Efficient Acquisition of Web Data through Restricted Query Interfaces

Simon Byers
byers@research.att.com
AT&T Labs-Research

Juliana Freire
juliana@research.bell-labs.com
Bell Laboratories

Cláudio Silva
csilva@research.att.com
AT&T Labs-Research

ABSTRACT

A wealth of information is available on the Web. But often, such data are hidden behind form interfaces which allow only a restrictive set of queries over the underlying databases, greatly hindering data exploration. The ability to materialize these databases has endless applications, from allowing the data to be effectively mined to providing better response times in Web information integration systems. However, reconstructing database images through restricted interfaces can be a daunting task, and sometimes infeasible due to network traffic and high latencies from Web servers. In this paper we introduce the problem of generating efficient query covers, *i.e.*, given a restricted query interface, how to efficiently reconstruct a *complete* image of the underlying database. We propose a solution to the problem of finding covers for spatial queries over databases accessible through nearest-neighbor interfaces. Our algorithm guarantees complete coverage and leads to speedups of over 50 when compared against the naive solution. We use our case-study to illustrate useful guidelines to attack the general coverage problem, and we also discuss practical issues related to materializing Web databases, such as automation of data retrieval and techniques which make it possible to circumvent unfriendly sites, while keeping the anonymity of the person performing the queries.

Keywords

query coverage, dynamic content, restricted query interfaces, spatial queries, wrappers

1. INTRODUCTION

The *hidden Web* has had an explosive growth as an increasing number of databases, from product catalogs and census data to celebrities' burial sites, go online. This information is often *hidden*, in the sense that it is placed behind form interfaces, and published on demand in response to users' requests. It is estimated that 80% of all the data in the Web can only be accessed via form interfaces [3].

There are many reasons for providing such interfaces on the Web. Transferring a big file with all the information in a database can unnecessarily overload Web servers (especially if users are interested in a small subset of the data). Furthermore, accessing a particular record within a big file can be rather cumbersome. The alternative of giving direct access to the databases through expressive query languages such as SQL [4] or XML-QL [2] is not practical, as these languages are too complex for casual Web users. Form interfaces are thus a good choice as they provide a very simple way to query (and filter) data.

Simplicity however, comes at a price. Form interfaces can be quite restrictive, disallowing *interesting* queries and hindering data exploration. In some cases, the restrictions are intentionally imposed by content providers to *protect* their data (*e.g.*, the book database and user comments are important IP for `amazon.com`, they are key to the business — and it is to Amazon's benefit that others are not able to replicate their data). In other instances, the restrictions are just an annoyance and often the result of bad design. Take for example the U.S. Census Bureau *Tract Street Locator*, [1] which requires a zip code and the first 4 letters of the street name, not allowing users to get information about all streets in a given zip code. As a result, there is a great wealth of information buried and *apparently* inaccessible in these Web sites.

For answering queries that are not allowed through a given interface, or simply to get better performance, many applications can benefit from materializing a view of this hidden data. This approach is currently used by a number of services, such as for instance, comparison shopping engines and job search sites. In general, these services act in cooperation with Web sites, which may give them a less restrictive back-door interface to the data. However, when Web sites do not cooperate, an interesting problem is how reconstruct their databases using the restricted query interfaces readily available on the Web.

The problem of querying Web sites through restrictive query interfaces has been studied in the context of information mediation systems (see *e.g.*, [6]). However, to the best of our knowledge, the problem of generating *efficient* query covers that retrieve complete images of the databases through restricted query interfaces has not been considered before.

In this paper we define and study the problem of generating efficient query covers for non-cooperative Web sites: given a restricted query interface, how to efficiently reconstruct a complete image of the underlying database. We focus on an important subset of this problem, namely, finding efficient covers for spatial queries over databases that are accessible through nearest-neighbor interfaces, and propose an algorithm that guarantees complete coverage and leads to speedups of over 50 when compared against the naive solution. We also discuss issues involved in building these database images and propose general guidelines.

2. SPATIAL COVERS THROUGH NEAREST-NEIGHBOR INTERFACES

Consider the following scenario. A large retail business is working on expansion plans, and needs to figure out where to place 10 new stores. A good strategy would be to discover where their competitors are located, and/or how their competitors have expanded over time. Nowadays, these data are often available on Web sites of the businesses themselves, as most retailers offer store locators which allow a potential customer to identify a location near them. One feature of many such locators is that they return only the closest k results to the query point, and in practice k ranges between 5 and 10. Using this restrictive form interface,

[1] http://tier2.census.gov/ctsl/ctsl.htm

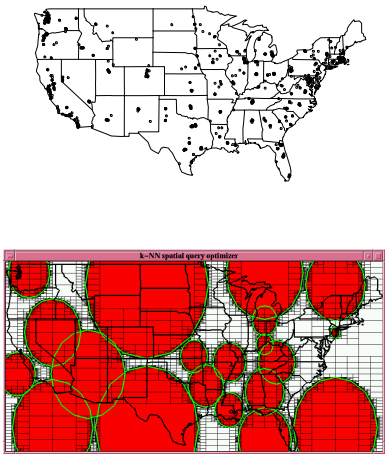

Figure 1: The top figure shows the location of stores from one U.S. retailer. The bottom figure shows some typical coverages queries, and the partial coverages while completing the cover for the region.

it can be tricky (and costly) to find the location of all the stores of a particular retailer in the U.S.. A natural solution would be to get a list of all the zip codes in the U.S., submit one by one, and merge the results. However, given the high latencies of most Web sites, executing up to 10,000 queries (1 per zip code) may take a long time; and this problem is compounded if these queries have to be executed periodically (for example, to track your competitor's expansion strategy).

Given a query interface for nearest-neighbor queries of the type "find the *k* closest stores to a particular location", and a region $\mathcal{R}$, our goal is to minimize the number of queries necessary to find all the stores in $\mathcal{R}$. While in principle our coverage algorithm works for any number of dimensions, we focus our discussion on the two-dimensional case. A naive technique for finding a cover of a region $\mathcal{R}$ is to simply break the region into small pieces, then perform one query for each piece (say, perform the query for the centroid of the region). This technique is quite popular on the Web. For instance, often to find all the stores of a particular chain, one performs one query per zip code. While this does not guarantee coverage, since it might happen that more than *k* stores belong to a single zip code, in practice, this often produces satisfactory results. Given that there are several thousand zip codes in the United States, this technique is likely to be very time consuming. Also, this technique does not explore the data-sensitive nature of the *k*-NN queries being performed, because it does not take into account the radius[2] returned by the query. As can be clearly seen in Figure 1, the radius returned by a given query can vary substantially. Our technique explores such variations to achieve an efficient solution. Our algorithm is quite simple, and is composed of two parts:

(1) We use a spatial data structure to keep track of which parts of $\mathcal{R}$ have already been covered by previous queries.

(2) At any given point in time, we use the coverage information obtained thus far to determine where to perform the next query as to minimize overlaps.

We use a simple greedy scheme for maximizing profit of queries. We assume the best place to perform a query is the largest empty circle in the uncovered region. In practice, we use a quadtree [5] to mark which regions of $\mathcal{R}$ have been covered (the unmarked regions are the regions of space which have not been seen, and for which no information is available). Given a query point $p \in \mathcal{R}$, the output of the query is a list of neighbors $n_1, \ldots, n_k$ of p. We simply mark on the quadtree the nodes inside a ball centered at p, and of radius $r = \max ||n_i - p||$. See Figure 1 for an example. In effect, we find the largest uncovered quadtree node, and use its center as the next query point. Note that we use the quadtree for two purposes: to determine coverages, and decide when we can stop; and to determine the next query. A nice feature of using recursive data structures such as the quadtree is that they make it easier to extend the technique to higher dimensions [5].

Querying for the U.S. locations of one retailer that has about 750 stores using the zip code query (which returns 10 stores, and can be performed in 1.2 seconds for each query) requires over 10,000 queries (corresponding to the different zip codes) at a total cost of 12,000 seconds, or over three-and-a-half hours. Using our simple scheme, we need only 191 queries at a total cost of 229 seconds. This is over 52 times faster (see [1] for more complete experimental results).

3. ACQUIRING WEB DATA

A critical requirement to reconstruct database images is to minimize the number of queries required to retrieve the data. In general, good strategy is to pose queries as general as possible, and some straightforward quidelines can be followed: leave optional attributes unspecified; for required attributes choose the most general options; and given a choice between required attributes, the attribute with smallest domain should be selected. However, for query interfaces that limit the number of returned answers, this strategy is not effective as one can not guarantee that all answers are retrieved. In the previous section we discussed how to address this problem by using spatial constraints to guarantee complete coverage.

There are a number practical issues involved in the Web data acquisition process: in order to build a warehouse of hidden Web data, several queries to one or more Web sites may be required — as a result, these queries must be automated, their execution optimized, and possibly anonymized; and since the degree of difficulty of coverage queries is directly related to how data is accessed and represented, finding alternative means to access the data, or different representations, may simplify the problem. We refer the reader to [1] for more details on these issues.

It is worth pointing out that the problem of reconstructing database images through restricted interfaces is very broad, and in this paper we propose a solution to a small but important subset of the problem. We are currently investigating a comprehensive framework to address the general reconstruction problem.

4. REFERENCES

[1] S. Byers, J. Freire, and C. Silva. Efficient acquisition of web data through restricted query interfaces. Technical report, AT&T Shannon Labs and Bell Laboratories, 2000.

[2] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for XML. In *Proc. of WWW*, pages 77–91, 1999.

[3] S. Lawrence and C. Giles. Searching the world wide web. *Science*, 280(4):98–100, 1998.

[4] J. Melton and A. Simon. *Understanding the New SQL: A Complete Guide*. Morgan Kaufmann, 1993.

[5] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.

[6] R. Yerneni, C. Li, H. Garcia-Molina, and J. Ullman. Computing capabilities of mediators. In *Proc. SIGMOD*, pages 443–454, 1999.

[2] In practice, some sites do not return the radius directly, but given an address (location), it is possible to find the latitude and longitude, and compute the radius by performing extra queries (possibly in different Web sites).