Efficient Query Rewrite for Structured Web Queries

Sreenivas Gollapudi Samuel Ieong Alexandros Ntoulas Stelios Paparizos
Microsoft Research, Silicon Valley
{sreenig, saieong, antoulas, steliosp}@microsoft.com

ABSTRACT

Web search engines incorporate results from structured data sources to answer semantically rich user queries, i.e. '`Samsung 50 inch led tv`' can be answered from a table of television data. However, users are not domain experts and quite often enter values that do not match precisely the underlying data, so a literal execution will return zero results. A search engine would prefer to return at least a minimum number of results as close to the original query as possible while providing a time-bound execution guarantee. In this paper, we formalize these requirements, show the problem is NP-Hard and present approximation algorithms that produce rewrites that work in practice. We empirically validate our algorithms on large-scale data from a major search engine.

1. INTRODUCTION

Web users commonly look for results beyond the traditional web pages and often their need is covered via the use of semantic information that comes from (semi-)structured data sources. For example, consider queries such as electronic goods (e.g. '`50 inch samsung led tv`'), fashion (e.g. '`$1600 prada handbags`'), or movie-showtimes listings (e.g. '`avatar showtimes near san francisco`').

A major challenge in using structured data to answer keyword queries is that users often lack domain expertise and may pose queries that lead to very few or no results. For example, consider the query '`50 inch samsung led tv`'. There exists work in the literature [9] that can correctly classify and semantically interpret the query to attribute-value pairs that correspond to underlying structured attributes. So the query can be thought as (`50 inch` ⇒ `display size`, `Samsung` ⇒ `Brand`, `led tv` ⇒ `display type`). However, if the query is directly evaluated as specified, there will be no results that can satisfy the interpretation as Samsung does not make 50-inch LED TVs. On the other hand, Samsung makes *46-inch* and *55-inch LED* TVs and 50-inch *PLASMA* TVs. Arguably, the users would prefer to see such results that are close to their original query instead of looking at an empty page with no results because they did not know the appropriate precise values when typing the query.

We are interested in rewriting the queries through semantic term expansion. For example, the above query may be rewritten as '`(46 to 55 inch) (samsung or sony) (led or plasma) tv`'. The quality of the rewrites depends on two factors. First, the rewritten query should preserve the meaning of the original query as closely as possible. We measure the fidelity of the rewrite by computing how far away the set of results retrieved are to the original query, as estimated by user preferences learned through click logs. Second, the rewritten query should ensure there are sufficiently many results returned to the user. We measure the coverage of the rewrite by counting how many times when a certain number of results is requested, the expectation is met.

If efficiency had not been an issue, a candidate solution would have been to expand the terms a little at a time, issue the rewritten query to the index, and repeat as necessary until the minimum number of results requested is retrieved. This solution would not be applicable to web search, however, as users expect results to be returned in half a second or less, thus placing a strict performance requirement on the query rewriting component. Hence, many search engines place a restriction on the number of re-written queries that can be issued to the index as part of the original query execution. To ensure this requirement is met, we require that the techniques may only use precomputed statistics of the index but may not access the index at run time more than once to execute the final query. Similarly, we also require that the techniques can take an input parameter that limits the number of alternative rewrites they can examine.

In this paper, we formulate the above problem as time-bound query rewriting for structured web queries. As part of our contributions we formally describe an optimization framework that takes as input a candidate query q, a desired number of results k, and a parameter T that governs how many rewrites can be considered, and produces a rewritten query that aims to retrieve at least k results as close to the original query as possible. We show that finding the optimal solution to this problem is NP-Hard and introduce a greedy and a dynamic programming solution that rewrite the query in a principled and controlled fashion. We evaluate the proposed solution using real queries from a commercial search engine's shopping vertical against a prototype commerce search engine.

2. RELATED WORK

Structured data is abundant on the web, and there have been studies on how to retrieve them in a manner suitable to web search [3, 4]. Fontoura *et al.* proposed a method to relax text queries using taxonomies [5]. Finally, given a query and a distance function, one can think of the problem we are trying to solve as a nearest neighbor problem. Nearest neighbor problems have been studied in the past, for example [8]. Similar to the k-nearest neighbors, but from a join relaxation problem in databases is the work described

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.

in [6]. We find such work very valuable in relaxing the user query and finding good quality results within a reasonable distance around what was specified in the query. However, it is not clear how these techniques will perform under strict time constraints like ours.

3. PROBLEM FORMULATION

We first describe a model of structured web queries, and assumptions on how they are parsed, and how items are evaluated with respect to the parsed queries. We then formally define the problem of time bound query rewrites.

3.1 Model

Given a keyword web query, we assume the existence of a semantic parser that identifies the attributes requested in the query and extracts their associated desired values, based on past work such as [9]. For example, the query '`50 inch samsung led tv`' is parsed as a *structured query* {*table:TV, brand:Samsung, type:LED, diagonal:50*}. Denote a generic parsed query by its attribute-value pairs, $q = \{a_1 : v_1, a_2 : v_2, \ldots, a_m : v_m\}$. Denote the value of attribute a_i in query q by q_{a_i}. For our example query, q_{brand} = Samsung. Consistent with the interpretation of web queries as conjunctions of keywords, we interpret the structured query under the AND-semantics as well.

Let P be a database of items, from which we retrieve results to serve the query. For each item $p \in P$, we represent it as a set of attribute-value pairs $\{a_1 : w_1, a_2 : w_2, \ldots, a_n : w_n\}$, and the value of attribute a_i by p_{a_i}. We assume that the semantic parser will only identify attributes for which we have data, hence the query specifies the values of a subset of these n attributes. Henceforth, given a query, we only focus on the m attributes mentioned.

As discussed in the Introduction, users may lack domain expertise and may be unfamiliar with the attribute specification of the underlying structured data. As a result, they may issue queries for which there exists no match in the database. Nonetheless, it is desirable that a search engine should return results that are close to the query. Our objective is to develop a formal framework that allows one to investigate how best to rewrite the user query in order to retrieve good results for the user efficiently.

To make the discussion formal, denote the domain of attribute a_i by A_i. Let the function $d_i : A_i \times A_i \to [0, 1]$, $d_i(v, w)$ measures the distance of attribute value w from attribute value v. When $d_i(v, w)$ is small, it means that attribute value w is similar to attribute value v. We note that our solution does not depend on assumptions such as symmetry or triangle-inequality about the distance function.

An aggregate distance function $ad : P \times Q \to \mathbb{R}$, $ad(p, q)$ measures how well item p matches query q. When $ad(p, q)$ is small, it means item p matches the query q well. We assume that the function depends only on the attribute values of the item and the query. We next define a basic yet fundamental property of aggregate distance functions that we assume throughout the paper.

DEFINITION 1 (MONOTONICITY). *An aggregate distance function, $ad(\cdot)$, is* monotonic *if for any query $q = \{a_1 : v_1, a_2 : v_2, \ldots, a_m : v_m\}$, any two items p^1 and p^2, if*

$$\forall 1 \le i \le m, \qquad d_i(v_i, p^1_{a_i}) \ge d_i(v_i, p^2_{a_i}) \ ,$$

then $ad(p^1, q) \ge ad(p^2, q)$.

Monotonicity ensures that an item closer to the query in each of the attributes will also be closer to the query in aggregate distance. This is a natural property that should be satisfied when the attribute distances determine how well an item matches a query. Example aggregate distance functions that satisfy monotonicity includes weighted sums of the attribute distances, and ℓ_p-norms that treat attribute distances as vectors in m-dimensional space.

3.2 Query Rewrite Formulation

Given a query q, our ultimate goal is to find the top-k items that match the query within a fixed time window. In order to find the top-k items, we must first be able to select at least k items from the database. This may not be possible when there are less than k items that match all the desired attribute values. The focus of our work is on how to rewrite the query in a principled way so as to ensure sufficiently many items are returned, keeping fidelity to the original query, while respecting the time constraints imposed on the individual components of a search engine.

For a given attribute-value pair $a_i : v$ and value $\delta \in [0, 1]$, let $B_i(v, \delta)$ be the set of attribute values that is δ-close to v, i.e.,

$$B_i(v, \delta) = \{w \in A_i | d_i(v, w) \le \delta\} \ . \tag{1}$$

In other words, $B_i(v, 0)$ are the set of attribute values that are equivalent to v, whereas $B_i(v, 1)$ are the set of all attribute values.

Denote a *relaxed query* q by $\{a_1 : v_1 \pm \delta_1, a_2 : v_2 \pm \delta_2, \ldots, a_m : v_m \pm \delta_m\}$. An item p matches q if and only if

$$\forall 1 \le i \le m, \qquad p_{a_i} \in B_i(v_i, \delta_i) \ .$$

At a high level, the query rewrite for structured web query problem is to take an input query and find a relaxed query that will result in at least k matches in the database. If time had not been an issue, a simple solution would be to iteratively make small relaxation to the query, issue it to the database to find out the number of matches, and repeat until we have found k results. However, due to the performance requirement imposed by web search, this approach is infeasible as database access is costly. Indeed, an algorithm may only be able to carry out a small amount of computations within the time envelope.

To capture these limitations, we propose to bound the time of any solution by the *number of different relaxed queries* it considers and include this as an explicit parameter to the problem specification. To ensure that this meaningfully reflects the performance requirement and is helpful in differentiating among solutions, we require that the amount of time it takes to evaluate each relaxed query to be constant. Note that different forms of evaluating the relaxation will lead to different classes of problems.

In this paper, we focus on evaluating the relaxed query using database histograms. Histograms are one of the most common and important statistics of an attribute used in databases. We now describe how they can be used to estimate the number of potential matches to a query without direct database access.

Formally, let the histogram of attribute a_i be h_i, and that for a set of attribute values $V \subseteq A_i$, $h_i(V)$ returns the number of items that have the corresponding attribute value. Assuming the attributes in the query are independent, one can estimate the number of matches to a relaxed query as

follows. For a query $q = \{a_1 : v_1 \pm \delta_1, a_2 : v_2 \pm \delta_2, \ldots, a_m : v_m \pm \delta_m\}$, the estimated number of matches equals

$$\text{EST}(q) = |P| \prod_{i=1}^{m} \frac{h_i(B_i(v_i, \delta_i))}{|P|} \quad . \tag{2}$$

When attributes are dependent, the estimate could be misleading. Functional dependencies may help to improve the estimate. A discussion of how they can help with estimation is beyond the scope of this paper.

Using histograms, we define our time-bound query rewrite problem using histogram statistics as follows.

DEFINITION 2 (QUERY-REWRITE-HISTOGRAMS). *Given (1) A query $q = \{a_1 : v_1, a_2 : v_2, \ldots, a_m : v_m\}$; (2) Database size $|D|$; (3) Histograms h_i for each attribute a_i; (4) The minimum number of items to return, k; (5) The maximum number of relaxed queries considered, T. Find a relaxed query $q' = \{a_1 : v_1 \pm \delta_1, a_2 : v_2 \pm \delta_2, \ldots, a_m : v_m \pm \delta_m\}$ with at most T relaxed queries considered, such that $\text{EST}(q')$ is at least k, and that the total amount of relaxation,*

$$tr(q') = \sum_{i=1}^{m} \delta_i \quad , \tag{3}$$

is minimized.

Note that ideally we would like to minimize the maximum aggregate distances of the set of selected items to a query. Unfortunately, the maximum aggregate distance cannot be estimated using histograms alone. Therefore, we instead focus on bounding the aggregate distance by controlling the total amount of relaxation as defined above.

We end by noting that the problem of QUERY-REWRITE-HISTOGRAMS is closely related to knapsack problems, and is hard to solve optimally.

THEOREM 1. QUERY-REWRITE-HISTOGRAMS *is NP-hard, even when there is no limit on the number of relaxations.*

4. ALGORITHMS FOR QUERY REWRITES

We present two algorithms for QUERY-REWRITE-HISTOGRAMS and discuss their trade-offs.

Greedy-Rewrite. A general template for solving QUERY-REWRITE-HISTOGRAMS is to (1) select an attribute based on some criteria, (2) relax it by a small amount ϵ to get relaxed query q, (3) compute the estimate $\text{EST}(q)$, and (4) repeat as long as $\text{EST}(q) < k$. Different selection criteria give rise to different heuristics.

In GREEDY-REWRITE, we select an attribute to relax based on how constraining the attribute is. Formally, for a relaxed query $q = \{a_1 : v_1 \pm \delta_1, a_2 : v_2 \pm \delta_2, \ldots, a_m : v_m \pm \delta_m\}$, we pick the most constraining attribute, a_i where

$$h_i(B_i(v_i, \delta_i))$$

is the smallest to relax.

If at the end of having evaluated T relaxed queries and none is found to have an estimated number of matches of at least k, the last relaxed query (i.e., the one with the largest amount of relaxation) is returned.

DP-Rewrite. Drawing on ideas similar to the dynamic program for knapsack-style problems, we also consider a dynamic programming heuristic DP-REWRITE. For a query $q = \{a_1 : v_1, a_2 : v_2, \ldots, a_m : v_m\}$, let

$F(j, d)$ = Maximum fraction of products satisfying the relaxed query on attributes $a_1, \ldots, a_j$ with total relaxation $\sum_{i=1}^{j} \delta_i \leq d$.

Let ϵ be a parameter to the heuristic that determines the step size, i.e., by what increment we increase the relaxation of an attribute. For each cell in $F(\cdot, \cdot)$, we need to consider one new relaxation. Therefore, for a given maximum number of relaxations T, we can consider only $\rho = \lfloor \frac{T}{m} \rfloor$ different values for each attribute. Hence, we compute $F(j, d)$ using dynamic programming as described in Algorithm 1.

Algorithm 1 Dynamic program for Query Rewrite Using Histograms, with step size ϵ and $\rho = \lfloor \frac{T}{m} \rfloor$.

```
for d ← 0, ε, 2ε, ..., min(ρε, 1) do
    F(1, d) ← h_1(B_1(v_1, d)) / |P|
end for
for j ← 2 to m do
    for d ← 0, ε, 2ε, ..., min(ρε, j) do
        F(j, d) ← max_{d'=0,ε,...,min(d,1)} ( (h_j(B_j(v_j, d')) / |P|) F(j−1, d−d') )
    end for
end for
```

The optimal solution is given by $\min_{d'} F(m, d')$ for which the value is at least $\frac{k}{|D|}$. The amount of relaxation for each attribute can be kept track of by an auxiliary table.

Similar to GREEDY-REWRITE, if no relaxed query with an estimated number of matches of at least k is found at the end of having evaluated T relaxed queries, the relaxed query with the largest amount of relaxation is returned.

Trade-off Between the Algorithms. There is a trade-off between the algorithms. On the one hand, for any fixed ϵ, if the number of relaxed queries allowed is large, DP-REWRITE is guaranteed to find a relaxed query q with $tr(q)$ no larger than the one found by GREEDY-REWRITE. However, when the number of relaxed queries allowed is small, DP-REWRITE will only be able to investigate solutions of small total amount of relaxation, and may fail to find a solution when GREEDY-REWRITE may succeed.

5. EXPERIMENTAL EVALUATION

In this section, we study the behavior and performance of our algorithms on effectively rewriting real user queries.

5.1 Experimental Setup

For our experimental evaluation we built a prototype search engine and we populated it with real data from the shopping vertical of a commercial search engine. To this end, we downloaded the detailed descriptions for about 5 million products related to 73 categories about electronics (such as Televisions, Equalizers, GPS Receivers, etc.) from [2]. Each product was retrieved in structured form, with its attribute and values clearly specified (e.g. [1]). We indexed the product details and computed the histograms for their attributes.

We randomly sampled $1K$ commerce queries from the logs of a commercial search engine and selected the 343 queries that returned too few results as our test set.

For our performance metric we used the average distance of all items within our result set (MEAN-DIST). The distance is computed as the average of pair-wise attribute distances between a given product and the query. The distances

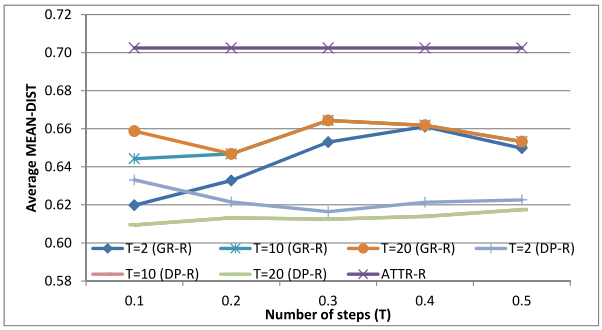

Figure 1: Effect of step size ϵ on avg Mean-Dist

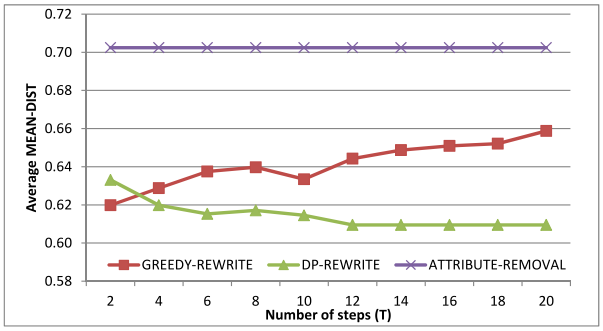

Figure 2: Change in avg Mean-Dist with time steps

between attribute values are estimated from search logs using the technique described in [7]. In general, the lower the MEAN-DIST the better the performance of an algorithm. Finally, we set the number of returned results to 10.

As a baseline, we implemented an approach similar to that of `amazon.com`, named ATTRIBUTE-REMOVAL. This approach rewrites a query that returns few results by directly dropping keywords. In our case, we implemented a version that removes attributes from the structured interpretation of the query. More specifically, we select the attrbute which is the most constraining to be dropped from the query.

5.2 Results

We start our experimental evaluation by studying the effect of the step size ϵ in the performance of our query rewrite algorithms. Both GREEDY-REWRITE and DP-REWRITE use a parameter ϵ that determines the amount of relaxation of an attribute at a step of the algorithm. To study this effect in more detail, we evaluate our algorithms over our data and we plot the graphs shown in Figure 1. The results for ATTRIBUTE-REMOVAL are not affected by the step size ϵ or the number of steps T. We evaluated our algorithms for increasing amounts of rewrites with a maximum of 20. Our goal was to ensure that they would consider rewrites that would return at least 10 results. Overall, for GREEDY-REWRITE, we observe that an increasing step size leads to lower performance for all number of steps allowed.

For DP-REWRITE however, we observe two different trends. First, for small number of steps, DP-REWRITE faces a trade-off in choosing the step size. When the step size is small, DP-REWRITE fails to find rewrites that retrieve at least 10 results. However, when it is larger, DP-REWRITE finds rewrites that obtain at least 10 results, but with large total amount of relaxation across all attributes. Hence, we observe a U-shaped curve for smaller number of steps. When the number of steps is larger, the performance of DP-REWRITE is closer to monotonically increasing with the step size, as relaxations of at least 10 results are found for any step size, and hence smaller step sizes lead to better performance.

We now turn to study the performance of our algorithms in terms of the number of steps that they are allowed. To this end, we fixed the step size to 0.1 and examined the performance for increasing step values. We show the results in Figure 2. Our first observation is that both GREEDY-REWRITE and DP-REWRITE perform substantially better than ATTRIBUTE-REMOVAL. The second observation is that more steps do not necessarily translate to better performance. GREEDY-REWRITE performs worse as the number of steps increases. This is an artifact of the understimation of the number of results during the relaxation. In this case, GREEDY-REWRITE will continue to relax beyond the point necessary, leading to a set of results with higher MEAN-DIST, even though a run with smaller number of steps would terminate with a relaxed query leading to lower MEAN-DIST. Indeed, the performance of GREEDY-REWRITE deteriorates after 10 steps since the additional relaxation of the attributes only adds more unrelated results to the final set.

In contrast, for DP-REWRITE, increasing the number of steps leads to steady improvements in MEAN-DIST. While in principle DP-REWRITE may be affected by the aforementioned problem for GREEDY-REWRITE due to underestimation, it is less affected by poor estimation compared to GREEDY-REWRITE because it explores the space of relaxed queries more broadly. Nonetheless, after about 12 steps, the quality of results does not gain any further improvement improve since the algorithm seems to identify the same relaxed query regardless of the number of steps.

6. CONCLUSION

In this paper we propose a query-rewrite framework for answering structured web queries when users pose queries that would have led to very few results. Our framework takes into account the stringent time requirement of answering web queries, and balances it with the need of retrieving results close to the user queries. We describe two approaches to solving this problem, and show experimentally that both solutions produce meaningful results given our constraints.

7. REFERENCES

[1] Msn shopping xml api: Specs. `http://shopping.msn.com/xml/v1/getspecs.aspx?itemid=1202956773`.

[2] Msn shopping xml api: Televisions. `http://shopping.msn.com/xml/v1/getresults.aspx?bcatid=4724`.

[3] M. Bergman. The deep web: Surfacing hidden value. *Journal of Electronic Publishing*, 7(1), 2001.

[4] M. J. Cafarella, A. Y. Halevy, D. Z. Wang, E. W. 0002, and Y. Zhang. Webtables: exploring the power of tables on the web. *PVLDB*, 1(1):538–549, 2008.

[5] M. Fontoura, V. Josifovski, R. Kumar, C. Olston, A. Tomkins, and S. Vassilvitskii. Relaxation in text search using taxonomies. *PVLDB*, 1(1):672–683, 2008.

[6] N. Koudas, C. Li, A. K. H. Tung, and R. Vernica. Relaxing join and selection queries. In *VLDB*, 2006.

[7] D. Panigrahi and S. Gollapudi. Result enrichment in commerce search using browse trails. In *WSDM*, 2011.

[8] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proc. SIGMOD Conf.*, June 1995.

[9] N. Sarkas, S. Paparizos, and P. Tsaparas. Structured annotations of web queries. In *SIGMOD Conf.*, 2010.