Complex Queries over Web Repositories

Sriram Raghavan Hector Garcia-Molina

Computer Science Department
Stanford University
Stanford, CA 94305, USA
{rsram, hector}@cs.stanford.edu

Abstract

Web repositories, such as the Stanford WebBase repository, manage large heterogeneous collections of Web pages and associated indexes. For effective analysis and mining, these repositories must provide a declarative query interface that supports *complex expressive Web queries*. Such queries have two key characteristics: (i) They view a Web repository simultaneously as a collection of text documents, as a navigable directed graph, and as a set of relational tables storing properties of Web pages (length, URL, title, etc.). (ii) The queries employ application-specific ranking and ordering relationships over pages and links to filter out and retrieve only the "best" query results. In this paper, we model a Web repository in terms of "Web relations" and describe an algebra for expressing complex Web queries. Our algebra extends traditional relational operators as well as graph navigation operators to uniformly handle plain, ranked, and ordered Web relations. In addition, we present an overview of the cost-based optimizer and execution engine that we have developed, to efficiently execute Web queries over large repositories.

1 Introduction

The Stanford WebBase repository [4] is a special-purpose warehouse that stores large collections of Web pages and associated indexes. The repository operates in conjunction with a "crawler" that periodically traverses the Web to gather pages to populate the repository. The pages and indexes in the WebBase repository provide a rich corpus for large-scale Web mining experiments (e.g., computing PageRank, trawling for communities, scalable clustering, similarity indexing, etc.) as well as for more focused Web analysis queries.

To illustrate the types of queries that a trained information analyst could execute over WebBase, consider the following two examples. In Example 1, we attempt to generate a list of universities that Stanford researchers working on "Mobile networking" collaborate with. To this end, we examine the hypertext links from important "Mobile networking" pages inside Stanford to the websites of other universities (see Figure 1).

Example 1. *Let S be a weighted set consisting of all the pages in the stanford.edu domain that contain the phrase 'Mobile networking'. The weight of a page in S is equal to the normalized sum of its PageRank and text search ranks. Compute R, the set of all the ".edu" domains (except stanford.edu) that pages in S point to (we say a page p points to domain D if it points to any page in D). For each domain in R, assign a weight equal to the sum of the weights of all the pages in S that point to that domain. List the top-10 domains in R in descending order of their weights.*

In Example 2, the editor of the local university newspaper wishes to determine the relative popularity of the three comic strips Dilbert, Doonesbury, and Peanuts, amongst people at Stanford University. With each comic strip C, he associates a website C_s, and a set C_w containing the name of the strip and the names of the characters featured in that strip. For example, $Dilbert_w$ = {Dilbert, Dogbert, The Boss} and $Dilbert_s$ = dilbert.com. He uses a combination of word occurrences and link information to compute a measure of popularity for each strip.

Example 2. *Extract a set of at most 10000 pages from the stanford.edu domain, preferring pages whose URLs either include the "$\sim$" character or include the path fragment "/people/". Call this set S. For each comic strip C, compute $f_1(C)$, the number of pages in S that contain the words in C_w, and $f_2(C)$, the number of pages in C_S that pages in S point to. Use $f_1(C) + f_2(C)$ as a measure of popularity for comic strip C.*

These examples illustrate two key characteristics of Web analysis queries:

1. Multiple views of a Web repository. Web analysis

**Proceedings of the 29th VLDB Conference,
Berlin, Germany, 2003**

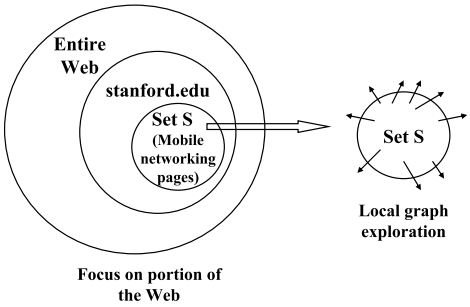


Figure 1: Example 1

queries combine *navigation* operations on the Web graph (to refer to pages based on their hyperlink interconnections), *text search predicates* (to refer to pages based on their content), and predicates on attributes of Web pages, to express complex query semantics. For instance, Example 1 combines a predicate on the domain, a text-search query, graph exploration around set S, and link weighting based on ranks. Thus, complex queries simultaneously employ three different views of a Web repository: as a *document collection*, as a *directed graph*, and as a *set of relations* storing page and link attributes.

2. Ranking, Ordering, and Top-k results. Web analysis queries employ user-defined notions of ordering and ranking (of pages and links) as a mechanism for dealing with the *size* and *heterogeneity* of Web data sets. The use of ranking and ordering functions enables complex queries to prioritize result elements and selectively retrieve only the "best" results.

For instance, in Example 1, the query defines the ranking of the domains in R as a function of the PageRank and text search rank of the pages in set S. This ranking enables selective retrieval of only as many results as are needed (the top-ranked 10 entries in this case), without wading through a huge result set. In addition, the *document collection* view of a repository requires support for ranking, since text query operators inherently return ranked results (e.g., a text query "CONTAINS 'Mobile networking"' will rank pages that contain the exact phrase higher than those that separately contain 'Mobile' and 'networking'.)

Similarly, in Example 2, since there is no guaranteed mechanism for identifying personal Web pages in the Stanford domain ("heterogeneity" at play), a heuristic is employed. The query identifies certain URL patterns that are more likely to yield personal Web pages and expresses a preference for such pages when conducting the analysis.

Currently, the Stanford WebBase repository provides two interfaces to its content: a *streaming bulk access interface* to retrieve large pieces of the repository as a stream of Web pages over the network, and a *programmatic interface* to access the repository indexes. However, execution of complex analysis queries using these interfaces is a tedious task, requiring users to design and implement query-specific execution plans for accessing indexes, retrieving pages, generating rankings, etc. Thus, there is a need to provide an interface for declaratively formulating and then efficiently executing complex Web queries.

2 Challenges and Solution Approach

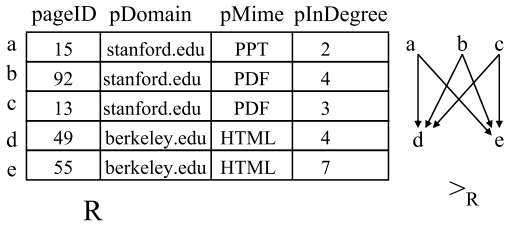
The design of a formal model and query algebra for Web repositories poses several challenges, due to the the unique properties of the queries highlighted in the previous section. Query models used in relational, semi-structured, or text retrieval systems provide some, but not all of the features required to support Web queries.

For instance, text retrieval systems employ one of the standard IR models (Extended Boolean, Vector-space, Probabilistic model, Bayesian Network, etc.) in conjunction with query languages that enable keyword querying, pattern matching (e.g., regular expressions, substrings), and structural queries [2]. Thus, treating a Web repository as an application of a text retrieval system will support the "document collection" view. However, queries involving navigation or relational operators will be extremely hard to formulate and execute.

On the other hand, the relational model provides a rich and well-tested suite of operators for expressing complex predicates over Web page attributes. However, ranks and orders are not intrinsic to the the basic relational model. Motivated by financial and statistical applications (e.g. computing moving averages), there have been previous attempts to introduce order into relations and relational operators (SEQUIN [14], SRQL [5], AQuery [8]). However, as we discuss in Section 7, there are several key differences between our approach and the extensions proposed in these systems. In particular, our approach admits a more general class of orders, uniformly deals with ranks and orders, allows more sophisticated ranking functions, and extends ranking and ordering to Web navigation.

In this paper, we propose a formal model and algebra for Web queries, with the following features:

- The use of a simple relational schema and the notion of "Web relations" to model the operands and results of Web queries
- Well-defined semantics when combining navigation, relational, and text search operators into complex queries
- A mechanism, based on "partial orders", for defining and manipulating orders in Web relations (to model user-defined preferences as in Example 2)
- Operators to specify application-defined ranking function, compose ranks from multiple functions, and retrieve "top-k" results prioritized either by ranks or by order relationships
- Unified treatment of "plain", ranked, and ordered Web relations, including (i) extensions to traditional relational operators, and (ii) precise definitions of navigation operators in the presence of ordering and ranking



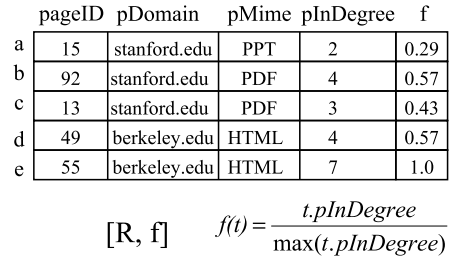


Figure 2: Ranked, and ordered relations

3 Model of a Web Repository

We model the repository as a collection of pages and links (corresponding to Web hyperlinks), with associated page and link attributes. We present a relational schema that is specialized for Web repositories and incorporates ranking and ordering of pages and links. Note that a relational schema is merely used as a conceptual modeling tool, and does not impose any restriction on the physical implementation of a Web repository (analogous to the notion of "physical data independence" adopted in traditional databases). For instance, the Stanford WebBase repository [4] implementation involves a combination of a relational database, an embedded database, specially formatted files, and custom file-based index structures.

3.1 Preliminary definitions

To formally describe our model, we adopt the following definitions and notational conventions:

Page. We use the term "page" to refer to any Web resource that is referenced by a URL, crawled, and stored in the repository (e.g., a HTML Web page, a plain text file, a PDF document, image, other media file, etc.). We associate a unique identifier $pageID$ with each page.

Link. We use the term "link" to refer to any hypertext link that is embedded in the pages in the repository. Each link is associated with a source page (the page in which the hypertext link occurs) and a destination page (the page that the link refers to), and a unique identifier $linkID$. A separate link identifier allows us to unambiguously identify a link even when there are multiple links between the same source-destination pair.[1]

Ordered relation. Given a relation R and a strict partial ordering $>_R$ (i.e., an irreflexive, anti-symmetric, and transitive binary relation) on the tuples of R, we refer to the pair $[R, >_R]$ as an ordered relation on R. Conversely, R is the "base relation" corresponding to $[R, >_R]$. For instance, in Figure 2, we define an ordered relation $[R, >_R] = [R, \{a >_R d, a >_R e, b >_R d, b >_R e, c >_R d, c >_R e\}]$, where each tuple whose domain attribute is stanford.edu is $>_R$-related to any tuple outside the stanford.edu domain. This partial order is depicted in Figure 2 using the standard Hasse diagram notation (in a Hasse diagram for a partial order $>$, a directed path from node a to node b implies that $a > b$, and conversely).

Ranked relation. Given a relation R and a function w that assigns weights (normalized to the range [0,1]) to the tuples of R, we can define a new relation $[R, w]$ that is simply R with an additional implicit real-valued attribute w.[2] Specifically, for each tuple $t \in R$, $[R, w]$ contains a tuple t' that contains all the attributes of t and in addition has $t'.w$ equal to the rank of t under w. We refer to $[R, w]$ as a ranked relation on R and to R as the "base relation" of $[R, w]$. For instance, in Figure 2, to generate $[R, f]$ the tuples are ranked using $f(t) = \frac{t.pInDegree}{\max_R(t.pInDegree)}$ (e.g., rank of tuple a is $\frac{2}{7} \approx 0.29$).

Note that we do not require the ranking function w to operate on a per-tuple basis. In particular, as in the example described above, the rank of $t \in R$ can depend on the attributes of all the tuples in R, not just on t. Formally, $w : R \times \{R\} \rightarrow [0, 1]$, even though we will use $w(t)$ as a shorthand for $w(t, R)$ when the relation is clear from the context. Finally, note that every ranked relation $[R, w]$ can be associated with an ordered relation $[R, >_w]$ by defining an ordering $>_w$ such that $t_1 >_w t_2$ iff $w(t_1) > w(t_2)$.

Ranking versus ordering. The notions of ranked and ordered relations help to model two different kinds of application semantics used in Web queries. Ordered relations are useful for expressing preferences for certain kinds of pages or links without necessarily quantifying how much one kind is preferred over another (e.g., "prefer PDF, Postscript, or plain text files to MS Word documents", "prefer intra-host links to inter-host links", "prefer pages crawled within the last week to older pages", etc.). In addition, since our representation of such preferences is based on partial orders, the preferences need not involve all available pages or links. For instance, we can express a preference for HTML files over powerpoint (PPT) files, and not involve other document types at all, by defining $[R, >] = [R, \{d > a, e > a\}]$ on base relation R in Figure 2.

In contrast, ranked relations are useful when applications (i) can precisely quantify their relative preferences (e.g., HTML files get a weight of 0.6, PDF and Postscript files get a weight of 0.4, and everything else is weighted 0.2), (ii) use precomputed ranks generated by sophisticated offline algorithms (e.g., PageRank), or (ii) mathematically compose ranks derived from multiple sources (e.g., *SUM(PageRank, text-search rank)*).

Formal model of a Web repository. We model a Web repository as a 6-tuple $\mathcal{W} = (\mathcal{I}_p, \mathcal{I}_l, \mathcal{W}_R, \mathcal{P}, \mathcal{L}, \mathcal{F})$, where:

[1] We do not include intra-page hyperlinks in defining our set of links.

[2] Without loss of generality, we assume that the attribute names in R do not clash with these specially added rank attributes.

- $\mathcal{I}_p$ (resp. $\mathcal{I}_l$) is an identifier space from which the $pageID$ (resp. $linkID$) for every page (resp. link) is chosen. Without loss of generality, we assume that $\mathcal{I}_p \cap \mathcal{I}_l = \emptyset$, and that $\mathcal{I}_p$ and $\mathcal{I}_l$ are disjoint with respect to the domain of any other attributes in $\mathcal{W}$.
- $\mathcal{W}_R$ is a set of plain, ranked, or ordered relations called *Web relations*. A relation R is said to be a Web relation if it contains *at least one* attribute whose domain is $\mathcal{I}_p$, $\mathcal{I}_l$, $2^{\mathcal{I}_p}$, or $2^{\mathcal{I}_l}$. A ranked relation $[R, f]$ or an ordered relation $[R, >_R]$ is a Web relation if the corresponding base relation R is a Web relation.
- $\mathcal{P} \in \mathcal{W}_R$ is a *universal page relation*. $\mathcal{P}$ contains one tuple for each page in the repository and one column for each page attribute. $\mathcal{P}$ includes an attribute $\mathcal{P}.pageID$ whose domain is $\mathcal{I}_p$ and which forms a primary key for $\mathcal{P}$. Thus, $\mathcal{P}$ has a schema of the form $\mathcal{P} = (pageID, ...)$.
- $\mathcal{L} \in \mathcal{W}_R$ is a *universal link relation*. $\mathcal{L}$ contains one tuple for each hyperlink in the repository and one column for every available link attribute. $\mathcal{L}$ includes an attribute $\mathcal{L}.linkID$ whose domain is $\mathcal{I}_l$ and which forms a primary key for $\mathcal{L}$. In addition, each link will include a $srcID$, the identifier of the page in which the link occurs, and a $destID$, the identifier for the target of the hyperlink. The domain of both $\mathcal{L}.srcID$ and $\mathcal{L}.destID$ is $\mathcal{I}_p$, and there is a referential integrity constraint from $\mathcal{L}.srcID$ to $\mathcal{P}.pageID$. Thus $\mathcal{L}$ has a schema of the form $\mathcal{L}$ = $(linkID, srcID, destID, ...)$.
- $\mathcal{F}$ is a set of predefined page and link ranking functions that have been registered in the repository (see examples below).

We identify four common types of Web relations. A Web relation $R = (A_1, A_2, \ldots, A_n)$ such that for some $i \in 1 \ldots n$, $domain(A_i) = \mathcal{I}_p$, $domain(A_j) \neq \mathcal{I}_p \, \forall j \neq i$ is called a *page relation*. Analogously, *link relations*, *pageset relations*, and *linkset relations* are Web relations with exactly one attribute whose domain is $\mathcal{I}_l$, $2^{\mathcal{I}_p}$, or $2^{\mathcal{I}_l}$ respectively. By definition, the special relations $\mathcal{P}$ and $\mathcal{L}$ are themselves page and link relations respectively.

The elements of set $\mathcal{F}$ are functions that operate on plain relations to produce ranked relations. For instance, $\mathcal{F}$ may contain an element f_{pRank} that operates on a page relation R to yield $[R, f_{pRank}]$, in which the tuples are ranked using the normalized PageRank of the constituent pages. As another example, $\mathcal{F}$ could contain an element $f_{tfidf}(s)$ that ranks tuples of a page relation based on the ranks of the pages (using the standard TF-IDF ranking scheme [2]) when searching for the string s (e.g., f_{tfidf}("Web repositories")).

Note that $\mathcal{W}_R$, the set of Web relations in the repository, contains the two special relations $\mathcal{P}$ and $\mathcal{L}$. In addition, since the result of any complex Web query is itself a Web relation (see Section 4), query results can be stored as elements of $\mathcal{W}_R$ and used in future queries.

Finally, though the schema definitions require only one attribute for P and three for L, a typical useful Web repository would have many more. In [13], we list the names and data types of the page and link attributes used in our experimental repository. We will also refer to some of these attributes when formulating sample complex queries in Section 5.

Category	Operator list
Unary relational	Select (σ) Project (Π) Group-by (γ)
Binary relational	Union ($\cup$) Intersection ($\cap$) Set-difference ($-$) Cross-product ($\times$)
Ranking and Ordering	Rank (Ψ) Order (Φ) Prune (Ω_k) Compose ($\Theta_{h,op}$)
Navigation	Forward navigation ($\overrightarrow{\Lambda}$) Backward navigation ($\overleftarrow{\Lambda}$)

Table 1: List of query operators in our model

4 Query Operators

In this section, we list and define some of the query operators that are used to build complex Web queries. Table 1 lists the complete suite of operators in our algebra, classified into four categories. All the operators listed in the table manipulate only *Web relations*, i.e., the result of any operator as well as its operands are plain, ranked, or ordered Web relations.

In the interest of space, we will present only the more interesting and non-standard operators in this section. Specifically, we will take up for detailed discussion, the operators in the last two categories of Table 1 ("ranking and ordering" and "navigation"), as well as the group-by and cross-product operators. For the remaining relational operators, extensions to their semantics to handle ordered and ranked relations are formally defined in [13]. Below, we summarize only the salient aspects of these extensions (throughout, we will refer to relational operators on multisets as in SQL):

Select. When selecting from a ranked relation $[R, f]$, the selection predicate can refer to the ranking attribute $R.f$. Also, $\sigma([R, >_R]) = [S, >_S]$ where $S = \sigma(R)$ and $>_S$ is merely $>_R$ restricted to the tuples in S.

Project. Projection on $[R, f]$ implicitly retains $R.f$ in the result. In addition, two special projection rules are employed to yield base relations: $\Pi_{-rank}([R, w]) = R$ and $\Pi_{-ord}([R, >_R]) = R$.

Set operations with ordering. Union, intersection, and set-difference of a pair of ordered relations $[X, >_X]$ and $[Y, >_Y]$ produces another ordered relation $[Z, >_Z]$ where $Z = X \cup Y$, $X \cap Y$, or $X - Y$ as the case maybe. For union and intersection $>_Z$ contains all the orderings among the tuples of Z that are consistent with both $>_X$ and $>_Y$. For set-difference, $>_Z$ includes all orderings among tuples of Z that are consistent with $>_X$ (see [13] for examples). These definition extend to any combination of ordered and

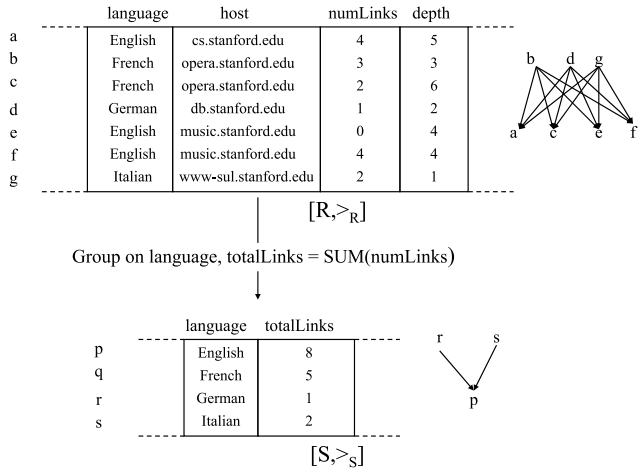


Figure 3: Group-by operator on an ordered relation

plain relations by setting one or both of $>_X$ and $>_Y$ to be empty.

Set operations with ranking. When a ranked relation $[R, f]$ is supplied as an operand to a union, intersection, or set-difference operator, $[R, f]$ is replaced by the corresponding ordered relation $[R, >_f]$[3]. Thus, set operations do not preserve or operate on actual rank values, only on orders induced by the ranks.

4.1 Group-by (γ)

Ranked relation. Group-by on a ranked relation $[R, f]$ merely treats the ranking attribute $R.f$ as yet another attribute that can be grouped, aggregated, or dropped. When $R.f$ is used as a grouping attribute, the result is simply another ranked relation $[S, f]$ with the same ranking attribute. If $R.f$ is aggregated (using a function such as AVG or MIN), the result is a ranked relation $[S, g]$ where the g-values are simply aggregations of f-values over each group. Finally, if $R.f$ is neither grouped nor aggregated, the rank values are lost and the result is a plain relation.

Ordered relation. To extend group-by to ordered relations, we must define how to order the tuples of the result (i.e., how to order the "groups") given the order relationships in the operand. The key is to ensure that the ordering of the result continues to be a partial order. We use the following rule: Suppose a group-by on $[R, >_R]$ yields $[S, >_S]$. Given tuples $x, y \in S$, we set $x >_S y$ iff for each tuple $t_1 \in R$ belonging to the group corresponding to x and each tuple $t_2 \in R$ belonging to the group corresponding to y, $t_1 >_R t_2$.

Figure 3 illustrates the application of this rule on an ordered page relation $[R, >_R]$. R represents various attributes of 7 different pages, all of which point to a website WS of interest to the user. The attribute $R.numLinks$ represents the number of links from each of these 7 pages to pages in WS. The partial ordering $>_R$ is used to express the following preference: "prefer pages with depths ≤ 3".[4] Thus, $b >_R c$, $g >_R e$, etc., as shown in the Hasse diagram in the figure. The figure shows how the incoming links are grouped based on the language of the page in which the links occur. The ordering of the result tuples p, q, r, and s is computed using the rule described above. For instance, since $d >_R a$, $d >_R e$, and $d >_R f$, (all the German pages are at depths less than 3 but all English pages are at higher depths), we set $r >_S p$ (the "German" group is preferred to the "English" group).

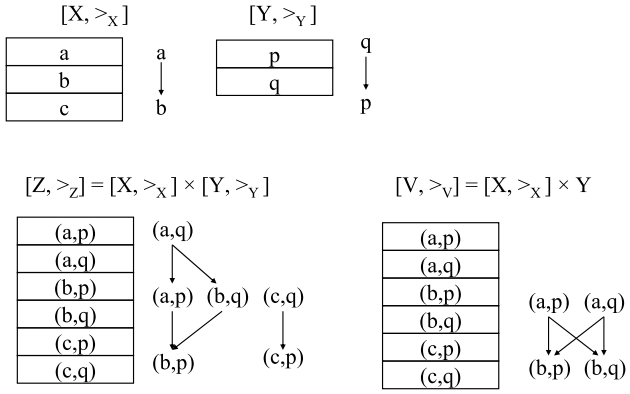


Figure 4: Cross product of ordered relations

4.2 Cross-product ($\times$)

Cross-product operations can involve any pair of plain, ranked, or ordered relations. The challenge is to define the ordering or ranking of the result for each possible combination of operands. The rules we set out below are based on the following intuition: if only one of the operands is ranked/ordered, the result must reflect the ranking/ordering in this operand; when both operands are ordered, the ordering in the result must be consistent with both operand orderings. Thus, we separately consider the following cases:

- *Case 1: Both operands are plain relations.* Standard definition of cross-product
- *Case 2: Both operands are ordered relations.* We define $[X, >_X] \times [Y, >_Y] = [Z, >_Z]$, where:
  - $Z = X \times Y$
  - If $a >_X b$ and $c >_Y d$, then $(a, c) >_Z (b, d)$
  - If $a >_X b$, then $(a, c) >_Z (b, c)$ for any $c \in Y$
  - If $c >_Y d$, then $(a, c) >_Z (a, d)$ for any $a \in X$

  Figure 4 illustrates this definition. For example, $(a, q) >_Z (b, p)$ since $a >_X b$ and $q >_Y p$, $(c, q) >_Z (c, p)$ since $q >_Y p$, and so on.
- *Case 3: One operand is plain, other is ordered.* We define $[X, >_X] \times Y = [V, >_V]$, where $V = X \times Y$ and $(a, c) >_V (b, d)$ iff $a >_X b$. The bottom right relation in Figure 4 illustrates this definition. Since $a >_X b$, every tuple in the result generated using a is $>_V$-related to every tuple generated using b.

[3] Recall from Section 3 that for tuples $a, b \in R$, $a >_f b$ iff $a.f > b.f$.

[4] The depth of a page is the minimum number of links that must be traversed to reach that page, starting from the root page of the website to which the page belongs.

	pageID	pLanguage	pDomain
a	185	English	airfrance.com
b	292	French	paris.org
c	103	French	paris.org
d	849	German	tagesspiegel.de
e	551	English	ibiblio.org
f	300	English	airfrance.com

R

$$\Phi_{pDomain \text{ LIKE } \%.com > pDomain \text{ LIKE } \%.org}(R) = [R, \{a > b, a > c, a > e, f > b, f > c, f > e\}]$$

$$\Phi_{(pLanguage = English \wedge pDomain \text{ LIKE } \%.org) > pLanguage \neq English}(R) = [R, \{e > b, e > c, e > d\}]$$

Figure 5: The Φ operator

- *Case 4: One operand is plain, other is ranked.* Analogous to Case 3, we define $[X, f] \times Y = [V, g]$ where $V = X \times Y$ and $g((a, b)) = f(a)$, i.e., ranks are purely determined by f.
- *Case 5: One operand is ranked, other is ordered.* To compute $[R, f] \times [S, >_S]$, the ranked relation is converted to the corresponding ordered relation $[R, >_f]$ and Case 2 is used to compute $[R, >_f] \times [S, >_S]$.
- *Case 6: Both operands are ranked.* Both operands are converted to the corresponding ordered relations and Case 2 applies.

Note that in Cases 5 and 6, the cross-product operation does not use the rank values even though the ordering induced by the ranks is taken into account. However, if there is a need to preserve or operate on the actual ranks, the compose operator (see Section 4.3) can be employed.

4.3 Ranking and ordering operators

In the previous section, we extended the semantics of traditional relational algebra operators to handle ordered and ranked Web relations as operands. In this section, we define four new operators specifically designed for creating and manipulating ranked and ordered relations.

Rank (Ψ). Operator Ψ simply formalizes the act of applying a ranking function to a base relation. Thus, given a relation R and ranking function $f : R \times \{R\} \rightarrow [0, 1]$, we define $\Psi(f, R) = [R, f]$.

Compose ($\Theta_{h,op}$). The compose operator Θ is used to merge two ranked relations to produce another ranked relation. Each instance of Θ is associated with a "composition function" h, that defines how ranks are assigned to the output relation, and a binary set operator op ($op \in \{\cup, \cap, -, \times\}$) that defines how the tuples of the resulting relation are constructed. The composition function assigns a new rank for each tuple of the result, using the ranks of all the tuples in the operand relations. We provide several examples of rank composition in Section 5.

Order (Φ). The Φ operator constructs an ordered relation, given either a ranked relation or a plain base relation. When applied on a ranked relation, $\Phi([R, f])$ returns the corresponding ordered relation $[R, >_f]$ (recall that $>_f$ is the ordering induced by the rank values in $R.f$, i.e., for $a, b \in R$, $a >_f b$ iff $a.f > b.f$).

To apply Φ to a plain relation, we specify an "ordering condition". An ordering condition on R is an expression $C_1 > C_2$ where C_1 and C_2 are any two valid selection predicates on R. We define $\Phi_{C_1 > C_2}(R) = [R, >]$, where $a > b$ iff $a \in \sigma_{C_1}(R) - \sigma_{C_2}(R)$ and $b \in \sigma_{C_2}(R) - \sigma_{C_1}(R)$. In other words, tuples that satisfy C_1 are preferred to those that satisfy C_2. However, tuples which satisfy both conditions are removed from the ordering. For convenience, we will interpret Φ_C as being equivalent to $\Phi_{C > \text{NOT C}}$.

Figure 5 shows two instances of using the Φ operator. The first ordering condition orders all pages from ".com" domains ahead of pages from ".org" domains. The second ordering condition orders English pages from ".org" domains ahead of pages in non-English languages.

Prune (Ω_k). The prune operator provides a mechanism for retrieving a fixed-size subset of tuples from a relation (refer to [13] for formal definition). In particular, given a relation R, $\Omega_k(R)$ selects a subset of size $\min(k, |R|)$. If R is a plain relation, the operator can non-deterministically choose any subset of this size. When applied to a ranked relation, $\Omega_k([R, f])$ returns a ranked relation containing the k top ranked tuples in $[R, f]$. Since multiple tuples of R may have the same f values, several top-k sets are possible and the actual result is non-deterministically chosen from among them.

Finally, when applied to an ordered relation $[R, >]$, Ω_k selects tuples on the basis of this ordering. For example, consider the ordered relation $[R, \{a > b, a > c, a > e, f > b, f > c, f > e\}]$ shown in Figure 5, corresponding to the preference for ".com" domains over ".org" domains. Ω_1 on this relation can non-deterministically return any one of the three tuples a, f, or d. However, Ω_1 cannot return b, c, or e since they are all ordered below a and f. Similarly, Ω_4 on this relation can yield any set of four tuples as long as at least a and f are part of the result (thus, 6 possible results). In all cases, the result of the operator is also an ordered relation and the ordering of tuples in the result is simply the ordering in $>_R$ restricted to the available tuples. Thus, one possible result of applying Ω_4 is $[\{a, f, e, d\}, \{a > e, f > e\}]$.

4.4 Navigation operators

Graph navigation in complex Web queries tends to be simple in structure, often focusing purely on connectivity and neighborhood properties (e.g., which pages point to page X, which pages does X point to, what are the pages that are at most 2-clicks away from X, how many links interconnect two sets of pages, etc.). This contrasts sharply with the sophisticated operators and path expressions used in navigating and branching through the label structure of semi-structured database graphs [6]. The reason is that the size and immense heterogeneity of Web data sets (and Web graphs) makes it very hard to formulate precise path queries. Often, when navigating Web graphs, the ability to "bias" the choice of links and pages (e.g., prefer intra-host

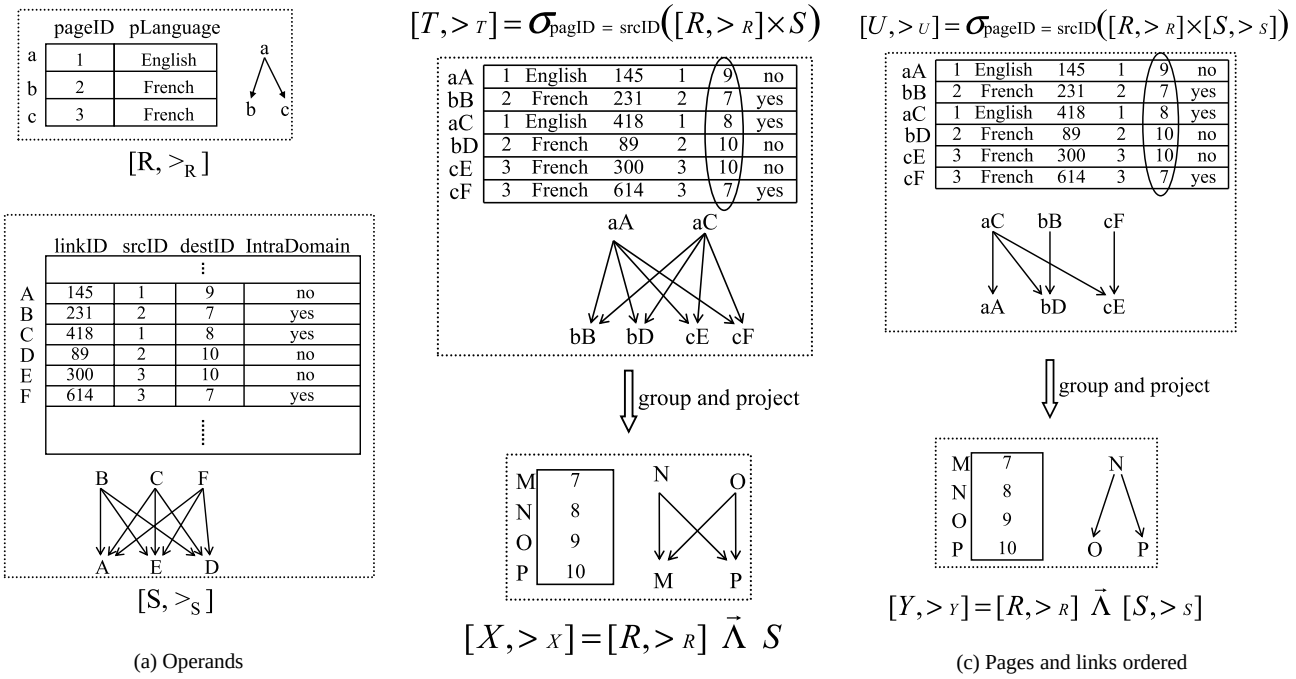


Figure 6: Navigation in the presence of ordering

links to inter-host links, prefer links from pages that link to `www.stanford.edu`) is more useful than the ability to precisely specify paths using complex regular expressions. Further, even if complex path queries could be formulated, their execution on massive Web graphs[5] would be prohibitively expensive.

Thus, the key challenge is to meaningfully propagate user-defined ranks and orders when navigating through the Web graph. In our model, navigation operators are expressed in terms of cross-product and group-by operations involving page and link relations. Hence, the semantics of navigation in the presence of ordering and ranking derive from the semantics of the operators defined earlier. We will first define how ranks and orders propagate through a "single-step" navigation operation, i.e., following exactly one link from a set of pages to reach another set of pages. Later, we will extend the definition to paths in the Web graph.

We use the symbols $\overrightarrow{\Lambda}$ and $\overleftarrow{\Lambda}$ to represent forward and backward navigation respectively. Forward navigation follows links in the Web graph in the direction of the actual hyperlinks whereas backward navigation is in the opposite direction. Our description will focus on $\overrightarrow{\Lambda}$ but the details for $\overleftarrow{\Lambda}$ are similar.

Operator $\overrightarrow{\Lambda}$ accepts a page relation (say R) and a link relation (say S) as operands. One or both of R and S may be plain, ordered, or ranked. $\overrightarrow{\Lambda}$ computes the set of pages reachable in 1-step from the pages in R by following any of the links in S in the forward direction. Recalling the definition of page and link relations, R and S must have exactly one attribute with domain $\mathcal{I}_p$ and $\mathcal{I}_l$ respectively. Let those attributes be $R.pageID$ and $S.linkID$. For convenience, we will assume that the link relation also includes source and destination page identifiers for each tuple, with attribute names $S.srcID$ and $S.destID$. We define,

$$R \overrightarrow{\Lambda} S = \Pi_{S.destID}(\gamma_{S.destID}(Z)), \text{ where}$$
$$Z = \sigma_{R.pageID=S.srcID}(R \times S), \text{ and}$$

Navigation with ordering.

The formula for $\overrightarrow{\Lambda}$ involves a join on the page and link relations followed by grouping and projection. To illustrate the application of this formula in the presence of ordering, we will take up two examples: one involving ordering only on the pages and another involving ordering on both pages and links. Figure 6(a) shows the two operands that we will use in our example; a page relation $[R, >_R] = \phi_{pLanguage=English > pLanguage \neq English}(R)$ with a preference for "English" pages and a link relation $[S, >_S] = \phi_{IntraDomain=yes > IntraDomain=no}(S)$ with a preference for intra-domain links.[6]

(i) Ordering only on pages. For this example, we will ignore the ordering on the links (i.e., ignore $>_S$) and compute $[X, >_X] = [R, >_R] \overrightarrow{\Lambda} S$ as shown in Figure 6(b).

[5] For example, a 110 million page Web data set translates to over a billion edges in the Web graph [12].

[6] A link from *cs.stanford.edu* to *db.stanford.edu* is within the stanford.edu domain but a link from *cs.stanford.edu* to *www.cnn.com* is not intra-domain.

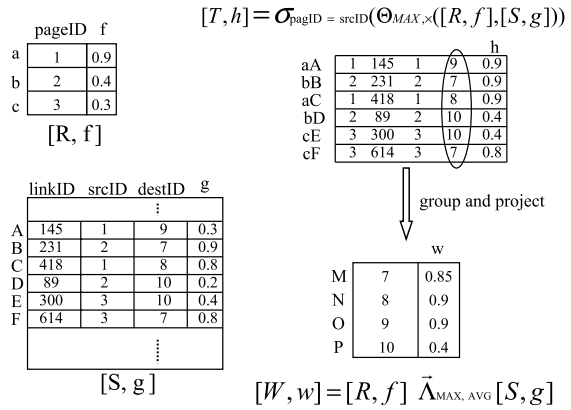


Figure 7: Navigation in the presence of ranking

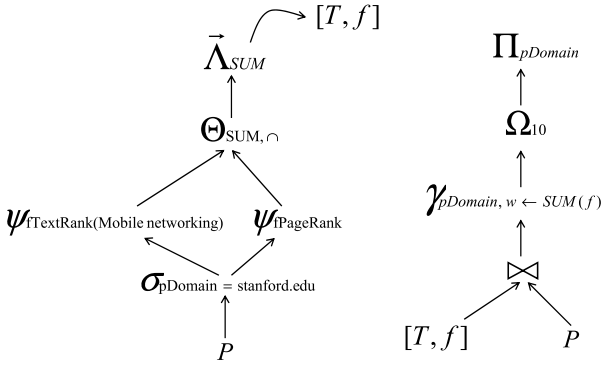


Figure 8: Query graph for Example 1

Relation $[T, >_T]$ in Figure 6(b) represents the intermediate step of joining the page and link relations. The final result $[X, >_X]$ is produced by grouping and projecting out the destination page identifiers (the circled column) from $[T, >_T]$. Applying the rules for cross-product from Section 4.2, since $a >_R b$ and $a >_R c$, all tuples of T generated using a are ordered ahead of the remaining tuples. Similarly, the ordering $>_X$ is derived by applying the rules for group-by from Section 4.1. For instance, $N >_X M$ because $aC >_T bD$ and $aC >_T cE$.

We note that the ordering in the final result matches our intuition, given the preferences expressed in $>_R$. For instance, inspecting S and $[R, >_R]$, we see that pages 8 and 9 are pointed to by page 1, pages 7 and 10 are pointed to by pages 2 and 3, and $>_R$ expresses a preference for 1 over 2 and 3. Thus, we would expect pages 8 and 9 to be ordered ahead of 7 and 10, which is precisely what is represented in the diagram for $>_X$.

(ii) Both pages and links ordered. For this example, both $>_R$ and $>_S$ are taken into account. So we compute $[Y, >_Y] = [R, >_R] \overrightarrow{\Lambda} [S, >_S]$) as shown in Figure 6(c). As before, the intermediate result $[U, >_U]$ represents the join of the page and link relations which is then grouped to yield $[Y, >_Y]$. Once again, the ordering $>_Y$ computed using our formula can be intuitively explained by looking at the operand relations. For instance, $N >_Y O$ (page 8 preferred to 9) is explained by the fact that though page 1 points to both 8 and 9, the $1 \rightarrow 8$ link is an intra-domain link (tuple $C \in S$) whereas the $1 \rightarrow 9$ link is not (tuple $A \in S$).

Navigation with ranking.

When page and link relations are ranked, the navigation operator accepts two aggregation functions as parameters ($\overrightarrow{\Lambda}_{a,b}$): a defines how to combine page and link ranks and b specifies how to aggregate all the incoming ranks (since multiple links may may point to the same target page). For instance, Figure 7 illustrates the computation for $[R, f] \overrightarrow{\Lambda}_{max,avg} [S, g]$. As in the earlier examples, the figure shows the intermediate step of joining the page and link relations to yield $[T, h]$ (note that since ranks must be preserved, we use the compose operator $\Theta_{max,\times}$ rather than plain cross-product). The max function is applied when computing $[T, h]$ (e.g., $h.aC = \max(a.f, C.g) = \max(0.9, 0.8) = 0.9$) and the avg function is applied when grouping to generate the final result (e.g., $M.w = (bB.h + cF.h)/2 = 0.85$).

Unary navigation operators.

For a page relation R, we define $\overrightarrow{\Lambda}(R) = R \overrightarrow{\Lambda} \mathcal{L}$, and $\overleftarrow{\Lambda}(R) = R \overleftarrow{\Lambda} \mathcal{L}$, where $\mathcal{L}$ is the universal link relation. Thus, instead of choosing the links from a set of tuples in a link relation, the unary navigation operators permit navigation using all available links in the repository. These unary operators are useful for the common scenario of computing the out-neighborhood $\overrightarrow{\Lambda}$ (R) and in-neighborhood $\overleftarrow{\Lambda}$ (R) of a set of pages. As with the binary operator, when R is ranked, the unary operator includes an aggregation function to indicate how to combine multiple incoming ranks (e.g, $\overrightarrow{\Lambda}_{MAX}(R)$ indicates that each page in the out-neighborhood of R will receive a rank equal to the maximum ranks of all the pages in R that point to it).

Further, since the result of $\overrightarrow{\Lambda}$ (R) is itself a page relation, we can recursively apply navigation on the result to compute expanding out and in-neighborhoods. Thus, we define $\overrightarrow{\Lambda}^i(R) = \overrightarrow{\Lambda}(\overrightarrow{\Lambda}^{i-1}(R) - R), \forall i > 1$. Notice that if R is ordered or ranked, each neighborhood will also be correspondingly ordered or ranked. Thus, our unary navigation operators provide a natural extension to simple unordered and unranked path navigation in Web graphs.

5 Examples of Complex Queries

In this section, to illustrate how the various operators in our algebra work together, we take up two sample Web analysis tasks and construct the corresponding queries. Queries for several other tasks, including Example 2 from Section 1, are described in [13].

We will begin with Example 1 described in Section 1. Figure 8 shows the query graph for this analysis task, with each node representing an operator in our algebra. For convenience, we have split the query into two pieces connected by the intermediate result relation $[T, f]$. Notice that in the

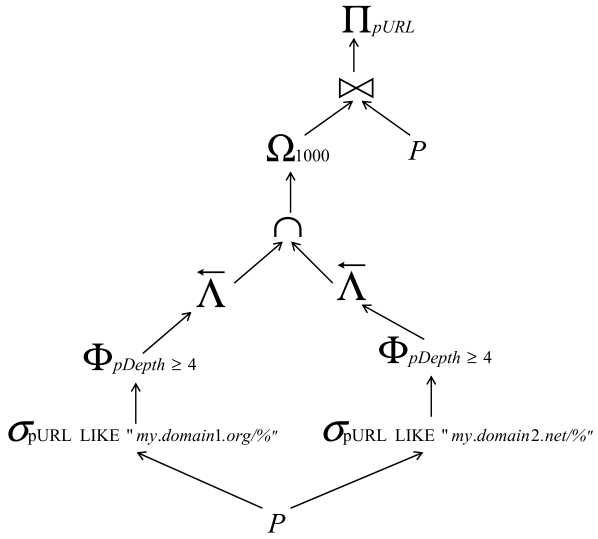


Figure 9: Query graph for Example 3

left piece of the query graph, after selecting "stanford.edu" pages from the universal page relation $\mathcal{P}$, we apply two ranking functions: $fTextRank$ to assign ranks based on a text search for "Mobile networking" and $fPageRank$ to assign ranks based on the PageRank attribute. We compose these two ranks using the Θ operator and compute the out-neighborhood of this ranked set of pages (labeled $[T, f]$). In the second part of the query graph, we join $[T, f]$ with $\mathcal{P}$ to retrieve the domain of each page (the "pDomain" attribute), group on pDomain (summing up ranks within each group as prescribed in Example 1), and finally choose the top-ranked 10 domains using Ω_{10}.

Example 3. In this example, the webmaster of "my.domain1.org" and "my.domain2.net" prepares a list of 1000 pages that point to both his websites. In particular, he is more interested in pages that deep-link into his web sites, i.e., link directly to deeply buried pages instead of linking to the main page or one of the top-level index pages. Figure 9 illustrates the query graph for this example. The query computes the intersection of the in-neighborhood of both websites. The preference for deep links is expressed as a preference for pages with depth ≥ 4. This ordering is automatically converted into an ordering on the neighborhood pages by the semantics of the $\overleftarrow{\Lambda}$ operator. Finally, the URLs of the top 1000 pages, subject to this order, are retrieved.

6 Optimizing and Executing Web queries

As part of the Stanford WebBase repository, we have developed an optimizer and execution engine for efficiently executing complex queries over Web data sets. Our cost-based optimizer is similar in spirit to the query optimizers employed in relational query execution systems. However, certain unique features of Web data sets, the storage structures used in Web repositories, and the characteristics of complex Web queries, pose new and interesting challenges. The discussion in this section is intended to highlight these challenge and provide an overview of our optimization scheme.

As with joins in relational queries, optimization of navigation operations is crucial for efficiently executing complex Web queries. However, the difference is that navigation operators must now be implemented over the specialized structures used to represent Web graphs, rather than as hash joins or sort-merge joins over relational tables. In this section, we will focus our attention on the techniques we have devised to optimize navigation over massive Web graphs. Our techniques are based on the following two key ideas:

Exploit query locality. While a repository may involve several hundred million pages and billions of hyperlinks, most analysis queries focus a relatively small "piece" of the data set (e.g., the piece could be all pages in domains X and Y related to topic Z). Further, as indicated before, navigation is often local and explores only the immediate forward or backward neighborhood of this piece. Thus, many queries are "localized" to one or more small portions of the Web graph. A key feature of our optimizer is an index structure and optimization scheme geared to identify and exploit such locality.

Exploit prune. The presence of the prune operator presents additional opportunities for optimization. Typically, the prune operator is employed as the last operation in a query to retrieve only a subset of the results. An optimizer that can push the "prune" operator down the query tree (ideally, at every stage in the query, generating only as many results as are used later on) can obtain significant speedup in query execution time. However, pushing the prune operator down the tree in the presence of ranking and ordering is a challenging problem.

6.1 Page Clusters

To identify and exploit locality during query execution, we partition the entire data set in the repository into *page clusters*. Each *page cluster* represents a set of pages that belong to the same top-level domain (e.g., all pages in a cluster may belong to stanford.edu), have lexicographically similar URLs, and possess similar out-neighborhoods, i.e., point to almost the same set of pages. Intuitively, we attempt to group together "related" pages so that all the pages relevant to a complex query are distributed among a relatively small number of clusters. We refer the reader to our previous work, reported in [12], for a precise characterization of these page clusters and an iterative algorithm for partitioning a Web data set into clusters.

Below, we briefly describe a representation scheme, based on page clusters, for physically organizing the Web graph for efficient navigation. In the next section, we discuss the role of page clusters in our overall optimization and query execution strategy.

S-Node representation. The S-Node representation scheme uses page clusters to physically organize the Web graph into a two-level structure as shown in Figure 10 [12].

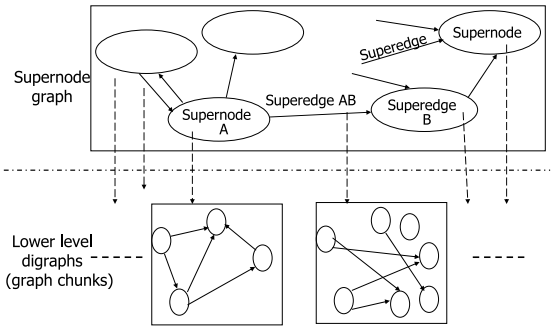


Figure 10: S-Node Representation

The top-level directed graph, called a supernode graph, has one node ("supernode") for every page cluster and an edge ("superedge") from supernode A to supernode B iff some page in cluster A points to some page in cluster B. A supernode A points to a lower-level directed graph that contains one vertex for each page in cluster A and one edge for each link between those pages. Superedge AB points to a lower-level bipartite digraph that represents all the links from pages in cluster A to pages in cluster B. Thus, the lower-level digraphs (called "graph chunks") together represent the entire Web graph and the top-level supernode graph acts as a compact cluster-level structural summary.

Typically, the supernode graph resides in memory whereas the graph chunks are loaded from disk on demand. We maintain two S-Node graph representation structures, one for the Web graph and one for its transpose, to aid in forward and backward navigation respectively.

6.2 Cluster-based optimization and execution

Page clusters and the graph chunks that they generate, are central to our optimization and execution strategy. The cost of each query plan is measured in terms of the estimated number and size of graph chunks that are transferred from secondary storage to main memory. In addition, akin to relational optimizers, we maintain statistics on the value distributions of various pages and link attributes. However, all our statistics are maintained at the cluster level, rather than for the repository as a whole. For instance, instead of a single histogram representing the distribution of PageRank values across the entire repository, we maintain one such histogram for each page cluster.

The search space of possible execution plans considered by our optimizer, and the cost of each of these plans, is primarily dependent on the navigation operators in a query. In the following, we describe how plans are enumerated and their costs computed.

Cost of a navigation operation.

We associate two sets of graph chunks with each navigation operation. The *input chunk set* (ICS) is the set of all graph chunks that must be available for the navigation operation to execute, i.e., all the chunks that may conceivably contain the source pages and links followed by the operation. The *output chunk set* (OCS) is the set of graph chunks that could

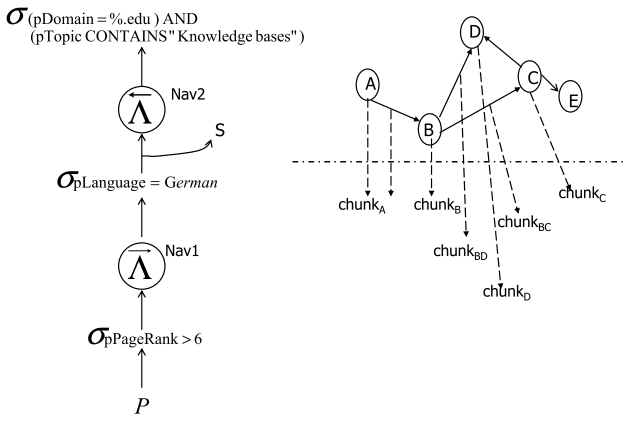


Figure 11: Example to illustrate cost model

conceivably contain the pages produced as a result of the navigation operation. The cost of a navigation operation is simply the sum of the sizes of all the chunks in its ICS.[7]

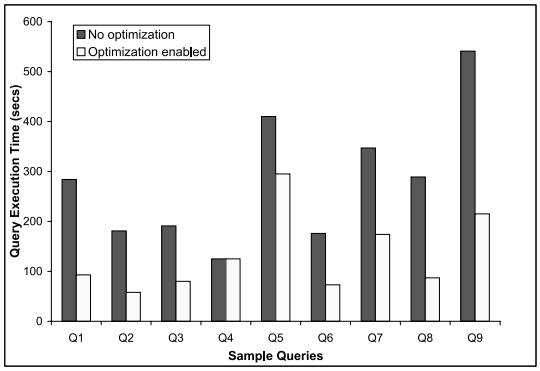
To illustrate, consider the query shown in the left half of Figure 11. The query performs the following computation: "If S is the set of all German pointed to by pages with PageRank > 6, compute a list of all .edu pages about 'Knowledge bases' that also point to pages in S". The two navigation operators in the query are labeled $Nav1$ and $Nav2$. To compute the ICS and OCS for $Nav1$ in Figure 11, the optimizer uses the following pieces of information: (i) the fact that the source pages for this operation must be pages with PageRank > 6, (ii) the fact that destination pages must be German language pages, and (iii) cluster-level statistics for the "pPageRank" and "pLanguage" attributes. For instance, using the cluster-level histograms of PageRank distribution, suppose the optimizer computes that the only page clusters containing pages with PageRank > 6 are A and B and similarly that the only page clusters containing German pages are C, D, and E. The right half of Figure 11 shows a portion of the supernode graph containing all the supernodes and superedges involving A, B, C, D, and E. Since A does not have a superedge connecting it to C, D, or E, (i.e., none of the pages in A point to any German page) and E does not have an incoming superedge from either A or B, the optimizer concludes that $ICS(Nav1) = \{chunk_B, chunk_{BD}, chunk_{BC}\}$. Thus, the estimated cost of this navigation operation is $size(chunk_B) + size(chunk_{BD}) + size(chunk_{BC})$.

Plan Enumeration.

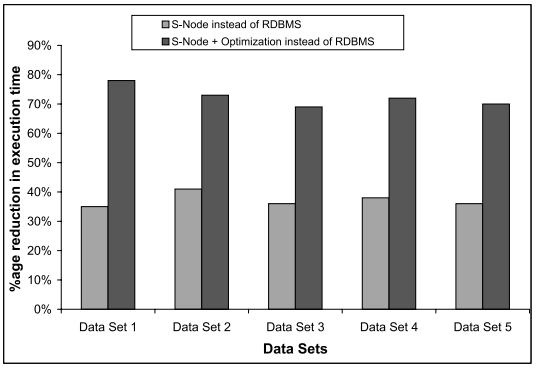
The space of possible query plans is influenced by two factors: the graph used to execute each navigation operator and the ordering among the operators.

Web graph versus Transpose. We observe that every navigation operation, irrespective of its specified direction (i.e., forward or backward), can be executed by either loading graph chunks from the Web graph or by loading graph

[7] We note that to enable cluster-based optimization, we augment the S-Node representation scheme so that each supernode and superedge records the size of the graph chunk that it points to.



(a)



(b)

Figure 12: Experimental results demonstrating the impact of cluster-based optimization

chunks from the transpose of the Web graph. For instance, $Nav1$ in Figure 11 can either be executed by following links from "db.stanford.edu" pages to German pages in the Web graph, or by selecting those German pages which have links to "db.stanford.edu" pages in the transpose of the Web graph. Therefore, the optimizer must compute the ICS, OCS, and cost for both strategies.

Multiple navigation operators. So far, we have treated each navigation operation in isolation when identifying the ICS and computing costs. However, for queries involving multiple navigation operators, the estimated cost of one operator is influenced by the presence of the other. Specifically, consider a query plan for the example in Figure 11 in which both navigation operations are executed using forward navigation, loading only chunks from the Web graph and none from its transpose. Considered on its own, the ICS for $Nav2$ would involve all German pages and would be a superset of $\{chunk_C, chunk_D, chunk_E\}$. However, based on our earlier example, since the OCS of $Nav1$ does not include $chunk_E$, we can safely eliminate $chunk_E$ from $ICS(Nav2)$ as well.

Thus, given a query Q containing a set of navigation operators $\{N_1, \ldots, N_k\}$, the optimizer explores a search space of $(2k)!$ plans (all possible orderings of $\{N_1, \ldots, N_k\}$ with each N_i either using chunks of the Web graph or its transpose), computes the cost of each plan measured in terms of chunk IO, and chooses the minimum cost plan.

Exploiting prune.

For queries involving prune, at every stage, the aim is to avoid generating result elements that later get discarded. We make two changes to the approach described earlier. First, when computing the ICS for each navigation operator, the optimizer produces a sorted list of graph chunks as opposed to an unsorted set. The graph chunks in the ICS are sorted in descending order of their estimated "yield", i.e., the number of output results that they are expected to produce. For example, in Figure 11, each chunk in the ICS of $Nav1$ is sorted based on the number of pages with $PageRank > 6$ that it is expected to yield (using the data from the cluster-level histogram of PageRank distribution). Second, during query execution, all navigation operators are simultaneously scheduled. Thus, akin to pipelining in relational execution engines, each navigation operator requests results from its predecessor in the pipeline. Each operator loads graph chunks from its ICS into memory one at a time in sorted order. When all the results from one chunk have been computed and the successor in the pipeline requests more, the next chunk is loaded.

In the interest of space, we have only introduced the key ideas behind cluster-based optimization in the presence of prune. For details on yield estimation and sorting ICS chunks in the presence of ranking and ordering functions, we refer to the reader to our extended technical report [13].

Experimental Results.

We conducted extensive experiments to measure the impact of our cluster-based optimization approach in reducing navigation times for complex Web queries. In Figure 12, we present a couple of sample results from our experiments.

Figure 12(a) displays the execution time for nine sample Web queries over a 35-million page data set (approximately 600 million links, 300 GB of HTML). Queries Q1 to Q4 correspond to Examples 1 to 4 described earlier in this paper and the remaining queries are described in [13]. For each query, the chart shows execution times with and without cluster-based optimization. In the latter case, the query is executed by simply scanning the query tree bottom-up, always choosing the Web graph for forward navigation and the transpose for backward navigation. Figure 12(a) shows that in most cases, enabling cluster-based optimization resulted in a 40-45% reduction in query execution time.

To generate Figure 12(b), we executed a suite of 30 Web queries over 5 different 20-million page data sets. Each query was executed in three ways: (i) using a relational database to store the Web graph, (ii) using the S-Node representation but without optimization, and (iii) using S-Node with cluster-based optimization. Figure 12(b) plots

the reduction in query execution time by using (ii) or (iii) as opposed to (i). The results indicate that on average, using a cluster-based Web graph representation structure provides a 35-40% reduction in navigation speed. Cluster-based optimization further boosts the reduction factor to the 70-80% range.

7 Related Work

Drawing inspiration graph and hypertext query systems, a number of Web query languages have been developed in the past (e.g., WebSQL, W3QL, WebLog, WebOQL, StruQL, etc.) [7]. However, our work differs from these past approaches in two respects. At the modeling level, none of these languages incorporate the notions of ranking and ordering. At the implementation level, these systems are intended either for "online" queries on the Web or for "Web-site management", as opposed to our "warehouse" model (i.e., crawl, locally build repository, and execute analysis queries).

The Squeal system [15] defines a relational schema to support SQL queries over Web data. However, the system is primarily aimed at relational queries over page and link attributes, without support for ordering, ranking, and navigation.

Theoretical work on computability of Web queries has been addressed in [1, 9]. While the authors of [9] also employ a relational Web data model, they do not include our extended operator set to handle navigation, ranking, and ordering. In addition, their queries are defined using an abstract machine model, analogous to Turing machines, in contrast to our algebraic approach. In [10], the authors study the expressive power of relational algebra when the underlying domains are partially ordered. However, this is fundamentally distinct from our model, in which each individual relation can be partially ordered based on a query-specific ordering condition.

In the context of financial and statistical applications, systems such as SEQUIN [14] and SRQL [5], and more recently AQuery [8] have proposed SQL extensions to incorporate order. However, in all these cases, the intention is to model sequences or linearly ordered data (e.g., calendar dates, sorted list of stock prices). In contrast, our approach is based on partial orders, is intended to model user "preferences", and incorporates a uniform treatment of ranking and ordering.

The prune operator discussed in Section 4.3 is similar to the "STOP AFTER" SQL extension proposed in [3], except that our definition extends to partially ordered relations as well.

Optimization of path queries has been studied in the object database community. However, a key difference is that unlike object graphs, Web graphs do not conform to any well-defined schema. Semi-structured query optimization techniques do not depend on a schema but instead, employ structural and statistical summaries of the graph structure [11]. Our clustered representation scheme is similar in spirit, but is specialized to Web graphs.

8 Conclusion

In this paper, we addressed the problem of formulating and executing complex expressive queries over Web repositories. We showed that the combination of navigation, text search, and relational operations and the ability to manipulate application-defined ranks and orders are key characteristics of Web queries. We presented a query model and algebra that precisely define the semantics of Web queries with these properties. Finally, we discussed some of the optimization techniques that we have devised to efficiently execute such queries over large Web repositories.

References

[1] S. Abiteboul and V. Vianu. Queries and computations on the web. In *Intl. Conf. on Database Theory*, 1997.

[2] R. Baeza-Yates and B. Riberio-Neto. *Modern Information Retrieval*. Addison-Wesley, New York, 1 edition, 1999.

[3] M. Carey and D. Kossman. On saying "enough already!" in SQL. In *Proc. of of the ACM SIGMOD Conf.*, pages 219–230, 1997.

[4] A. Arasu et al. Searching the Web. *ACM Transactions on Internet Technology*, 1(1):2–43, August 2001.

[5] R. Ramakrishnan et. al. SRQL: Sorted relational query language. In *Proc. of the 10th Intl. SSDBM Conf.*, pages 84–95, July 1998.

[6] S. Abiteboul et. al. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kauffman Publishing, San Francisco, 1st edition, 1999.

[7] D. Florescu, A. Y. Levy, and A. O. Mendelzon. Database techniques for the World-Wide Web: A survey. *SIGMOD Record*, 27(3):59–74, 1998.

[8] A. Lerner and D. Shasha. AQuery: An arrable-based approach to array query languages and optimization. Unpublished.

[9] A. O. Mendelzon and T. Milo. Formal models of Web queries. In *Proc. of the 16th ACM Symposium on Principles of Database Systems*, pages 134–143, 1997.

[10] W. Ng, M. Levene, and T. I. Fenner. On the expressive power of the relational algebra with partially ordered domains. *Intl. Journal of Computer Mathematics*, 71:53–62, 2000.

[11] N. Polyzotis and M. Garofalakis. Structure and value synopses for XML data graphs. In *Proc. of the 28th VLDB Conf., Hong Kong*, 2002.

[12] S. Raghavan and H. Garcia-Molina. Representing Web graphs. In *Proc. of the IEEE Intl. Conf. on Data Engineering*, March 2003.

[13] S. Raghavan and H. Garcia-Molina. Complex queries over web repositories. Technical report, Stanford Univ., Feb. 2003. `http://dbpubs.stanford.edu/pub/2003-11`.

[14] P. Seshadri, M. Livny, and R. Ramakrishnan. The design and implementation of a sequence database system. In *Proc. of the 22nd VLDB Conf.*, 1996.

[15] E. Spertus and L. A. Stein. Squeal: a structured query language for the Web. In *9th Intl. WWW Conf.*, 2000.