JAMES COOK UNIVERSITY

Content-based Navigation in a Mini-World Web

Curtis E. Dyreson

TR 97/06

DEPARTMENT OF COMPUTER SCIENCE

TOWNSVILLE
QUEENSLAND 4811
AUSTRALIA

Title	Content-based Navigation in a Mini-World Web
Primary Author(s)	Curtis E. Dyreson
Contact Information	Department of Computer Science James Cook University Townsville, QLD 4811 Australia curtis@cs.jcu.edu.au
Date	November 14, 1997

Abstract

Several database query languages have recently been developed to locate and retrieve documents in the vast network of World-Wide Web pages. These languages combine *path expressions*, which specify the structure of a path through the network to the desired information, with *content predicates*, which force the path to pass through pages with particular content. The straightforward implementation of these languages is based on breadth-first search of the network, with heavy reliance placed on the user's understanding of network topology to both direct and constrain the search via the appropriate use of the path expressions.

In this paper we describe a system that removes the reliance on path expressions to safeguard the search during a query and enables the user to navigate by refining *content* rather than by specifying *structure*. Our system uses a cost-constrained model for query evaluation. Links between pages are assigned costs. The user controls how far a query can navigate by specifying a permissible cost for each path. Only paths that are within the user-given cost are navigated during query evaluation.

We show how to implement content-based navigation by assigning a low cost to links that lead from more general to more specialized content and high cost to links that lead from more specialized to more general content. Our implementation has an efficient, scalable architecture which requires no changes to HTTP servers, and utilizes standard relational database technology to evaluate a query.

1 Introduction

The World-Wide Web (WWW) has grown tremendously over the past few years in terms of both the number of sites and the number of pages at each site. As the WWW has grown, so to has the importance of mechanisms to quickly find relevant information. To meet this need, the database research community has recently developed several languages to query WWW servers and retrieve (collections of) pages. Languages such as WebSQL [MMM96, AMM97, MMM97], W3QL [KS95], and WebLog [LSS96], combine *content predicates* to search a page with *path expressions* to traverse links between pages. A content predicate tests whether a page or link to a page has some specified content. A path expression is a description of the path that connects pages. For example, a user might issue a query to retrieve a page which contains the content 'Housing' that is reachable following at most four links from a page with the content 'James Cook University.'

This kind of query differs significantly from that supported by *search engines* e.g., Infoseek, Altavista, etc. Search engines only index *individual* pages. So a query for 'James Cook University AND Housing' using, say, Infoseek would only locate pages that have *both* concepts on a *single* page. In contrast, a query in a WWW query language could determine if there is a path from a page on 'James Cook University' to a page about 'Housing' that does not mention 'James Cook University'.

This paper describes a content-based approach to WWW query navigation. In content-based navigation no path expressions are used in a query. Instead, the navigation proceeds by refining content. Initially a user will identify a search space containing desired content, e.g., information about 'James Cook University.' Next, the user will narrow that space by refining the content, e.g., within the James Cook University space, find information about 'Housing.' This differs from the *path-based* navigation found in other WWW query languages. In path-based navigation, the user describes the *structure* of each path that can be navigated. For instance, the user might stipulate that the 'Housing' page be exactly three links from the 'James Cook University' page. The user must be careful to describe a path structure that can be feasibly navigated during a query. For instance, the user should avoid specifying a path which consists of any number of links between servers, since such a query might well visit every server on the WWW.

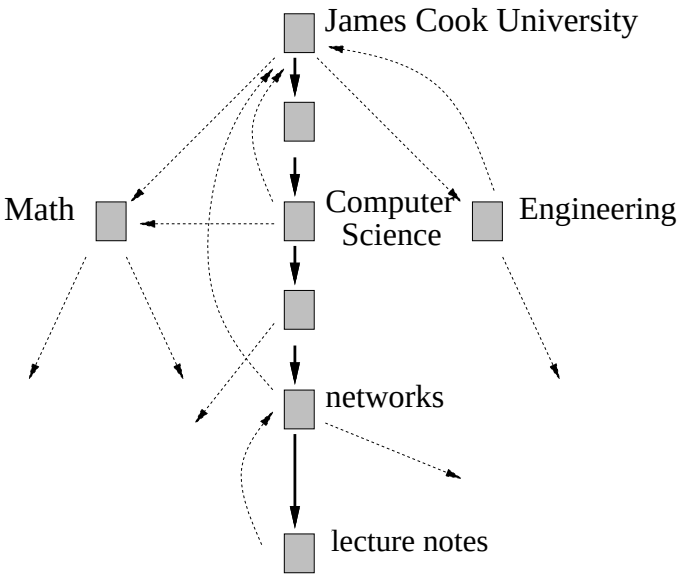

Figure 1: A simple navigation strategy

In the next section we argue that path-based navigation requires the user to have some a priori understanding of the topology of links in the region of the desired content to properly direct and constrain the navigation. We then briefly discuss the graph model of the WWW and how search engines locate content in the graph. Next, cost-constrained navigation is introduced. The idea behind this kind of navigation is to use cost constraints, rather than path expressions, to limit how far a query can navigate. We show how cost-constrained navigation can be implemented directly in SQL, assuming certain tables exist. After describing the SQL semantics we introduce a Mini-World Web (MWW). A MWW is a small subgraph in the WWW. We show how to implement content-based navigation in a MWW by building the necessary tables. We also give experimental results to demonstrate that the implementation is very practical. Finally we show how to create a federation of MWWs and discuss related and future work.

2 Motivating Example

The example given below is similar to the motivating queries given in most other WWW query language papers. Assume that a user wants the *lecture notes* for the *networks* course taught by the *Computer Science Department* at *James Cook University* (JCU). To retrieve the notes, the user formulates the following navigation strategy. First find pages on James Cook University. From those pages traverse links to the Computer Science Department pages, then to the networks course, and finally to the lecture notes. Figure 1 shows the desired navigation. In the figure, pages are shown as shaded boxes, and links between those pages as arrows. The desired navigation follows the thick solid arrows and ignores the dashed arrows.

A WebSQL query for this navigation strategy is given in Figure 2. The **FROM** clause stipulates that the Uniform Resource Locators (URLs) associated with 'James Cook University' are retrieved from a search engine (by using a **MENTIONS** to build the **DOCUMENT** *jcu*). For simplicity we will assume that only the URL of the university's home page, `http://www.jcu.edu.au`, is retrieved.[1] (Alternatively, the user may already know the URL

[1] A search for 'James Cook University' using the the Infoseek search engine does return the university's home page, as one of 2982 other URLs.

SELECT *notes*.url
FROM
    DOCUMENT *jcu*
        SUCH THAT *jcu* **MENTIONS** 'James Cook University',
    DOCUMENT *department*
        SUCH THAT *jcu* $\rightarrow \mid \rightarrow\rightarrow$ *department*,
    DOCUMENT *course*
        SUCH THAT *department* $\rightarrow^*$ *course*,
    DOCUMENT *notes*
        SUCH THAT *course* $\rightarrow \mid \rightarrow\rightarrow \mid \rightarrow\rightarrow\rightarrow$ *notes*
WHERE *department* **CONTAINS** 'Computer Science' **AND**
    course **CONTAINS** 'networks' **AND**
    notes **CONTAINS** 'lecture notes';

Figure 2: An example WebSQL query

for JCU's home page and can use it directly in the query.) Next, pages that are one or two *local* links from JCU's home page are visited as specified by the path constraint '$\rightarrow \mid \rightarrow\rightarrow$'. A local link is a link between pages on the same server. WebSQL distinguishes between local links and *global* links, which are links between pages on different servers. A '$\rightarrow$' indicates that one local link is followed, while '$\rightarrow\rightarrow$' specifies that two local links are followed. The '|' symbol is equivalent to an 'or' in the path expression. The **WHERE** clause adds the constraint (with a **CONTAINS** predicate) that for each page visited, navigation should continue only from those that contain the string 'Computer Science.'

But how many local links are on the path from JCU's home page to the Computer Science Department? Our survey of fifty university home pages shows a surprising variation in the link distance from a university home page to the (equivalent of) a Computer Science Department's home page: between one and seven links [Les96]. So the user might revise the path expression '$\rightarrow \mid \rightarrow\rightarrow$' in the query to specify any path, consisting of at most seven links, as follows.

```
...
DOCUMENT department
    SUCH THAT jcu → | →→ | →→→ | →→→→ | →→→→→ |
                  →→→→→→→ | →→→→→→→→ department,
...
```

Within seven links of the JCU home page there are approximately three thousand local pages comprising about 18MB of text data; three of these pages mention the Computer Science Department. However, searching this space will not locate any Computer Science Department pages since they happen to be on a different server; it is one *local* link and one *global* link to reach the Computer Science Department from the JCU home page.

Some WWW query languages do not distinguish between local and global links, e.g., WebLog and W3QL. A query in one of these languages would find the Computer Science home page, but would have to use some other mechanism to restrict the search (to avoid searching sites in, say, Canada that are within seven links of JCU's home page).

Suppose however that the query is reformulated to utilize a global link between *jcu* and *department*. Once it locates the Computer Science home page[2] it then searches for pages that

[2]We will assume that Computer Science Department home page for other universities are not located within

contain 'networks' using the path expression '$\rightarrow^*$'. This directs the search to follow any number of local links, and potentially to search the entire collection of pages available from a server. If a networks course does not exist, for instance, then the entire space will be searched in vain. A search of the Computer Science server will visit approximately eight thousand pages. While such a search is possible to execute it is not feasible to do so frequently.[3]

To constrain the query's search space, the user must know (approximately) how many links exist between pieces of information in order to limit the number of links in a path expression. But even with this knowledge, there is no guarantee that the query will evaluate quickly. Suppose, for instance, that the user knows that there are exactly three links between the Computer Science home page and the networks course home page, and limits the path expression to be exactly three local links (e.g., *department* $\rightarrow\rightarrow\rightarrow$ *course*). The resulting query will still traverse a significantly larger portion of the WWW than necessary, since there are many pages within three links of the Computer Science home page, very few of which have anything to do with the networks course.

The last path to traverse in the query leads from the networks course home page to the lecture notes. The navigation can follow paths consisting of at most three local links '$\rightarrow \mid \rightarrow\rightarrow \mid \rightarrow\rightarrow\rightarrow$'. Unfortunately, the lecture notes for all of the courses taught by the department are within three local links; it is one local link back to a list of courses page, then one to a course home page, and finally, one to the lecture notes for that course. Paths of length three also exist from the networks course through lecturers' home pages to notes in other courses. However, if the path expression is shortened ever so slightly to '$\rightarrow \mid \rightarrow\rightarrow$' then the query will find only the notes for the networks course. It is unlikely that a user will have such specific (and transitory) knowledge about the topology of the computer science pages.

In general, in order to direct and constrain the navigation in a WWW query using path expressions a user needs to have some a priori knowledge of the topology of the WWW in the region of desired content. The user may have to distinguish between local and global links, or she may have to know roughly how many links exist between pieces of information. Unfortunately, content is only loosely related to the graph topology of the WWW and although the user will likely be perfectly capable of specifying the content, the user will usually not have a priori knowledge of the specific topology (in the terminology of Lax et. al [LSCS97], the user has "zero knowledge" of the topology).

Finally, even when the query is completely specified appropriately, an implementation strategy is needed which is *not* based on dynamic searches of WWW pages. In this paper we present an implementation that shifts the cost of the search to a pre-processing phase and implements content-based navigation as an SQL query.

3 Graph Model of the WWW

All WWW query languages make use of a graph model of the WWW. In this section we describe this model. We also observe that the graph has some important, special properties.

The graph model is basically very simple. Pages[4] correspond to nodes, and links between pages to edges.[5] Because links are uni-directional, the graph has directed edges.

---

one global and seven local links of the JCU home page.

[3] We implemented the search in Perl 5.003 using the `libwww` library on our local server —a DEC-alpha— and, locally, with few network delays, the search took just under three hours. A similar search on the Association of Computing Machinery server from here in Australia took over twenty-eight hours, before we killed it after 12540 pages.

[4] We use the term 'pages' informally to refer to any Uniform Resource Instance (URI) such as images, video, sound, applets, etc.

[5] We will ignore the minor issue of links within a page.

At a practical level, the graph model of the WWW has some important properties.

- It is large. As of July, 1997 there were were an estimated 650 thousand servers and 350 million pages on the WWW.
- It cannot be traversed quickly. Only very small parts of the graph can be reasonably searched. A search involves downloading pages, extracting links, and following those links to download more pages. Retrieving a single page from a remote site, let alone a collection of pages, is slow (relative to reading a page from disk) because the page must be retrieved over the network.[6] In addition, network bandwidth is a limited, shared resource. To protect this resource, the Robot Exclusion protocol exists to restrict the activities of automated searches (i.e, precisely to restrict the kinds of traversals done in WWW queries).
- It is dynamic. Pages and links are constantly being added, updated, and to a lesser extent, deleted.
- Cycles in the graph are common, especially within a site. Site designers often overcome the lack of bi-directional links by providing 'back links' to a home page.

Because the WWW graph is large, constantly changing, and slow to search it is impossible to obtain a complete, accurate, up-to-date representation of the graph. But this does not mean that WWW queries must dynamically search the WWW to obtain information. The vast majority of the graph changes relatively little over short periods of time (say one to two weeks). Most of these changes are additions to the graph. Subgraphs, on the order of hundreds of thousands of nodes, relative to some time in the past can be obtained and stored. If the subgraph is explored and stored every week, this subgraph may not be completely accurate, but in general, it will be a close approximation of the actual subgraph.

4 Search Engines

A content-based navigation strategy is to traverse, in order, from pages with some particular content to pages with "refined" content. How this navigation is to be performed is not specified, only the content-based constraints that a path between pages must fufill.

To identify a set of pages with particular content, we typically utilize a *search engine*. A search engine can be thought of as a function that maps a content description to a set of URLs.

Definition 4.1 [search engine]
A search engine, S, is a function that maps a content description, x, to a set of URLs corresponding to the pages that "match" the content description,

$$S(x) = \{u \mid u \text{ matches } x\}$$

where u is a URL.
∎

A search engine consists of two parts: a *robot* that periodically searches the WWW and extracts page content and an *index* that maps content descriptions to URLs. There are many techniques for extracting the content. Content is commonly extracted from the text within the title, meta, and/or header tags, but it could be extracted from any text in a page. The extracted content can be individual words, phrases, or summaries of the page. Search engines also differ in how

[6] Pundits sometimes refer to the WWW as the World-Wide Wait.

much of the WWW they search. A search engine could be a global engine that indexes the entire WWW, such as Altavista or Infoseek, a search engine that provides a uniform interface to a set of global engines, such as MultiSurf, or a local engine that indexes information on a single server, such as `GlimpseHTTP`. What search engines have in common is that they map content to URLs.

Search engines extract content *statically*. Every few weeks or months a search engine will search a region in the WWW and extract the necessary content. Because the content has been extracted relative to some past state of the WWW, search engines might well map strings to URLs that no longer exist or match the extracted content.

Some WWW query languages have content predicates that *dynamically* extract content during query evaluation. For example, WebSQL supports a **CONTAINS** predicate that determines whether a page which has been retrieved during the evaluation of a query contains a particular string. Since dynamic extraction of content also associates a URL with a content description, it can be viewed as a special case of a search engine, one that is computed on the fly during query evaluation.

5 Cost-constrained Navigation

In addition to using search engines to identify which URLs match desired content, a WWW query language needs some strategy for navigating between pages. In this section we present a general strategy which we call *cost-constrained* navigation. In subsequent sections we show how to implement WebSQL and content-based navigation as instances of cost-constrained navigation.

The general idea behind cost-constrained navigation is to constrain the navigation to paths that have the right cost, and to prevent exploration of paths that are too expensive. In order to determine the cost of a path, each link between pages is assigned a weight. We assume that the weight is a natural number or a string; the weight to give a link depends on the desired navigation semantics. The cost of a path in a weighted WWW graph is computed from the weight of every link on that path, for instance, it could be the product of all the link weights. A cost-constrained navigation query takes as a parameter a permissible cost for each potential path. Only paths that are within the permitted cost are navigated.

A formal semantics for a query evaluated using cost-constrained navigation is given below.

Definition 5.1 [cost-constrained navigation]
A cost-constrained query, Q, takes n content descriptions, $x_1, \ldots, x_n$, and n permissible-cost sets $C_0, \ldots, C_{n-1}$. It produces a set of n-ary tuples of URLs as follows.

$$\begin{aligned} Q(x_1,\ \ldots,\ x_n,\ C_0,\ \ldots,\ C_{n-1}) = \{\ (u_1,\ \ldots,\ u_n)\ \mid \\ & u_1 \in S_1(x_1)\ \wedge\ \ldots\ \wedge\ u_n \in S_n(x_n)\ \wedge \\ & u_1 \rightarrow^* u_2 \rightarrow^* \ldots \rightarrow^* u_{n-1} \rightarrow^* u_n \in G\ \wedge \\ & C(u_1 \rightarrow^* u_2 \rightarrow^* \ldots \rightarrow^* u_{n-1} \rightarrow^* u_n) \in C_0\ \wedge \\ & \forall i[1 \leq i < n\ \wedge\ C(u_i \rightarrow^* u_{i+1}) \in C_i]\ \} \end{aligned}$$

where

- each x_i is a content description,
- C_0 is the set of permissible costs for the entire path from u_1 to u_n,
- each C_i is the set of permissible costs for the path from u_i to u_{i+1},
- each u_j is a URL,

path expression	cost set
$C(\rightarrow)$	$\{ 0 \}$
$C(\Rightarrow)$	$\{ 1 \}$
$C(p_1 p_2)$	$\{ x.y \mid x \in C(p_1) \ \wedge \ y \in C(p_2) \}$ (. is concatenation)
$C(p_1 \mid p_2)$	$C(p_1) \bigcup C(p_2)$
$C(p^*)$	$C(p) \bigcup C(pp) \bigcup C(ppp) \ \ldots$

Table 1: Modeling the cost of WebSQL path expressions

- each S_k is a search engine,
- G is a weighted WWW graph,
- $u_i \rightarrow^* u_j$ is a path (which could include cycles) in G from u_i to u_j, and
- the cost of that path is denoted $C(u_i \rightarrow^* u_j)$.

∎

A query evaluated using cost-constrained navigation determines if a cheap enough path exists in the WWW graph between URLs that have the desired content. Note that the query does not identify all the URLs on the path, that is, the exact path is irrelevant. The user controls the extent of the navigation via the cost sets, rather than by using path expressions. The cost sets are sets of permissible costs. C_0 is the cost set for the overall path, while the cost sets $C_1, \ldots, C_{n-1}$ specify the cost of paths between pairs of URLs with the desired content. In the next section we give an example of cost-constrained navigation.

5.1 Cost-constrained navigation in WebSQL

The path-based navigation semantics in existing WWW query languages can be reformulated using a cost-constrained navigation model. In this discussion we focus on WebSQL since it has the richest path expressions.

To model WebSQL queries, we first have to create a weighted WWW graph. WebSQL distinguishes between local and global links, a distinction that we will make by assigning weight 0 to a local link and 1 to a global link.

Next we must define a cost for paths in the WWW graph. Each path consists of some number of local and global links. We define the *cost* of a path to be a string consisting of the sequence of 1's and 0's on the path. That is, the cost of a path of length n is defined to be a string of length n where the i^{th} character in the string is '1' if the i^{th} link in the path is a global link, and '0' otherwise. For example, a path of three local links costs 000, a path of two local links followed by two global links costs 0011, while a path consisting of a single global link costs 1. Note that the cost of every path is a binary number, with significant leading zeros.

The cost sets in a WebSQL query are derived from the path expressions in the **FROM** clause. Recall that a path expression could be a local link ($\rightarrow$), a global link ($\Rightarrow$), a sequence of local or global links (e.g., $\rightarrow\rightarrow$), an alternation in the path (e.g., $\rightarrow \mid \Rightarrow$), or the Kleene closure of a path expression (e.g., $\rightarrow^*$). The path expression is, in effect, a regular expression over the alphabet of local and global links [FFLS97]. Table 1 gives the possible path expressions and their resulting cost sets. In the table, the cost set for a path expression, p, is denoted $C(p)$. The cost set is the set of strings recognized by the corresponding regular expression. For example, the cost sets for the query in Figure 2 are given below.

$C_0 = $	the set of all strings over the alphabet $\{ 1, 0 \}$	any number of local or global links
$C_1 = C(\rightarrow \mid \rightarrow\rightarrow) = $	$\{ 0, 00 \}$	one or two local links
$C_2 = C(\rightarrow^*) = $	$\{ 0, 00, 000, \ldots \}$	any number of local links
$C_3 = C(\rightarrow \mid \rightarrow\rightarrow \mid \rightarrow\rightarrow\rightarrow) = $	$\{ 0, 00, 000 \}$	one, two, or three local links

WebSQL currently does not utilize the cost set for the overall path, C_0, so it can be any cost. With these cost sets, a cost-constrained query can be given that implements the example query.

$Q('\texttt{James Cook University}', '\texttt{Computer Science}', '\texttt{Networks}', '\texttt{Lecture Notes}', C_0, C_1, C_2, C_3)$

Cost-constrained extensions of WebSQL can be envisioned. Say, for instance that we wanted the query given above to perform its navigation within an overall limit of five local links and one global link. We would use exactly the same query, but replace the overall cost set with the following one.

$$C_0 = \{ \text{strings of length} \leq 6 \text{ with at most one '1'} \} - \{ 000000 \} = \{ 0, 1, 01, 10, 001, \ldots, 100000 \}$$

We could also relax the cost constraints on individual paths, so that the query is allowed to explore a space of at most five local links and one global link from JCU's home page, with no specific constraints on paths that connect pairs of pages with desired content by issuing the same query with the following cost sets.

$C_0 = $	$\{ 0, 1, 01, 10, 001, \ldots, 100000 \}$	upto five local and one global links
$C_1 = $	the set of all strings over the alphabet $\{ 1, 0 \}$	any number of local or global links
$C_2 = $	the set of all strings over the alphabet $\{ 1, 0 \}$	any number of local or global links
$C_3 = $	the set of all strings over the alphabet $\{ 1, 0 \}$	any number of local or global links

Other query languages, such as WebLog and W3QL, have a similar cost model. In these languages each link in the WWW graph has a cost of one, and path expressions do not distinguish between local and global links.

5.2 An SQL Semantics

It may surprise the reader that a WWW query using a cost-constrained semantics can be given directly in SQL, assuming that a *SearchEngine* table, a *Reachable* table, and tables for each *CostSet* exist.

The SearchEngine table maps strings to URLs, and has the following schema.[7]

CREATE TABLE SearchEngine(content **STRING**, url **STRING**);

A tuple in the SearchEngine table associates a string describing some content with the URL of a page that contains that content.

The Reachable table records which URLs can be reached from which others by following some number of hyperlinks. In other words, it represents the transitive closure of the weighted WWW graph. The Reachable table has the following schema.

CREATE TABLE Reachable(from_url **STRING**, to_url **STRING**, cost **INTEGER**);

A tuple in the Reachable table indicates that the `to_url` is reachable from the `from_url` with the given `cost`. We will assume that the cost is an integer (perhaps the encoding of a string). This table might be infinite for one of the following reasons.

[7] We use a non-standard SQL type, **STRING**, to indicate a **CHARACTER** type with no fixed size.

- Cycles exist in the graph. If a path between two pages loops through a cycle, there are an infinite number of paths between those two nodes, each possibly having a different cost. To break cycles we stop a path if reaches a node that is already in the path.

- There are an infinite number of nodes in the graph. Assume that page `http://www/A.html` has a relative link to page `A.html` in directory `B`, i.e. `<A HREF="B/A.html">link</A>`. If directory `B` is a *symbolic link* to `A.html`'s parent directory, i.e., to '..', then the following infinite set of URLs locate page `A.html`.

```
{ http://www/A.html,
  http://www/B/A.html,
  http://www/B/B/A.html,
  http://www/B/B/B/A.html,
  http://www/B/B/B/B/A.html,
  ...  }
```

  We will assume a maximum limit on the length of a URL to prevent an infinite number of nodes. URLs longer than the maximum length will be ignored.

- The graph is growing faster than it can be traversed and computed [MM97]. In other words, the process of constructing the Reachable table never terminates since pages and links are being added ad infinitum. Assuming a finite alphabet, the limit on the size of a URL means that eventually the space of URLs will be exhausted, and consequently some URLs may well be ignored during construction of the Reachable table.

This is not to suggest that computing the Reachable and SearchEngine tables for the entire WWW is feasible or even possible. We would argue that it is impractical to do so for even a small part of the WWW, e.g., for any graph bigger than several hundred thousand nodes. Our goal in this section is to illustrate that an SQL semantics exists, given that these tables exist. In the next section we will discuss how to build real SearchEngine and Reachable tables.

We also need a CostSet table for each cost set. The cost set is simply a set of permissible costs.

CREATE TABLE CostSet(cost **INTEGER**);

A consequence of the Reachable table being finite is that all paths are bounded by some maximal cost (e.g., the product of the cost for every edge in the graph), and consequently each CostSet table is finite.

With these tables, we are in a position to give an SQL query that implements a content-based navigation.

Definition 5.2 [SQL implementation for cost-constrained navigation]
A cost-constrained query, $Q(x_1, \ldots, x_n, C_0, \ldots, C_{n-1})$ where each x_i is a content description and each C_j is a CostSet table, is equivalent to the following SQL query.

SELECT S_1.url, ..., S_n.url
FROM SearchEngine **AS** S_1, Reachable **AS** R_1, CostSet$_0$ **AS** C_0, CostSet$_1$ **AS** C_1
WHERE R_1.cost $\oplus$ ... $\oplus$ R_{n-1}.cost = C_0.cost **AND**
  R_1.cost = C_1.cost **AND**
  S_1.content = x_1 **AND** S_1.url = R_1.from_url **AND** R_1.to_url **IN** (
    SELECT S_2.url
    FROM SearchEngine **AS** S_2, Reachable **AS** R_2, CostSet$_2$ **AS** C_2

```
            WHERE R2.cost = C2.cost AND
                S2.content = x2 AND S2.url = R2.from_url AND R2.to_url IN (
                    ...
                    SELECT Sn.url
                    FROM SearchEngine AS Sn
                    WHERE Sn.content = xn
                    )
            ...
        )
    );
```

Note that the cost of the entire path is computed over some operation $\oplus$. The exact operation depends on how the cost of a path is computed (for WebSQL it would be a combination of shifts and additions).

A worst-case time analysis of this query will illuminate its mechanics. The query has $O(n)$ 'rounds', where each round extends the path(s) from pages matching one content description to pages matching the next content description. The path can be extended by finding all the pages that match the next content description, then determining which of those pages can be reached from the pages found in the previous round. This strategy performs a selection in the SearchEngine table first, and then joins the result with the Reachable table and the results from the previous round (a relation of URLs that we will call PreviousRound). The result is then joined to the CostSet table to validate the cost. Overall the method performs one selection and three joins per round as detailed below.

$$\sigma_{x_i}(\text{SearchEngine}) \underset{\text{url=to\_url}}{\bowtie} \text{Reachable} \underset{\text{from\_url=url}}{\bowtie} \text{PreviousRound} \underset{\text{Reachable.cost=cost}}{\bowtie} \text{CostSet}_i$$

Assume that in a single round, the query retrieves $O(s)$ tuples from the search engine table and $O(r)$ tuples from the Reachable table. Then the result set from a round has at most $O(sr)$ tuples. Further assume that the CostTable has $O(c)$ tuples. Overall the complexity of a single round, consisting of a selection and three joins is $O((sr)^2c)$. Given that there are $O(n)$ rounds, the query has a complexity of $O(n(sr)^2c)$. In general, the number of pages matching a content description, s, and the number of pages reachable from a page with desired content, r, will dominate the other factors.

While it is a desirable property that the query can be formulated entirely in SQL, that is, an extension of SQL is unnecessary, the query has some fairly obvious drawbacks. Namely, the Reachable table does not exist, nor would it be feasible to compute it for the entire WWW. In the next section we suggest one strategy for building the Reachable table, and we present a feasible implementation. The key to an SQL implementation is to limit the size of the Reachable table.

In summary, if we have a Reachable, SearchEngine, and CostSet tables, it is entirely possible to perform, in SQL, a cost-constrained navigation of a WWW graph.

6 A Mini-World Web

The WWW is too big and growing too fast to permit construction and maintenance of a single, unified graph using existing technologies. But small pieces of the WWW are a different matter.

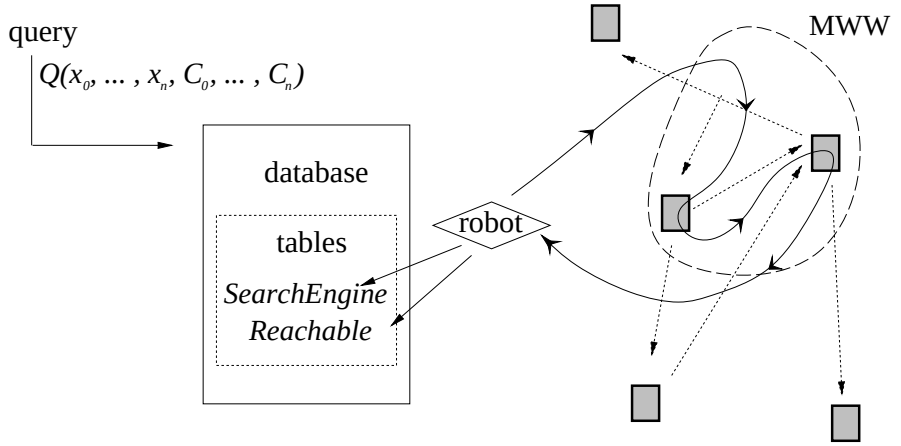

Figure 3: The Mini-World Web architecture

A *Mini-World Web (MWW)* is a relatively small part of the WWW that covers some domain of interest. A MWW could cover information on a single server, or on a small collection of servers, or information on some specific topic that is part of many servers. For instance, the user might be interested in a MWW consisting of the information available from the servers at James Cook University, or from servers at all the universities in Queensland, or just information in networks courses, available from universities world-wide. The difference between a MWW and the WWW is size. A MWW should be on the order of hundreds of thousands of pages, rather than hundreds of millions. Whether these hundreds of thousands of pages are on a single server, or distributed among thousands of servers is largely immaterial, as we shall see.

Within a MWW, our goal is to support fast, efficient queries that free the user from having to give path expressions to direct and constrain the query. In order to reach this goal, we adopt a *static* query evaluation architecture that avoids generating slow HTTP requests during query evaluation. The architecture is depicted in Figure 3. It has two major components. One component is a robot that periodically traverses the MWW and computes the Reachable and SearchEngine tables. The MWW must be small so that these tables will not be too large, and so that the robot can frequently recompute the tables. How frequently to run the robot is, of course, up to whomever maintains the MWW. The second component is a database that evaluates the SQL semantics for cost-constrained navigation. This architecture is *static* since a query will not dynamically traverse the MWW. The key assumption in this architecture is that the user will be willing to trade speed in query evaluation for a slightly out-of-date graph of the MWW.

6.1 The SearchEngine Table

For brevity we will omit discussion of this issue since techniques for building search engines on collections of pages, e.g., GlimpseHTTP, exist and are orthogonal to the focus of this paper. Note however that only static search engines are supported by a MWW, although some dynamic search engine predicates, such as a **CONTAINS** predicate for individual words, can be supported by a full text index of the pages in the MWW.

6.2 The Reachable Table

Content-based navigation is the successive refinement of a concept. Initially, a large concept space is identified. Next the concept space is refined or "narrowed". In the motivating query

presented in Section 2 the user first requested information on James Cook University. Subsequently the information space was narrowed to the Computer Science Department, then further narrowed to the networks course, and finally to lecture notes.

What this suggests is that the graph structure used for navigation must similarly refine or narrow the concept space. The WWW graph, however, does not have any such narrowing. Links can lead anywhere, they do not naturally lead from more general to more specialized information. Instead, it is necessary to define a narrowing graph structure on top of the underlying WWW graph. To support content-based navigation *some* such structure must be defined and used. In the remainder of this section we will present one such structure. This structure exists at many WWW sites, but not at all sites. Other structures exist, and we will briefly outline several in future work.

The structure we propose is based on URLs. A URL for a page pinpoints the page's *disk location.* For example, the URL

`http://www.cs.jcu.edu.au/ftp/web/Subjects/Informal.html`

specifies that the page is in the `ftp/web/Subjects` directory relative to the server's root directory. The server's root directory is established when the server is configured during installation or initialization.

The fact that a URL names a disk location leads to problems in the management of pages since the URL effectively "pins" the resource, if it is moved to a new location it breaks existing links. Most HTTP servers support URL relocation, whereby requests for resources that have been moved are automatically relocated to the new location where the resource currently resides.

But because the URL is related to the hierarchy of directories on a disk, it offers a primitive mechanism by which it is possible to make a initial guess as to which information is general and which is specialized. A general rule of thumb is that information in subdirectories is more specialized with respect to the information in parent directories. This is the common "cultural" convention for dividing information between parent directories and subdirectories in both WWW and non-WWW information repositories. So for instance, we might infer that the information in `http://www.cs.jcu.edu.au/ftp/web/Subjects/index.html` is "more general" than that in `http://www.cs.jcu.edu.au/ftp/web/Subjects/cp2001/index.html` since the former resides in a parent directory of the latter. We should note that we are *not* stating that the URL itself be used to infer the content of a page, since the name of the directory is often entirely unrelated to the content; few would know that cp2001 means a "Data Structures" course. Rather we are suggesting that some general/specialized relationship holds between the pages, at least, such a relationship is assumed by whomever maintains the pages. A page in a subdirectory contains information (whatever it may be) that is more specialized than that contained in a page in the parent directory (whatever that may be). Certainly, it will not always be the case that this is so, but it is a useful, cheap, and simple rule of thumb.

This assumption gives us enough information to distinguish different *kinds* of links. One kind of link is a *back link*. A back link is a link to a page that is more general, that is, to a page in a parent directory. Another kind is a *side link*. A side link is a link to a page in an unrelated directory which is neither a parent directory nor a subdirectory, e.g., a link to a page on a different server, or from a page in the server's root directory to one in a user's `public_html` directory. Links from parent to subdirectories are called *down links*. Down links lead from more general to more specialized pages. The most common kind of link, however, is between pages in the same directory, which are labeled according to the following rules. A link from an *index.html*[8] page to a page in the same directory is a down link, whereas one leading to an

[8] We use *index.html* in this discussion as the default name, the actual page name depends on the server. We mean the page that is loaded by default when the directory is used as the URL.

index.html is a back link. For other links within the same directory, it depends upon whether there is an incoming down or side link to a page from some other directory. If so, then the page can be reached from outside the directory and links from that page to pages in the same directory are treated as down links. If not, then the links are treated as side links.

In the cost model each kind of link is given a different weight. Back links are assigned an infinite weight, effectively pruning them from the graph. In general, we believe that back links should not be traversed. They usually represent a link back to a central entry page, e.g., back to the server root page, and consequently are counter to the narrowing needed in a content-based navigation scheme. Down links are weighted 1. These are the desirable links to follow since they lead to specialized information. Finally, side links are weighted 0. In general, few of these edges should be traversed, but perhaps some are needed.

A desirable property of using URLs in this manner is that the structure is completely under the site's control. Just as people who maintain pages can add information to a page to make those pages more attractive to search engines, the page maintainers can also introduce symbolic links or move pages as necessary in the hierarchy to change the information to better suit content-based navigation, although this is not required.

The final step to constructing the Reachable table is to define the cost of a path. We will use virtually the same definition given in Section 5.1. We define the *cost* of a path to be a string consisting of the sequence of 1's and 0's on the path. That is, the cost of a path of length n is defined to be a string of length n where the i^{th} character in the string is '1' if the i^{th} link in the path is a down link, and '0' if it is a side link (back links have been pruned from the graph). For example, a path of three side links costs 000, a path of two side links followed by two down links costs 0011, while a path consisting of a single down link costs 1.

6.3 Building the cost sets

In content-based navigation, the navigation is encouraged to follow any number of down links, since these links have the best chance, in general, of leading to more refined content. A single side link that leads to a page with the desired content is also permitted. More specifically, we define content-based navigation as only interested in paths that are some number of down links followed by at most one side link. All the paths of interest are a sequence of 1's followed by at most one 0. We stipulate that each cost set, $C_1, \ldots, C_{n-1}$, only permits these kinds of paths. The overall cost, C_0, is not used in our version of content-based navigation.

Since the costs are fixed and remain the same for every single query, we can impose the cost by eliminating all tuples from the Reachable table that do not have the desired cost. This prunes a significant number of paths from the table. Only paths that are either completely within a single directory "tree," or have at most one link outside of that tree are retained. From an original size of $O(n^2)$ tuples (assuming n nodes in the graph) the Reachable table is pruned to a size of $O(n\log_k(n))$, where k is the branching factor in the directory tree. The resulting Reachable table should be looked upon as the *default* content-based navigation strategy. MWW administrators can add other paths to the default table as they deem fit.

6.4 Queries

There are several benefits to the content-based navigation described above. First, the size of the Reachable table is modest, just slightly above linear in the size of the graph. Second, the cost constraints are optimized out of queries and applied to the Reachable table, saving some time during query evaluation. Finally, the semantics of a query are simplified.

Definition 6.1 [content-based navigation]

A query, Q, that uses content-based navigation takes n content descriptions, $x_1, \ldots, x_n$, and produces a set of n-ary tuples of URLs as follows.

$$Q(x_1, \ \ldots, \ x_n) = \{ \ (u_1, \ \ldots, \ u_n) \mid \ u_1 \in S_1(x_1) \ \wedge \ \ldots \ \wedge \ u_n \in S_n(x_n) \ \wedge \\ \forall i[1 \leq i < n \ \wedge \ (u_i, u_{i+1}) \in \text{Reachable}] \ \}$$

Although the MWW architecture that we have described uses a dedicated SearchEngine table, the semantics allows any search engine, e.g., Infoseek, to be easily incorporated. Generally search engines like Infoseek are slightly more costly to use in a query since querying such an engine involves making an HTTP request, and we believe that efficiency is attained by limiting network traffic. Also engines like Infoseek are not as comprehensive as dedicated search engines, since they will only index some of the pages at a site, usually just the top few levels in the server directory tree and the public_html trees.

6.5 Enriching the query language

The query semantics given above can be enriched by permitting each content description to be a *content expression*. A content expression is an expression built from the operands of *wildcards* and content descriptions and operations of negation, alternation, and conjunction. A wildcard means any content description, and should be used with care. Content expressions are built according to the following production rules.

expression ::= (*expression* **OR** *expression*) |
(*expression* **AND** *expression*) |
(**NOT** *expression*) |
description |
wildcard

wildcard ::= *****

description ::= 'any string'

So valid content expressions would include: notes, *****, (**NOT** 'notes'), ('notes' **OR** 'assignments'), and ('tutorials' **AND** ('notes' **OR** 'assignments')).

The enriched query language modifies the semantics of content-based navigation.

Definition 6.2 [extended content-based navigation]
A query, Q, that uses content-based navigation takes n content expressions, $e_1, \ldots, e_n$, and produces a set of n-ary tuples of URLs as follows.

$$Q(e_1, \ldots, e_n) = \{(u_1, \ldots, u_n) \mid \ u_1 \in E(e_1) \ \wedge \ \ldots \ \wedge \ u_n \in E(e_n) \ \wedge \\ \forall 1 \leq i < n, \ (u_i, u_{i+1}) \in \text{Reachable} \ \}$$

where $E(e_i)$ is a set of URLs recursively constructed as follows. If e_i is

- a description, then $S_i(e_i)$.
- $(e_i'$ **OR** $e_i'')$, then $E(e_i') \bigcup E(e_i'')$.

key	description	URL	pages	time (minutes)	rate (pages per minute)
JCU	James Cook University	`http://www.jcu.edu.au`	6929	74	93.63
JCUCS	Comp. Sci. at JCU	`http://www.cs.jcu.edu.au`	7939	230	34.51
Cairns	JCUCS - Cairns campus	`http://cairns.cs.edu.au`	389	6	64.83
UQ	The U. of Queensland	`http://uq.edu.au`	10541	582	18.11
UQCS	Comp. Sci. at UQ	`http://uq.cs.edu.au`	384	9	42.66
GU	Information Technology at Griffith University	`http://cit.gu.edu.au`	6012	102	58.94
CQU	Maths and Computing at Central Queensland Uni.	`http://science.cqu.edu.au/mc`	23648	1183	19.98
QLDGOV	Queensland State Government	`http://www.qld.gov.au`,	300	7	42.85
Novell	Novell Australia	`http://www.novell.com.au`	100	2	50.00
ACM	The Association for Computing Machinery	`http://www.acm.org`	12540	1710	7.33

Table 2: Robot performance on selected sites

- $(e_i'$ **AND** $e_i'')$, then $E(e_i') \bigcap E(e_i'')$.
- a wildcard, then $\pi_{\text{url}}(S_i)$.
- (**NOT** e_i'), then $E(*) - E(e_i')$.

■

6.6 Implementation and performance

The MWW architecture described in this paper has been implemented in Perl, version 5.003. We chose Perl to facilitate easy distribution. Our implementation does not use a commercial database, we wrote an object-oriented database to store the tables and evaluate queries. The database is a stand-alone Perl module, which is currently configured to use a GDBM file for each table, but could be configured to use most any database (the database API is very simple). The robot component of the architecture is based on modified versions of the `WWW::Robot` and `LWP` modules available from CPAN. We modified the standard packages to improve performance, primarily by writing our own `HTML::Parse` routines. The code and an example MWW interface are available from `http://www.cs.jcu.edu.au/~curtis/htmls/mww.html`.

To test the robot component we ran the robot on various sites over a period of several days. Table 2 shows the performance of the robot. The sites tested are primarily university servers in Queensland, Australia since we are building a MWW of these universities in order to improve the information discovery on the local subnet. The table shows the number of pages searched, the time taken (in minutes), and the search rate in pages per minute. The robot only explores HTML pages and does not download images, sound, PostScript documents, video, etc. The timings do not include the construction of the Reachable and SearchEngine tables. The timings reflect a normal department load and normal variations in network traffic. Each site was completely explored by the robot, except for CQU and ACM, where we killed the robot prior to completion since we felt that the time taken was excessive.

key	total links per page	links to HTML pages		
		total	internal	external
JCU	10.61	5.46	4.75	0.71
JCUCS	14.37	6.60	6.01	0.59
Cairns	5.02	4.08	2.76	1.32
UQ	9.47	6.34	3.91	2.43
UQCS	8.47	6.03	3.22	2.80
CQU	23.44	13.55	9.76	3.78
GU	9.36	7.83	6.88	0.96
QLDGOV	9.03	4.65	3.94	0.72
Novell	10.43	8.90	5.26	3.64
ACM	10.43	8.31	7.80	0.51

Table 3: Graph size of selected sites

The page rate attained by the robot fluctuates between 7 and 94 pages a minute. The rate will vary depending on the network load, pages loaded in various proxy servers, and a host of other factors, but we can conclude that the page rate will likely not exceed the low hundreds even under optimal conditions.

Table 3 summarizes the link information for each site. The table lists total number of links per page and the number of links to other pages (excluding obvious links to sound, images, etc.). The last category is subdivided into the number of links between pages internal to a site, and the number of external links.

The numbers in these two tables validate our decision to separate the WWW search from the query evaluation. Our goal is to make queries work on the order of seconds rather than minutes or hours. We believe that the user will be willing to use a slightly out-of-date WWW graph in order to evaluate queries quickly. A user cannot control how many pages will be fetched in a *dynamic* content-based or path-based query where the WWW is searched during query evaluation. For example, from a single starting page on the ACM server, a path expression such as '$\rightarrow\rightarrow\rightarrow$' will traverse 8.31^3 pages on average.

We then constructed example MWWs for all the servers. During construction we analyzed the number of down, side, and back links in the graph. In our sample data collection, 8% of links were back links and 14% were down links. Of the remaining 72%, over half were side links between pages within the same directory. This is due to the fact that many pages are automatically generated by programs, e.g., `latex2html` and `javadoc`, and these programs tend to place pages in a single directory. These cyclic clusters in the WWW graph are common, but usually they do not have links that lead out of the cluster, and so they have little impact on the overall size of the Reachable table. For example, the JCUCS MWW has 57398 links overall of which 9700 are back links and 12000 are down links. The Reachable table has 118237 tuples, which represents a doubling of the size of the graph.

On the JCUCS MWW we tested query evaluation performance. We designed a random query test whereby we randomly selected n content descriptions. We ran 100 random tests for $1 \leq n \leq 6$. The results are shown in Figure 4. The z-axis is time in seconds. The queries evaluated quickly because in our SearchEngine table most content descriptions index only a single page, so randomly chosen descriptions tend not to describe content-based paths in the MWW. In some sense, Figure 4 shows best case performance. To test worst case performance, we changed the pool of random content descriptions and performed the same test. In the new

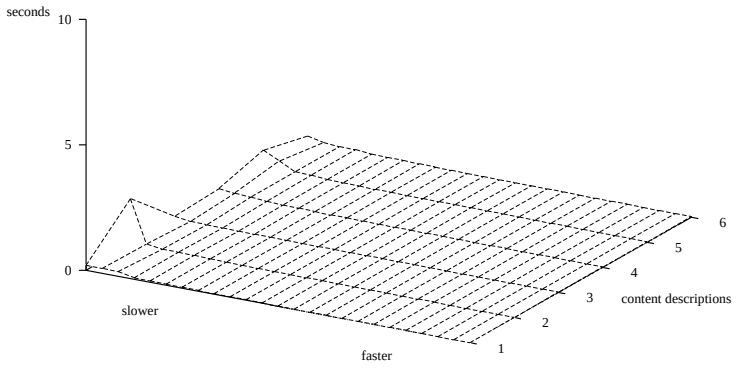

Figure 4: A "best-case" test

test we used only the top 50 content descriptions in our SearchEngine in terms of number of URLs indexed. These descriptions index between 113 and 724 URLs in the graph. The results are shown in Figure 5. The graphs are plotted with the same scale for the z-axis to highlight the differences, although this crops the graph. Figure 6 shows the uncropped graph with a larger time scale for the z-axis. These tests do not demonstrate that all queries will be fast, but they indicate that the design of a MWW does at least offer some hope for reasonable query evaluation speeds. We have not yet tested in any way to determine if content-based navigation produces "good" results (i.e., does it really refine content?). We hope to conduct a test (e.g., a user survey) of this highly subjective measure in future.

7 Scaling MWWs to the WWW

A single MWW maintains a subgraph of the WWW and responds quickly to queries for information in that subgraph. The experimental data shows that an individual HTTP server could provide a MWW for itself with little impact on its performance, e.g., as a CGI script and a nightly cron job to recompute the Reachable and SearchEngine tables. In this section we present a strategy for combining the information in individual MWWs which are distributed among a number of servers. We show that a query on a federation of independent MWWs, each with its own local SearchEngine table and Reachable table, can be solved efficiently, within a cost of two messages per MWW server.

In a MWW, a link from one server to another is a side link. Recall from Section 6.2 that just one side link is permitted on any path that connects a pair of pages in the Reachable table. Moreover, this side link must be the last link in the path. Hence in the evaluation of a content-based navigation query, a path that extends from one content description to the next can involve at most two MWWs. Either the path is entirely within a single MWW, or the path upto the final link is within a single MWW and the last link jumps to the other MWW. With a single side link on the path, no other paths are possible.

We can take advantage of the fact that paths are limited to adjacent MWWs by *decomposing* a query, querying each MWW independently, and building an appropriate answer set from the

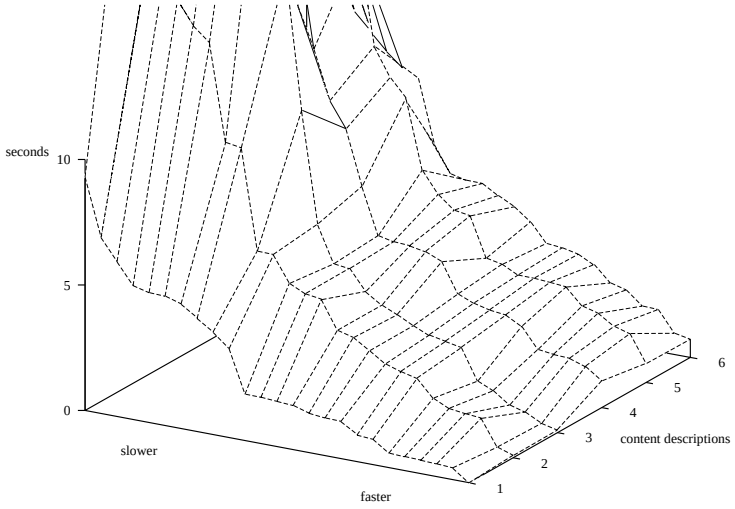

Figure 5: A "worst-case" test

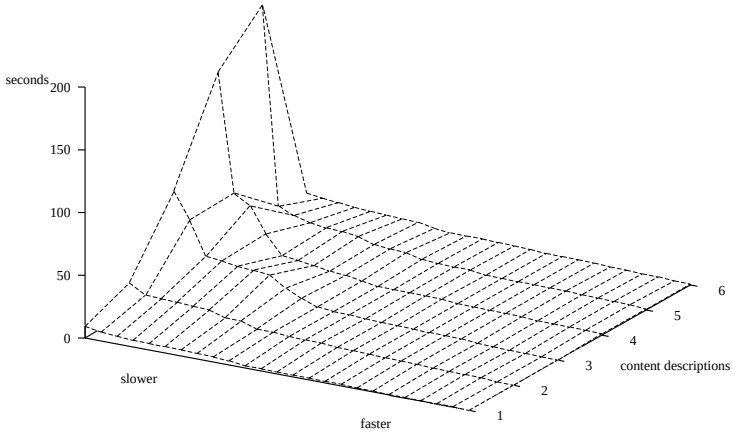

Figure 6: A "worst-case" test, uncropped

partial results. Assume that there is a site, which we call a *moderator*, that answers content-based navigation queries for a federation of k MWWs, $M_1, \ldots, M_k$. Only the moderator knows which MWWs are in the federation, each MWW is completely independent. Assume that each MWW in the federation supports a content descriptor, τ, that identifies reachable URLs which are external to the MWW (but not necessarily in the federation). Then we define the *decomposition* for a query as follows.

Definition 7.1 [query decomposition]
The *decomposition* of a query $Q(x_1, \ldots, x_n)$ is the following set of queries.

$$\{ Q_{ij}(x_i, x_{i+1} \ldots, x_{j-1}, x_j, \tau) \mid 1 \leq i \leq j < n \} \bigcup \{ Q_{in}(x_i, x_{i+1} \ldots, x_{n-1}, x_n) \mid 1 \leq i \leq n \}$$

■

The query decomposition is the set of queries for all partial paths in a single MWW, including paths to any adjacent MWWs. There are $O(n^2)$ queries in the decomposition. For example, the query Q(notes, Java, awt) has the following decomposition.

$$\{ Q_{11}(\text{notes}, \tau), Q_{22}(\text{Java}, \tau), Q_{12}(\text{notes, Java}, \tau) \} \bigcup \{ Q_{33}(\text{awt}), Q_{23}(\text{Java, awt}), Q_{13}(\text{notes, Java, awt}) \}$$

The moderator sends the decomposed query to each MWW in the federation in a single message (an HTTP GET on the appropriate CGI script) and receives the response in a second message. When the k MWWs have responded, the moderator pieces together the partial results to determine if the complete query has been answered. Partial results from different servers are pieced together using the following equivalence.

$$Q_{ik} = Q_{ij} \bowtie Q_{jk}$$

By building larger results from the join of partial results, the moderator can piece together the answer in $O(k^n)$ joins. In general, this is an infeasible number of joins, however n will generally be small (≤ 4) and many partial results will be empty or terminate at servers which are not in the federation. In any case, we suggest that the moderator determine whether the partial results are too many or too large to join, and act accordingly.

8 Related Work

W3QL [KS95], WebLog [LSS96], WebSQL [MMM96, AMM97, MMM97], and AKIRA [LSCS97] are query languages that have been designed specifically to retrieve information from the WWW. Each of these languages uses path-based navigation, except AKIRA, which is a hybrid path and content-based navigation system. AKIRA supports "fuzzy" navigation whereby a user can retrieve information "near" to a particular page. We base our query evaluation on content-based navigation and eschew the use of path expressions altogether to direct and constrain a query. We believe that many WWW queries will be content-refinement kinds of queries and that, for these queries, content-based navigation is more "user-friendly."

W3QL, WebLog, WebSQL, and AKIRA each have a *dynamic* query evaluation architecture in which a query will dynamically traverse the WWW. A dynamic architecture has several benefits. The most recent version of the constantly changing WWW graph is always used. The architecture is scalable since no large tables need to be computed or stored. A rich set of content predicates can be supported since content is extracted dynamically (WebLog and AKIRA have

extensive sets of content predicates). Dynamic pages (e.g., pages generated by CGI scripts) can be explored, which W3QL supports. In contrast, we propose a *static* architecture. Prior to query evaluation, a subgraph of the WWW, which we have called a Mini-World Web, is traversed and the graph stored in Reachable and SearchEngine tables. These tables optimize query evaluation within the subgraph. Our belief is that the user will be willing to use a slightly out-of-date WWW graph and sacrifice the other benefits of a dynamic architecture if doing so helps to improve the speed of query evaluation.

We share with WebSQL a practical focus both on the issue of cost and the means to control that cost by having different kinds of links. WebSQL distinguishes between local and global links. We distinguish between side, down, and back links. WebSQL estimates the cost of a query as a function of the number of global links in a path expression. We use side links to limit the cost of a query.

Semi-structured query language have also been applied to the WWW [Bun97, BDHS96, MAG+97, FFLS97]. Unlike WWW query languages, semi-structured query languages have content-based navigation. A semi-structured query can be thought of as a regular expression over an alphabet of content descriptions. A query is evaluated by matching the regular expression against the content descriptions on a path. This paper borrows several ideas from semi-structured query languages for the WWW. The semantics for enriching the query language presented in Section 6.5 is taken directly from StruQL [FFLS97]. The issue of scaling the MWW by decomposing queries is related to a similar problem addressed in [Suc96], although in content-based navigation the problem is much simpler. The cycle breaking strategy, resulting in a "tree-like" model, is from UnQL [BDHS96].

In contrast to these other papers however, we focus on the issue of creating a semi-structured data model from the unstructured information in the WWW. A straightforward approach would make a one-to-one mapping between hyper-links in the WWW and links in a semi-structured data model. In contrast, our approach is to remove some links (back links) and add a significant number of additional links (all reachable down links). We have not investigated techniques for minimizing the additional links [GW97].

Finally, unlike most previous work in WWW and semi-structured query languages, we directly address performance issues and we reuse traditional database technology to support the new query techniques.

9 Conclusions

In this paper we introduced content-based navigation in WWW query languages. Content-based navigation is a mechanism that enables a user to navigate by refining *content* rather than by specifying *structure*. We showed how to implement content-based navigation by assigning a low cost to links that lead from more general to more specialized content and high cost to links that lead from more specialized to more general content. Our implementation has an efficient, scalable architecture which requires no changes to HTTP servers, and utilizes standard relational database technology to evaluate a query.

References

[AMM97] G. Arocena, A. Mendelzon, and G. Mihaila. Applications of a web query language. In *Sixth International World Wide Web Conference*, Santa Clara, CA, April 1997.

[BDHS96] Peter Buneman, Susan B. Davidson, Gerd G. Hillebrand, and Dan Suciu. A query language and optimization techniques for unstructured data. In H. V. Jagadish

and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 505–516, Montreal, Quebec, Canada, 4–6 June 1996.

[Bun97] P. Buneman. Semistructured data. In *SIGMOD/PODS '97 (tutorial notes)*, Tucson, AZ, May 1997.

[FFLS97] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for a web-site management system. *SIGMOD Record*, 26(3), September 1997.

[GW97] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *Proceedings of the International Conference on Very Large Databases (VLDB '97)*, pages 436–445, Athens, Greece, September 1997.

[KS95] D. Konopnicki and O. Shmueli. W3qs: A query system for the world wide web. In *Proceedings of the International Conference on Very Large Databases (VLDB '95)*, pages 54–65, Zurich, Switzerland, September 1995.

[Les96] N. Leslie. Redesigning jcu web pages. Internal JCUCS Memo, December 1996.

[LSCS97] Z. Lacroix, A. Sahuguet, R. Chandrasekar, and B. Srinivas. A novel approach to querying the web: Integrating retrieval and browsing. In *ER97 - Workshop on Conceptual Modeling for Multimedia Information Seeking*, Los Angeles, CA, November 1997.

[LSS96] L. Lakshmanan, F. Sadri, and I. Subramanian. A declarative language for querying and restructuring the web. In *Proceedings of the Sixth International Workshop on Research Issues in Data Engineering (RIDE '96)*, New Orleans, February 1996.

[MAG+97] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, September 1997.

[MM97] A. Mendelzon and T. Milo. Formal models of web queries. In *Proceedings of the Sixteenth ACM Symposium on Principles of Database Systems (PODS '97)*, pages 12–15, Tucson, AZ, May 1997.

[MMM96] A. Mendelzon, G. Mihaila, and T. Milo. Querying the World Wide Web. In *Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems*, pages 18–20, Miami, FL, December 1996.

[MMM97] A. Mendelzon, G. Mihaila, and T. Milo. Querying the World Wide Web. *International Journal on Digital Libraries*, 1(1):54–67, January 1997.

[Suc96] D. Suciu. Query decomposition and view maintenance for query languages for unstructured data. In *Proceedings of the International Conference on Very Large Databases (VLDB '96)*, pages 227–238, Mumbai, India, September 1996.