Sections 3 and 4, respectively. In Section 5 we present an experimental study within IBM intranet search, and we conclude in Section 6.

2. SEARCH-ARCHITECTURE OVERVIEW

In this section, we outline the architecture of the engine deployed in IBM intranet search [21], to the extent needed as background for this work.[3] Figure 2 depicts a simplified description of the conceptual runtime flow. The front-end takes the input search query q and, by applying rewrite rules, produces a set Q of queries. In turn, the queries in Q are evaluated against the index, resulting in a list of results for each query. The results are then aggregated together to produce the final ranked list.

Similarly to existing search methodologies [6, 15, 23], this engine leverages structural information from Web pages (such as links, anchor text, headers and titles) in retrieval. The basic idea is to produce high-quality *fields* associated with each document, and at runtime use these fields for ranking.

Rewrite rules program the *query rewriting* component of Figure 2 to affect the search as needed. The rewrite rules in our engine are in the spirit of the *query-template* rules [1, 9, 20]. Rather than giving an elaborate 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 is belongs to a dictionary of products; it introduces a new query with the term info removed.

$$EQUALS: \quad x[\text{in PRODUCT}]\ \text{info} \rightarrow x$$

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.

$$CONTAINS: \quad \text{lotus}\ x(\text{presentations}|\text{spreadsheets}) \rightarrow \text{lotus symphony}$$

The next rule fires when the query contains msn search, and introduces a new query with msn search replaced by bing.

$$CONTAINS: \quad \text{msn search} \rightarrow_+ \text{bing}$$

The plus sign in $\rightarrow_+$ assigns *preference* to the new query (containing bing) over the original query (containing msn search). In effect, results for the new query are rewarded compared to those for the original query. We do not discuss here the details of this reward, as they are of low importance to this work.

To simplify our study, this work is restricted to rewrite rules of the form *CONTAINS:* $s \rightarrow_+ t$, which we denote simply as $s \rightarrow t$, where s and t are terms. We focus on this particular type of rules since, besides their simplicity, we found that they are the most commonly used in our enterprise. Moreover, with this restriction the problems we consider are already challenging. While some of our contributions can be easily extended to more general rules, others require nontrivial future effort.

[3]More details on this search engine can be found in the IBM page of *Infrastructure for Intelligent Information Systems* http://www.almaden.ibm.com/cs/disciplines/iiis.

3. RECOGNIZING NATURAL RULES

In this section, we consider the first challenge discussed in the introduction, where we explained the intuition underlying the notion of *natural* rewrite rules and their importance to the realization of automatic rule suggestion in enterprise search. Our approach is to first generate a (possibly large) set of candidate rules by pure syntactic means, and then to classify them through (supervised) machine learning.

Candidate generation. The use case we consider involves a query q and a desired match (document) d that is missing from the top results. Our generation of candidate rules $s \rightarrow t$ is fairly straightforward: we produce a set S of left-hand-side candidates, a set T of right-hand-side candidates, and output the Cartesian product $S \times T$. The set S consists of all the n-grams (subsequences of n tokens) of q, where n is bounded by a constant (5 in our implementation). In principle, we could similarly choose T as the set of all n-grams of d (which could result in a huge number of candidates, even if n is bounded). But in our implementation, T consists of the n-grams just from the high-quality fields of d (produced during back-end analysis). Recall that those fields were discussed in Section 2. As an example, suppose that q is the query "change management info" and one of the considered fields is "welcome to scip strategy & change internal practice." The following are among the candidate rules.

- management → scip
- change → strategy & change internal
- change management → scip strategy

Even with the restriction to high-quality fields, the above step may generate a large number of candidate rules, most of which are unnatural (hence, useless to the administrator), as illustrated in Figure 1. We address this problem by taking a machine-learning approach: we identify a set of features and learn classification models from manually labeled data to classify rules into natural and unnatural ones.

Features. Table 1 lists the feature set we use for classifying rules. These features fall into three categories: syntactic features, features based on query-log statistics, and features based on corpus statistics. Next, we briefly discuss the features in each category. We denote the examined rule as $s \rightarrow t$, where s and t are strings of words.

The syntactic features can be indicative of the syntactic coherence of a rule. For example, using the Boolean features $beginSW(r)$ and $endSW(r)$ follows our observation that when s or t begins or ends with stop words, the rule is rarely deemed natural. We consider two types of stop words: those from conventional (English) dictionaries and those from domain-specific dictionaries (including words like "welcome," "index" and "homepage").

Table 1: Features for natural-rule recognition; the considered rule is $s \to t$, and u refers to either s or t

Syntactic	
$beginSW(r)$	Whether u begins with a stop word
$endSW(r)$	Whether u ends with a stop word
$\|r\|$	Number of tokens in u
Query-Log Statistics	
$\log refFreq(s,t)$	Log. of the s-to-t reformulation frequency
Corpus Statistics	
$\log freq(u)$	Log. of the frequency of u
$\log freq(s \wedge t)$	Log. of concurrence frequency of s and t
$\log HQfreq(u)$	Log. of the frequency of u in titles

The category of query-log statistics has the single feature $\log refFreq(s,t)$, representing common wisdom on query reformulation. In our case, we analyzed the query log of (four months of) intranet search at IBM. For a pair q_1 and q_2 of queries, $refFreq(q_1,q_2)$ is the number of sessions that begin with q_1 and contain q_2 (posed later than q_1 in the session).

Under the category of corpus statistics we have numerical statistics on s and t drawn from our engine index. The number $freq(u)$ counts the documents containing u as an n-gram (where u is s in one feature, and t in another). Similarly, $freq(s \wedge t)$ counts the documents containing both s and t as n-grams. We use $\log freq(u)$ and $\log freq(s \wedge t)$ to capture the popularity of s and t as well as correlation thereof. Finally, $HQfreq(u)$ is the frequency of u in our high-quality fields, which roughly reflects the popularity of u in titles.

Classification models. We implemented two classification models over our vector $\vec{f}$ of features. The first model is simply a linear classifier, which we trained on manually labeled data via SVM. The second model, which we found to provide a significant improvement over the first, generalizes the first by incorporating a *Decision Tree* (DT for short). More specifically, the second model is a restricted version of a DT with linear-combination splits [16, 17], and we call it *rDTLC* for short. Given the vector $\vec{f}$ of features, a DT with linear splits has a condition $\sum a_i f_i \leq \tau$ in each node (where the a_i and τ are to be learned in training). Our rDTLC restricts this family by *(1)* bounding the depth (by 3 in our implementation), *and (2)* having univariate splits (i.e., one-variable comparisons to thresholds) in all but the bottommost levels. Figure 3 illustrates our rDTLC. Note that here we essentially follow Kraft and Zien [14], who studied a problem of a similar flavor, that used a linear classifier and a standard decision tree (rather than rDTLC).

Using an algorithm for learning a linear classifier (e.g., SVM), learning an rDTLC can be done straightforwardly by selecting thresholds from the feature values in the training data, trying out every combination of the upper-level thresholds τ_i, and taking the combination that minimizes the classification error. For efficiency sake, we implemented a fix-and-optimize heuristic where we optimized one threshold at a time while fixing the others.

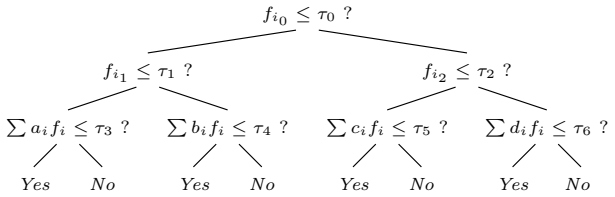

Figure 3: An rDTLC

4. OPTIMIZING MULTI-RULE SELECTION

We now consider the second challenge discussed in the introduction: selecting a set of rewrite rules from a large set of previously defined rules, so as to maximize the overall quality of search results. We formalize this task as a combinatorial optimization problem, discuss its (intractable) complexity, and propose heuristic solutions.

4.1 Administration Settings

A *rule-administration setting* (or just *administration setting* for short) is essentially an abstraction of the search-engine's interplay between queries, rewrite rules, and documents. It consists of a set R of rewrite rules and a graph G that we call the *rule-administration graph* (or just *administration graph* for short). A rule in R transforms an input *user query* q into a *rewritten query* q'. For brevity and clear distinction between the two types of queries, we refer to q (the user query) simply as a *query*, and to q' (the rewritten query) as an *r-query*. A query and an r-query are simply finite sequences over an infinite alphabet of tokens (words), and a rule $s \to t$ in R is defined similarly to the previous sections (i.e., it replaces s, a subsequence of tokens in q, with a sequence t). The administration graph is a tripartite graph describing the relationships between queries, r-queries and documents. Before we give the precise definitions, we introduce our running example for this section.

EXAMPLE 4.1. Figure 4 shows an administrator setting (R,G). The set R is depicted in the top of the figure and the graph G is depicted in the bottom. □

Formally, an administration graph is a directed, edge-weighted graph $G = (V, E, w)$, where:

- V is the union of three disjoint sets of nodes: V_{q} is a set of queries, V_{r} is a set of r-queries, and V_{d} is a set of documents.
- E is the union of three disjoint sets of edges: $E_{\mathrm{qr}} \subseteq V_{\mathrm{q}} \times V_{\mathrm{r}}$ is a set of *reformulations*, $E_{\mathrm{qd}} \subseteq V_{\mathrm{q}} \times V_{\mathrm{d}}$ is a set of *query matchings*, and $E_{\mathrm{rd}} \subseteq V_{\mathrm{r}} \times V_{\mathrm{d}}$ is a set of *r-query matchings*.
- $w : E \to \mathbb{R}_+$ assigns a positive score to each matching in $E_{\mathrm{qd}} \cup E_{\mathrm{rd}}$, and is zero on all the edges of E_{qr}.

EXAMPLE 4.2. Consider again our running example. Each side of the tripartite graph G is surrounded by a dashed rectangle. Let $q \in V_{\mathrm{q}}$ be the query lotus notes download. The outgoing edges of q include the (zero-weight) reformulation to the r-query $q' =$ lotus notes issi in V_{r}, and the query matching to the document $d_1 \in V_{\mathrm{d}}$ (which, for presentation sake, is associated with the URL http://issi.ibm.com/lnotes). The weight of the latter edge is 2, representing the extent to which the engine scores the matching of d_1 to q. A higher score (namely 5) is assigned to the matching of d_1 to q', as indicated by the weight of the r-query matching (q', d_1). □

In principle, the same string can be represented by two distinct nodes: a query in V_{q} and an r-query in V_{r}. So a

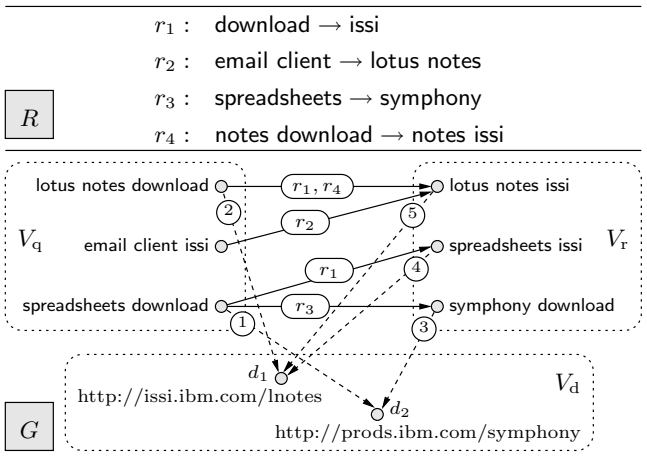

Figure 4: The administration setting (R, G) of the running example

node in $V_q \cup V_r$ should have an *identifier*, which we omit from the formal model to simplify the presentation.

Let G be an administration graph, and let $q \in V_q$ be a query. Observe that a path in G from q to d consists of either one edge (in E_{qd}) or two edges (one in E_{qr} and the other in E_{rd}). If G has a path (of length one or two) from the query q to the document d, then we denote by $\text{score}(d|q)$ the maximal weight of a path from q to d. For a query q and a natural number k, we denote by $\text{top}_k[q|G]$ the series of k documents with the highest $w(q, d)$, ordered in descending $w(q, d)$; if fewer than k documents are reachable from q via directed paths, then $\text{top}_k[q|G]$ is restricted to only the reachable documents (hence, $\text{top}_k[q|G]$ may have fewer than k documents). We implicitly assume an underlying linear order among the documents to resolve ties.

Example 4.3. Consider again our running example (Figure 4). Let q be the query spreadsheets download. There are two paths from q to the document d_2: a direct edge, and through the r-query symphony download. Since the latter has the maximal weight, 3, among all the paths from q to d_2, we get that $\text{score}(d_2|q) = 3$. We can similarly verify that $\text{score}(d_1|q) = 4$. In particular, $\text{top}_1[q|G]$ is the series (d_1), and $\text{top}_2[q|G]$ (as well as $\text{top}_3[q|G]$) is the series (d_1, d_2). □

For a rewrite rule r and query q, let $r(q)$ denote the r-query that is obtained from q by applying r. An *administration setting* is a pair (R, G), where R is a set of rewrite rules and G is an administration graph, such that the set E_{qr} of reformulations of G is the set $\{(q, r(q)) \mid q \in V_q \wedge r \in R\}$. For a reformulation $e = (q, q') \in E_{qr}$, the set of rules $r \in R$ with $r(q) = q'$ is denoted by $R(e)$.

Example 4.4. Consider again the administration setting (R, G) of our running example. For the edge e_1 from $q_1 =$ lotus notes download to $q_1' =$ lotus notes issi we have $R(e_1) = \{r_1, r_4\}$ (where the r_i are specified in the top of the figure). Similarly, for $q_2 =$ email client issi and the reformulation $e_2 = (q_2, q_1')$ we have $R(e_2) = \{r_2\}$. □

4.2 Abstract Rule Optimization

Consider an administration setting (R, G). Given a subset R' of R, we denote by $G_{R'}$ the administration graph that is obtained from G by removing from E_{qr} every reformulation e that is produced by none of the rules in R' (i.e., $R'(e) = \emptyset$).

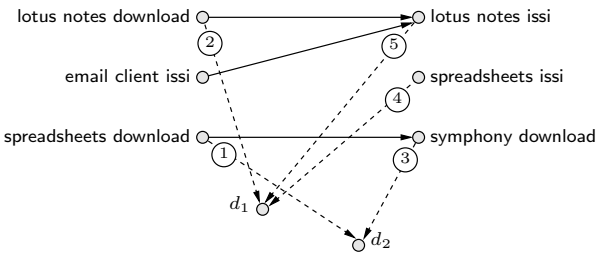

Figure 5: The graph $G_{R'}$ for $R' = \{r_2, r_3, r_4\}$

Example 4.5. Consider again our running example (Figure 4), and let R' be the set $\{r_2, r_3, r_4\}$ (i.e., $R \setminus \{r_1\}$). The graph $G_{R'}$ is depicted in Figure 5. Observe that unlike G, the graph $G_{R'}$ has no edge from the query $q_1 =$ spreadsheets download to the r-query $q_1' =$ spreadsheets issi, because $R'(e_1) = \emptyset$, where e_1 is the edge (q, q') of G (recall that $R(e_1) = \{r_1\}$). However, there is an edge from the query $q_2 =$ lotus notes download to the r-query $q_2' =$ lotus notes issi since, although for $e_2 = (q_2, q_2')$ the set $R'(e_2)$ lost r_1, this set still contains r_4. □

Let G be an administration graph. A *desideratum* is a function $\delta : V_q \to 2^{V_d}$ that maps each query $q \in V_q$ to a set $\delta(q) \subseteq V_d$ of *desired matches*. A *quality measure* μ determines a quality score[4] for each query q based on the series $\text{top}_k[q|G]$ and the set $\delta(q)$, for a natural number k of choice. We denote this score by $\mu(\text{top}_k[q|G], \delta(q))$. As an example, *precision at k* is the following μ:

$$\mu(\text{top}_k[q|G], \delta(q)) = \begin{cases} \frac{|\text{top}_k[q|G] \cap \delta(q)|}{|\text{top}_k[q|G]|} & \text{if } \text{top}_k[q|G] \neq \emptyset; \\ 0 & \textit{otherwise.} \end{cases} \quad (1)$$

As another example, DCG_k (without labeled relevance scores) is the following μ:

$$\mu(\text{top}_k[q|G], \delta(q)) = \sum_{i=1}^{j} \frac{2^{a_i} - 1}{\log_2(i+1)}, \quad (2)$$

where $\text{top}_k[q|G] = (d_1, \ldots, d_j)$, and each a_i is 1 if $d_i \in \delta(q)$ and 0 otherwise.

The *top-k quality* of G, denoted $\mu_k(G, \delta)$, is obtained by summing up the scores across all the queries:

$$\mu_k(G, \delta) \stackrel{\text{def}}{=} \sum_{q \in V_q} \mu(\text{top}_k[q|G], \delta(q)) \quad (3)$$

For readability, we may omit $\delta(q)$ and δ from the term $\mu(\text{top}_k[q|G], \delta(q))$ and $\mu_k(G, \delta)$, respectively, when δ is clear from the context.

Example 4.6. Consider the following desideratum δ for our running example:

- $\delta(\text{lotus notes download}) = \delta(\text{email client issi}) = \{d_1\}$
- $\delta(\text{spreadsheets download}) = \{d_2\}$

The reader can verify that $\text{top}_1[q|G] = (d_1)$ for all three queries q in V_q. Thus, for each of the functions μ of Equations (1) and (2) we get $\mu_1(G) = 1 + 1 + 0 = 2$.

[4]In principle, μ can represent a utility function that is not necessarily a vanilla quality measure, such as ad benefits [18].

Algorithm G-Greedy$((R, G), \delta, k)$

1: $S \leftarrow \emptyset$
2: $\Delta \leftarrow 1$
3: **while** $\Delta > 0$ **do**
4: $\quad r \leftarrow \text{argmax}_{r \in R \setminus S} \left(\mu_k(G_{S \cup \{r\}}) - \mu_k(G_S)\right)$
5: $\quad \Delta \leftarrow \mu_k(G_{S \cup \{r\}}) - \mu_k(G_S)$
6: $\quad$ **if** $\Delta > 0$ **then**
7: $\quad\quad S \leftarrow S \cup \{r\}$
8: **return** S

Figure 6: The globally greedy algorithm

Now, consider the graph $G_{R'}$ of Figure 5 (discussed in Example 4.5). There we have $\text{top}_1[q|G] = (d_1)$ for $q =$ lotus notes download and $q =$ email client issi, but $\text{top}_1[q|G] = (d_2)$ for $q =$ spreadsheets download. In particular, $\mu_1(G_{R'}) = 3$ for the two aforementioned functions μ. □

Abstract rule optimization is the following problem. We are given an administration setting (R, G), a desideratum δ, and a natural number k. The goal is to find a subset S of R that maximizes $\mu_k(G_S)$; that is, S is such that $\mu_k(G_S) \geq \mu_k(G_{R'})$ for every subset R' of R. Such S is an *optimal solution*. A weaker goal is to find an *α-approximate optimal solution*, where $\alpha \geq 1$ is either a number or a numeric function of the input; such a solution is a set $S \subseteq R$ that satisfies $\alpha \cdot \mu_k(G_S) \geq \mu_k(G_{R'})$ for all $R' \subseteq R$.

EXAMPLE 4.7. Suppose that μ is one of the functions of Equations (1) and (2), and consider again the administration setting (R, G) of our running example (Figure 4). Suppose that the input for abstract rule optimization contains, in addition to (R, G), the desideratum δ of Example 4.6 and $k = 1$. In Example 4.6 we discussed the graph $G_{R'}$ of Figure 5 (where $R' = \{r_2, r_3, r_4\}$). As $\mu_1(G_{R'}) = 3$ is clearly optimal, R' is an optimal solution in this case. □

To simplify the discussion on computational complexity, we assume that μ is fixed (i.e., not part of the input), and that $\mu(\text{top}_k[q|G])$ is computable in polynomial time. Next, we show that abstract rule optimization is hard to approximate. We make the following assumption on the quality measure μ: there is a positive constant c, such that for all queries q, if $\delta(q)$ consists of *exactly one* document then $\mu(\text{top}_1[q|G]) = c$ if $\text{top}_1[q|G] = \delta(q)$, and $\mu(\text{top}_1[q|G]) = 0$ otherwise (i.e., if $\text{top}_1[q|G]$ does not contain the single document in $\delta(q)$). Note that this assumption holds in standard measures that are parameterized by a restriction to the top-k results (assuming that different results are not weighted by different levels of relevancy), such as DCG_k, normalized DCG_k, and precision at k. We say that a function with this property is *reasonable at one*. The following theorem states that abstract rule optimization is extremely hard (to even approximate), no matter which quality measure μ is used, as long as μ is reasonable at one. The proof is in the appendix.

THEOREM 4.8. *Whenever μ is reasonable at one, abstract rule optimization is NP-hard to approximate by any constant factor, or even by $|V_q|^{1-\epsilon}$ for every $\epsilon > 0$.*

Algorithm L-Greedy$((R, G), \delta, k)$

1: $S \leftarrow \emptyset$
2: $T \leftarrow \{(q, d) \in V_q \times V_d \mid d \in \delta(q)\}$
3: **for all** $(q, d) \in T$ **do**
4: $\quad r \leftarrow \text{argmax}_{r \in rel_k(q,d)} \left(\mu_k(G_{S \cup \{r\}}) - \mu_k(G_S)\right)$
5: $\quad$ **if** $\mu_k(G_{S \cup \{r\}}) > \mu_k(G_S)$ **then**
6: $\quad\quad S \leftarrow S \cup \{r\}$
7: **return** S

Figure 7: The locally greedy algorithm

Due to the inherent hardness of our problem, in the following section we present heuristic approaches (that are explored experimentally in Section 5).

4.3 Greedy Heuristics

In this section, we devise two simple heuristic algorithms, each following a greedy approach. We call the first algorithm *globally greedy* and the second *locally greedy*. The reasons for the algorithm names will be clear from their descriptions. Later, we discuss the optimization of these algorithms.

4.3.1 The Globally Greedy Algorithm

The globally greedy algorithm, depicted in Figure 6 under the name G-Greedy, applies the following very simple approach. It initializes an empty solution $S \subseteq R$ (line 1), and iteratively adds to S the best missing rule r (found in line 5). The *best* rule r is such that the difference Δ between the quality obtained by adding r, namely $\mu_k(G_{S \cup \{r\}})$, and the current quality, namely $\mu_k(G_S)$, is maximal. The algorithm terminates (and returns S) once Δ is non-positive (hence, no improving rule can be found).

In our experiments we found that the globally greedy algorithm performs very well in terms of the quality of the returned solution S. However, this algorithm is extremely slow, due to its inherent cubic time: the loop of line 3 can take up to $|R|$ iterations, and then line 4 (finding r) entails traversing over all $r \in R \setminus S$ and computing $\mu_k(G_{S \cup \{r\}})$, which requires computing $\text{top}_k[q|G]$ (and summing up the $\mu(\text{top}_k[q|G])$) for all queries $q \in V_q$. Later, we discuss optimizations of this algorithm. But even the optimized version of this algorithm hardly scales up. Therefore, we consider the next, lower complexity algorithm.

4.3.2 The Locally Greedy Algorithm

The locally greedy algorithm, L-Greedy, is depicted in Figure 7. Similarly to the globally greedy algorithm, the locally greedy one initializes an empty solution S (line 1) and incrementally adds rules (lines 2–6). The main difference between the algorithms is that in the main loop of the locally greedy one, we search for the rule r (to add to S) in a practically tiny subset of the set R of rules (rather than in all of R).

More specifically, line 2 constructs the set T of all *tasks*, where a task is a pair (q, d) such that q is a query (in V_q) and d is a desired document for q (i.e., $q \in \delta(d)$). Then, the main loop (line 3) traverses all tasks in an arbitrary order. For a considered task (q, d), we define $rel_k(q, d)$ to be the set of all the rules r that are relevant to the task, that is: *(1)* r is on a path from q to d, or more formally, for some $q' \in V_r$

we have $e = (q, q') \in E_{\text{qr}}$, $r \in R(e)$, and $(q', d) \in E_{\text{rd}}$, and *(2)* taken alone r can push d to the top-k results of q, or more formally, $d \in \text{top}_k[q|G_{\{r\}}]$. In line 4, a rule r that maximizes $\mu_k(G_{S\cup\{r\}}) - \mu_k(G_S)$ is added to S, provided that this maximum value is positive.

4.3.3 Optimization

In both the globally greedy and the locally greedy algorithms, a significant portion of the computation takes place on computing the quality of the system on intermediate sets of rules (that is, the computations of $\mu_k(G_S)$ and $\mu_k(G_{S\cup\{r\}})$ in line 4 of Figure 6 and line 4 of Figure 7). This computation is done to obtain the difference $\mu_k(G_{S\cup\{r\}}) - \mu_k(G_S)$, where S is the current set of rules and r is a considered candidate rule. Let $\Delta(S, r)$ denote that difference. We can optimize the computation by observing that $\Delta(S, r)$ is affected by only a few of the queries, namely those on which r fires.

Formally, consider a query q, a rule r and a set $S \subseteq R$. Define $\Delta(q, S, r) = \mu(\text{top}_k[q|G_{S\cup\{r\}}]) - \mu(\text{top}_k[q|G_S])$. From (3) we get that $\Delta(S, r) = \sum_{q\in V_{\text{q}}} \Delta(q, S, r)$. Denote by $V_{\text{q}}(r)$ the set of queries $q \in V_{\text{q}}$, such that $r \in R(e)$ for some edge $e \in E_{\text{qr}}$ emanating from q. An easy observation is that $\Delta(q, S, r) = 0$ whenever $q \notin V_{\text{q}}(r)$, and therefore $\Delta(S, r) = \sum_{q\in V_{\text{q}}(r)} \Delta(q, S, r)$.

The optimization is as follows. During the computation we maintain each $\mu(\text{top}_k[q|G_S])$ for the current S. (So, at the beginning we compute $\mu(\text{top}_k[q|G_\emptyset])$.) When a rule r is considered, we iterate over the queries in $V_{\text{q}}(r)$ and compute $\mu(\text{top}_k[q|G_{S\cup\{r\}}])$, thus obtaining $\Delta(q, S, r)$. We sum up all these $\Delta(q, S, r)$ into $\Delta(S, r)$ to be used in the next steps. Hereafter, the optimized versions of G-Greedy and L-Greedy are denoted by G-Greedy-opt and L-Greedy-opt, respectively.

4.4 Weighted Queries

In realistic scenarios, queries may carry different levels of importance, which may be expressed by means of query weights. For instance, this weight can be the frequency in which the query is posed. In the experimental study of the next section, we use as weights frequencies derived from query logs. The abstraction presented in this section can incorporate weighted queries in a straightforward manner. In particular, the abstract model assigns a weight $w(q)$ to each query $q \in V_{\text{q}}$, and in the definition of $\mu_k(G)$ (see (3)), each addend $\mu(\text{top}_k[q|G])$ is multiplied by $w(q)$. The greedy algorithms automatically adjust to weights by using the weighted μ_k. An intuitive traversal order on T in the locally greedy algorithm (Figure 7) is by decreasing $w(q)$; we indeed apply this order in the experiments of the following section.

5. EXPERIMENTAL EVALUATION

In this section, we describe an empirical evaluation of our proposed solutions over the IBM intranet search engine, with datasets provided by the company's search administrators.

More specifically, our dataset was obtained from a list of 1894 suggested matches provided by IBM CIO Office, where a *suggested match* consists of a query q and a document d desired as a top match for q. Generally, the queries involved in the suggested matches are frequent, and hence are recorded for quality maintenance. Note that a query can be matched against multiple documents, and a document may be suggested for multiple queries. As mentioned in Section 3, by analyzing the high-quality fields (e.g., titles and URLs) of desired documents, we generated for each suggested match a set of query-rewrite rules, each of which, in the absence of any other rule, is capable of pushing the desired document up to the top-5 results for the given query. This produced a total of 11907 rules. Our methodology for extracting input from these suggested matches and rules will be described in the following sections. Our experiments ran on a Linux SUSE (64-bit) server with eight 4-core Intel Xeon (2.13GHz) processors and 48GB of memory. The algorithms were implemented in Java 1.6 and ran with 12GB allocated memory.

5.1 Recognizing Natural Rules

We first evaluated the classification models of Section 3 for recognizing natural rules. Among all generated rules, we randomly selected and manually labeled 1187 rules as either natural or unnatural. Such labeling is subject to human judgment; moreover, labeling often required domain-specific knowledge, and for that we needed to inspect the pages relevant to the terms involved in the rule. Using this labeled dataset, we evaluated the accuracy of the classifiers SVM and rDTLC. The results reported in this section were consistently obtained by performing a 5-folder cross validation.

We also explored the question of how the classifiers perform on rules for more *popular* suggested matches. We analyzed the query logs of four months of intranet search at IBM and estimated the popularity of each suggested match by counting the number of sessions that have a click on the suggested document for the corresponding query. Therefore, each quality measure has two versions—the *unweighted* version is denoted by uw and the *weighted* version is denoted by w. In addition, we retrained the two classifiers by incorporating the weights on rules, thereby obtained a third version denoted by wt (*weighted training*). Whether or not to train the classifier with weights is a matter of a choice that depends on the specific scenario, and for that reason we evaluated both the w and the wt versions.

Figure 8 reports the accuracy (ratio of correctly classified rules) for SVM and rDTLC. As shown, when trained without weights rDTLC outperforms SVM by about 10% for both the unweighted and weighted measures. When the classifiers are trained with weights, rDTLC has a smaller improvement (about 3%) and both classifiers achieve high accuracy.

Even with classification, the number of natural rules may still be large. An administrator may be interested in only the top-k rule suggestions for some k. We explored the naive strategy of ranking the top-k candidate rules by decreasing confidence of the classifier at hand. We leave for future work the exploration of more sophisticated *learning to rank* strategies for this task. Nevertheless, we observe that this naive approach is already beneficial compared to a random selection of k candidates (a strategy denoted as *RND*). Specifically, Figure 9 gives the Mean Reciprocal Rank (MRR) of the different rankers. Figures 10 and 11 give the normalized Discounted Cumulative Gain at the top-k results (nDCG_k) where $k = 1, 3, 5$ for the unweighted and weighted cases, respectively. We can see that when the classifiers are not retrained with weights, rDTLC performs significantly better than SVM (and RND), but its advantage over SVM disappears when they are retrained with weights; in particular, when trained with weights both are almost always capable of suggesting natural rules for all $k = 1, 3, 5$.

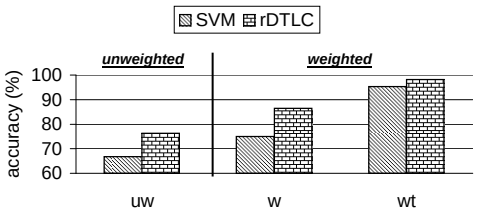

Figure 8: Accuracy of rule classification

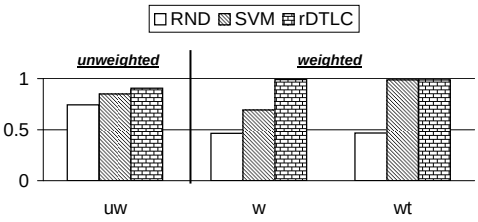

Figure 9: MRR for rule ranking

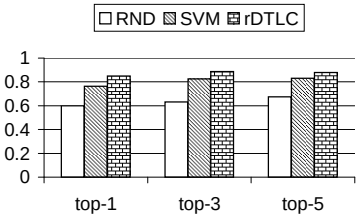

Figure 10: nDCG$_k$ (unweighted) for rule ranking

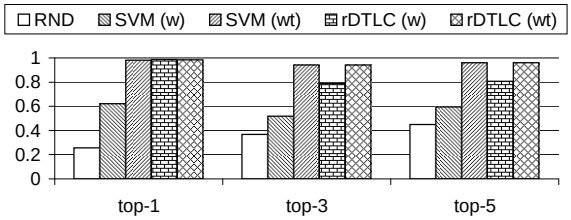

Figure 11: nDCG$_k$ (weighted) for rule ranking

5.2 Optimizing Multi-Rule Selection

We now present an experimental study of our solutions for the abstract rule optimization problem (Section 4).

Datasets. We constructed several datasets from our collection of suggested matches. Each dataset is an administration setting constructed by selecting a subset M of the suggested matches, and a subset R of the rules automatically created for M. (Observe that M implies a desideratum δ.) Later, we will explain how exactly M and R are selected for each dataset. Once M and R are selected, the administration graph G is constructed as follows. The queries in M form the set V_{q} of queries. The set V_{r} of r-queries is obtained by applying the rules in R to the queries in V_{q}. The set V_{d} of documents contains all the documents in M, as well as the top-5 results for each query and r-query in $V_{\mathrm{q}} \cup V_{\mathrm{r}}$, as obtained by invoking our search engine without rewrite rules. The edges of G are defined in the obvious manner.

Algorithms. We compared the G-Greedy and L-Greedy algorithms and their optimized versions G-Greedy-opt and L-Greedy-opt, as described in Section 4.3. We also considered several baseline algorithms: G-Random, L-Random and AllRules. G-Random and L-Random are similar to their greedy counterparts, except that their selection of rules is random. More precisely, G-Random selects a random subset of rules among all the rules, and L-Random randomly selects an effective rule for each suggested match. The algorithm AllRules simply selects all the rules.[5]

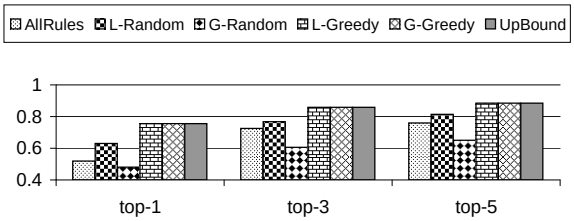

Figure 12: nDCG$_k$ (unweighted) for the labeled dataset

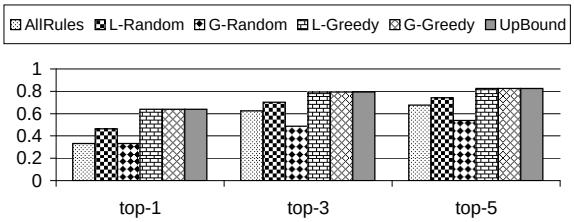

Figure 13: nDCG$_k$ (weighted) for the labeled dataset

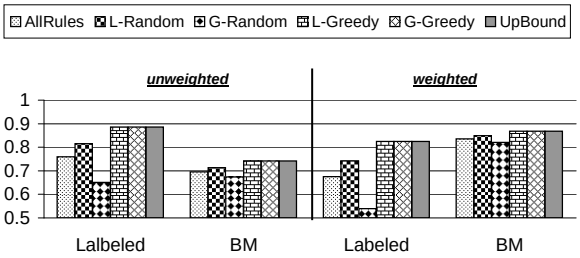

Figure 14: MRR for the labeled dataset

Measures. We consider two quality measures μ: the normalized Discounted Cumulative Gain at the top-k results (nDCG$_k$), and the Mean Reciprocal Rank (MRR). Since our administration graph is restricted to the top-5 matches for each query and r-query, MRR is restricted to the top-5 results as well (as it would not make sense otherwise).

To estimate the gap between the solution provided by each of the algorithms and the optimal solution, we computed an upper bound on the optimum for each quality measure μ, as follows. We evaluated the sum of Equation (3), where each addend $\mu(\mathrm{top}_k[q|G], \delta(q))$ is computed as if each suggested match is ranked as high as possible using a rule from the our collection. So, in principle, we allow this bound to place two documents in the same rank. Note that this number is not smaller than, and may be strictly larger than, the optimum (namely $\mu_k(G_S)$ for an optimal solution S).

Weights. We used both unweighted and weighted quality measures. In the weighted version, each query is weighted by the number of sessions where it is posed, as recorded in the aforementioned four months of query logs. For the unweighted version, the algorithms L-Greedy, L-Greedy-opt and L-Random iterate over the suggested matches (which

[5]Albeit straightforward, AllRules represents a common practice of our search administrators.

Table 2: Running time

μ = MRR, $k = 5$ (ms)			
Alg.	Labeled	BM	Extended
G-Greedy	25193	82566	> 24 hours
G-Greedy-opt	2911	7719	> 24 hours
L-Greedy	404	3517	1648137
L-Greedy-opt	39	110	87484

μ = nDCG, $k = 1$ (ms)			
Alg.	Labeled	BM	Extended
G-Greedy	26356	82595	> 24 hours
G-Greedy-opt	2638	6896	> 24 hours
L-Greedy	399	3462	3522894
L-Greedy-opt	38	112	124613

form the set T in Figure 7) in a random order obtained by applying a random permutation thereof. For the weighted version, iteration over the suggested matches is ordered by decreasing weight of the involved queries.

5.2.1 Evaluation on the Labeled Dataset

In the first set of experiments, we used the *labeled dataset* obtained by taking as M the suggested matches that were used for labeling rewrite rules (see Section 5.1), and as R the set of rules that are manually labeled as natural. The resulting administration graph contains 135 queries, 300 r-queries, 423 documents, and a total of 1488 edges. Note that only a small portion of the suggested matches were used for labeling; the others, for which the rules are not labeled, will be used later in Section 5.2.2.

We first examined the quality of solutions produced by each of the algorithms. Figures 12 and 13 report the nDCG$_k$, with $k = 1, 3, 5$, for the unweighted and weighted cases, respectively. Figure 14 reports the MRR for both unweighted and weighted cases. (The groups labeled with BM will be discussed later.) Observe that on all quality measures, L-Greedy and G-Greedy score significantly higher than the other alternatives. In fact, they already reach the upper bound, and hence provide optimal solutions.

The goal of the next experiment is to explore the solutions in the presence of suggested matches that hit the top search results without requiring rules (yet, they can be affected by introducing rules). This setting represents a typical scenario where a search administrator needs to address problematic queries without compromising the overall search quality. For that, we enhanced the set M, from the construction of the labeled dataset, with 373 such suggested matches that are used as a *benchmark*. The groups of bars labeled with BM in Figure 14 report the MRR for both the unweighted and the weighted cases. Note that the greedy algorithms get better scores than the other alternatives, and again reach the upper bound. But now, the other alternatives get high scores as well, since many suggested matches in the benchmark are not affected by the rules in R. The results for nDCG$_k$ show a similar picture, and are therefore omitted.

Next, we compare the execution cost of the algorithms. AllRules, G-Random and L-Random entail a negligible running time (less than 0.2 ms). We focus on the globally and locally greedy algorithms, and examine the contribution of the optimization. Table 2 summarizes the running times for two different combinations of μ and k. The columns entitled "Labeled" and "BM" refer to the labeled dataset and the one enhanced with the benchmark, respectively. (The columns entitled "Extended" will be discussed later.) Observe that the locally greedy algorithms are over one order of magnitude faster than their globally greedy counterparts. In addition, the optimized versions are generally over one order of magnitude faster than their unoptimized counterparts. In particular, the optimized version of our locally greedy algorithm is capable of finding an optimal solution in real time for the typical usage scenarios.

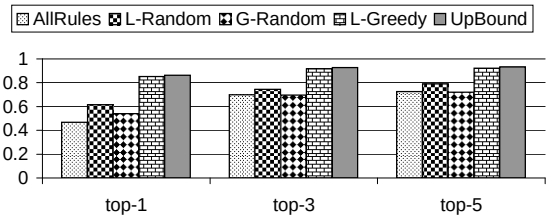

Figure 15: nDCG$_k$ (unweighted) for the extended dataset

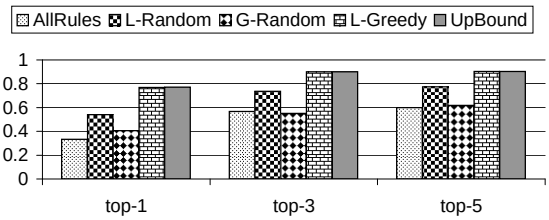

Figure 16: nDCG$_k$ (weighted) for the extended dataset

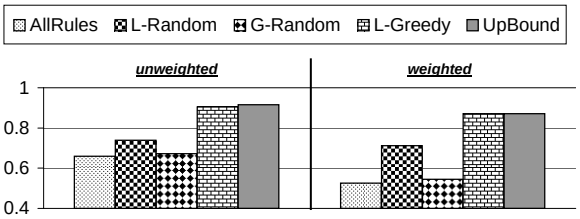

Figure 17: MRR for the extended dataset

5.2.2 Evaluation on the Extended Dataset

The results of the previous experiments demonstrate the effectiveness and feasibility of the greedy algorithms. To further evaluate the scalability of the greedy algorithms, we constructed a larger *extended dataset*. To create this dataset, we used as M the set of *all* the suggested matches, and as R the set of *all* the rules automatically generated for them (including those that are not labeled). The resulting administration graph contains 1001 queries, 10990 r-queries, 4188 documents, and a total of 36986 edges.

Again, the running times for two different combinations of μ and k are shown Table 2, now by the columns entitled "Extended." The globally greedy algorithms (including the optimized one) do not scale to the extended dataset. For the locally greedy algorithms, the improvement achieved by the optimization is again by over an order of magnitude.

Finally, we evaluated the quality of the solutions over the extended dataset. Figures 15 and 16 report the nDCG$_k$, with $k = 1, 3, 5$, for the unweighted and weighted cases, respectively. Figure 17 reports the MRR for both unweighted

and weighted cases. Consistently with the previous experiments, L-Greedy outperforms the other alternatives. The gap between L-Greedy and the upper bound is generally around one percent, and is barely notable in the figure. These results demonstrate that even in the extreme case where the administrator adopts all candidate suggestions, L-Greedy-opt can find a practically optimal solution within a reasonable amount of time (around 2 minutes).

6. CONCLUDING REMARKS

The rule-based architecture aims to provide search administrators with the means of adapting the search engine to the content and dynamics of the enterprise. In this paper, we explored the incorporation of automation in the manual practice of administrators. Specifically, we studied the problem of suggesting natural rewrite rules, and proposed corresponding machine-learned classifiers for rules. We also studied the problem of selecting rules, from a given collection, with the goal of optimizing the quality on a benchmark. We presented a theoretical model that captures this task as a combinatorial optimization problem, analyzed its theoretical complexity, and proposed heuristic algorithms to accommodate its hardness. Experiments on a real enterprise case (IBM intranet search) indicate that the proposed solutions are effective and feasible.

While the testbed for this work has been the IBM internal search, the two challenges we studied hold in essentially every search system that supports administration by rewrite rules. Moreover, the setting and solutions we proposed are at a level of abstraction that hardly ties them to our specific testbed. In future work, we plan to focus on extending our techniques to handle significantly more expressive rules.

7. 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*, pages 1–10, 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*, pages 346–352, 2011.

[3] R. A. Baeza-Yates, C. A. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. In *EDBT Workshops*, volume 3268 of *Lecture Notes in Computer Science*, pages 588–596, 2004.

[4] B. Billerbeck, F. Scholer, H. E. Williams, and J. Zobel. Query expansion using associated queries. In *CIKM*, pages 2–9, 2003.

[5] A. Z. Broder and A. C. Ciccolo. Towards the next generation of enterprise search technology. *IBM Systems Journal*, 43(3):451–454, 2004.

[6] N. Craswell, D. Hawking, and S. E. Robertson. Effective site finding using link anchor information. In *SIGIR*, pages 250–257, 2001.

[7] P. A. Dmitriev, N. Eiron, M. Fontoura, and E. J. Shekita. Using annotations in enterprise search. In *WWW*, pages 811–817, 2006.

[8] R. Fagin, B. Kimelfeld, Y. Li, S. Raghavan, and S. Vaithyanathan. Understanding queries in a search database system. In *PODS*, pages 273–284, 2010.

[9] R. Fagin, B. Kimelfeld, Y. Li, S. Raghavan, and S. Vaithyanathan. Rewrite rules for search database systems. In *PODS*, pages 271–282, 2011.

[10] R. Fagin, R. Kumar, K. S. McCurley, J. Novak, D. Sivakumar, J. A. Tomlin, and D. P. Williamson. Searching the workplace web. In *WWW*, pages 366–375, 2003.

[11] D. Hawking. Challenges in enterprise search. In *ADC*, volume 27 of *CRPIT*, pages 15–24, 2004.

[12] D. Hawking. Enterprise search - the new frontier? In *ECIR*, volume 3936 of *Lecture Notes in Computer Science*, page 12, 2006.

[13] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *WWW*, pages 387–396, 2006.

[14] R. Kraft and J. Y. Zien. Mining anchor text for query refinement. In *WWW*, pages 666–674, 2004.

[15] Y. Li, R. Krishnamurthy, S. Vaithyanathan, and H. V. Jagadish. Getting work done on the web: supporting transactional queries. In *SIGIR*, pages 557–564, 2006.

[16] W.-Y. Loh and Y.-S. Shih. Split selection methods for classification trees. *Statistica Sinica*, 7:815–840, 1997.

[17] W.-Y. Loh and N. Vanichsetakul. Tree-structured classification via generalized discriminant analysis. *Journal of the American Statistical Association*, 83:715–728, 1988.

[18] A. Malekian, C.-C. Chang, R. Kumar, and G. Wang. Optimizing query rewrites for keyword-based advertising. In *EC*, pages 10–19, 2008.

[19] U. Ozertem, E. Velipasaoglu, and L. Lai. Suggestion set utility maximization using session logs. In *CIKM*, pages 105–114, 2011.

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

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

[22] Z. Zhang and O. Nasraoui. Mining search engine query logs for query recommendation. In *WWW*, pages 1039–1040, 2006.

[23] H. Zhu, S. Raghavan, S. Vaithyanathan, and A. Löser. Navigating the intranet with high precision. In *WWW*, pages 491–500, 2007.

[24] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007.

APPENDIX

Proof of Theorem 4.8. We show a reduction from the *maximum clique* problem: given an undirected graph H, find a clique of a maximum size, where a *clique* is a set C of nodes such that every two members are adjacent. Zuckerman [24] showed that for all $\epsilon > 0$, it is NP-hard to approximate this problem to within $n^{1-\epsilon}$, where n is the number of nodes.

Given an undirected graph H, we construct the administration setting (R, G) as follows. For each node v of H, let $N(v)$ denote the set of neighbors of v, and we define two unique words a_v and b_v. For each such v, R contains the rule $a_v \rightarrow b_v$, G has the query q_v that consists of a_v followed (in some order) by all the a_u where $u \notin N(v)$, and G has two unique documents d_v^{gd} and d_v^{bd}. The r-queries of G are all those obtained by applying a rule in R to a query of G. Let q' be an r-query of G. If q' begins with a_v, then G has the r-query matching (q', d_v^{bd}) with the weight 2; and if q' begins with b_v, then G has (q', d_v^{gd}) with the weight 1. There are no query matchings in G. Finally, the desideratum δ maps each q_v (where v is a node of H) to the singleton $\delta(q_v) = \{d_v^{\text{gd}}\}$.

Observe the following for a subset R' of R and a node v of H. If R' contains a rule $a_u \rightarrow b_u$ for some node $u \notin N(v)$, then q_v is reformulated into an r-query that begins with a_v and thus matches d_v^{bd} with the weight 2, implying $\text{top}_1[q_v|G_{R'}] = (d_v^{\text{bd}})$. If R' contains $a_v \rightarrow b_v$ but no rule $a_u \rightarrow b_u$ where $u \notin N(v)$, then q_v is reformulated only into an r-query that begins with b_v and thus matches d_v^{gd} with the weight 1, implying $\text{top}_1[q_v|G_{R'}] = (d_v^{\text{bd}})$. If R' contains none of $a_v \rightarrow b_v$ and $a_u \rightarrow b_u$ ($u \notin N(v)$), then $\text{top}_1[q_v|G_{R'}] = \emptyset$. From this observation, the correctness of the reduction is proved straightforwardly: for each subset $R' \subseteq R$, if $\mu_1(G_{R'}) = c \cdot m$ (where c is taken from the definition of *reasonable at one*), then we can obtain from $G_{R'}$ (in polynomial time) a clique of size m in H; on the reverse direction, from each clique of size m in H we can construct a subset $R' \subseteq R$ with $\mu_1(G_{R'}) = c \cdot m$. $\square$