

Singapore Management University Institutional Knowledge at Singapore Management University

Research Collection School Of Information Systems

School of Information Systems

9-1999

Locating web information using web checkpoints

Aik Kee LUAH

Wee-Keong NG


Ee Peng LIM

Singapore Management University, eplim@smu.edu.sg

Wee Peng LEE

Yinyan Cao

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

 Part of the [Databases and Information Systems Commons](#), and the [Numerical Analysis and Scientific Computing Commons](#)

Citation

LUAH, Aik Kee; NG, Wee-Keong; LIM, Ee Peng; LEE, Wee Peng; and Cao, Yinyan. Locating web information using web checkpoints. (1999). *International Workshop on Internet Data Management (IDM'99), in conjunction with DEXA'99*. Research Collection School Of Information Systems.

Available at: https://ink.library.smu.edu.sg/sis_research/989

This Conference Proceeding Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

Locating Web Information Using Web Checkpoints*

A.-K. LUAH W.-K. NG E.-P. LIM W.-P. LEE Y.-Y. CAO

Centre for Advanced Information Systems, School of Applied Science
Nanyang Technological University, Singapore 639798, SINGAPORE
{awkng, aseplim}@ntu.edu.sg

Abstract

Conventional search engines locate information by letting users establish a single *web checkpoint*. By specifying one or more keywords, users direct search engines to return a set of documents that contain those keywords. From the documents (links) returned by search engines, user proceed to further probe the WWW from there. Hence, these initial set of documents (contingent upon the occurrence of keyword(s)) serve as a web checkpoint. Generally, these links are numerous and may not result in much fruitful searches. By establishing multiple web checkpoints, a richer and controllable search procedure can be constructed to obtain more relevant Web information. This paper presents the design and implementation of permitting multiple checkpoints to facilitate improved searching on the WWW. Web checkpointing is performed as part of the WHOWEDA project.

1 Introduction

Today, business organizations are turning to information on the WWW to assist in their decision making process instead of relying solely on their in-house data warehouse. This has led to the prevalence of search engines like **Yahoo** and **Alta Vista**. Search engines help to locate information on the WWW by providing a list of URLs and a brief summary of each website (corresponding to each URL). This is, sadly, the only information provided by search engines. With the URLs as initial *checkpoints*¹, users manually probe and seek information from there. This process is not only tedious, it is also quite impractical to follow through all the URLs returned by search engine since there could be thousands of them. Such incompleteness in exhaustive searching leaves the user in doubt as to whether a potential website has been missed. Above all, the search process might not be fruitful eventually.

In addition to poor support for comprehensive searching from search engines, there are currently no proper softwares or applications to help manage downloaded information. After a period of

*This work was supported in part by the Nanyang Technological University, Ministry of Education (Singapore) under Academic Research Fund #4-12034-5060, #4-12034-3012, #4-12034-6022. Any opinions, findings, and recommendations in this paper are those of the authors and do not reflect the views of the funding agencies.

¹In this paper, we use a 'checkpoint' to refer to an intermediate point in a traversal sequence. It is not used in the sense of a 'snapshot', as in other computer science domains.

surfing, most users have many folders of downloaded files on their local disk. When the time comes to locate a wanted file, the success of retrieval depends heavily on users' recall capacity. If a user cannot remember where the file is, it is as good as not having the file at all.

As part of the WHOWEDA project [1, 2, 3, 4, 11, 12] that looks into building a warehouse of web data, we propose the following to alleviate the current situation in searching for WWW information and managing downloaded information: First, we propose the use of multiple checkpoints for a more comprehensive WWW search. Presently, users specify keyword $K_1, K_2, \dots, K_n, n \geq 1$, to search engines to obtain a set of initial links. This is a single checkpoint A . Checkpoint A corresponds to a set of WWW documents containing these keywords. By allowing a second checkpoint (Figure 1), say B (which contain keywords $H_1, H_2, \dots, H_m, m \geq 1$) that follows directly after A , we narrow down search results to only documents (corresponding to A) that contains (one or more) links to documents (corresponding to B). Hence, each additional checkpoint constrains search results further to more specific documents. By constructing a collection of useful checkpoints, users can obtain more meaningful search results. The searching of WWW documents that satisfy a set of checkpoints can be automated so that with multiple starting URLs, a more comprehensive search over the WWW is performed.

The second proposal is to create a suitable storage structure to store *structural results* that are obtained via the multi-checkpoint approach so that results can be further queried, in the same manner as relational tables. With a suitable query language or interface, one can access and manipulate stored results like a database system.

Web checkpointing is performed as part of the WHOWEDA project [1, 2, 3, 4, 11, 12]. The Web Warehousing Project at the School of Applied Science, Nanyang Technological University, Singapore, started in July 1997 with the following key objective: To design and implement a warehousing capability that materializes and manages useful information from the Web so as to support strategic decision making. We aim to build a web warehouse containing strategic information coupled from the web that may also inter-operate with conventional data warehouses. The project is named WHOWEDA which stands for **Warehouse of Web Data**.

The rest of this paper is organized as follows: The next section discusses issues in the design of a procedure to incorporate multiple checkpoints for WWW searching. Section 2 describes algorithms for various modules that implement the proposed procedure. In Section 3, we bring the reader through a sample process of querying the WWW using multiple checkpoints. Next, Section 4 addresses some performance issues in the proposed way of searching the WWW. In Section 5, we review existing related works in querying the WWW. We give a brief conclusion to this paper in Section 6.

2 Design Issues

In this section, we investigate design issues in incorporating checkpoints for WWW searching. Searching for WWW information is a typical process with input, analysis, execution and output phases. Let us examine the design issues in each of these phases.

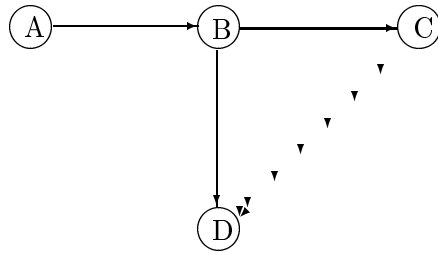


Figure 1: Example of a web query.

2.1 Input Phase: Web Query Specification

Existing search engines provide a well-known WWW for users to input keywords. When multiple checkpoints are desired, and the connectivity of these checkpoints are user-created, a user interface beyond what conventional search engines provide must be designed.

Figure 2 shows several examples of a web query involving multiple checkpoints. Each node represents a checkpoint. An edge from node A to B captures hyperlink (that is desired) from a set of WWW documents (corresponding to A) to another set of WWW documents (corresponding to B). To specify a web query, a user interface must support the following functionalities:

- draw a web query as a directed graph (called *query graph*) consisting of nodes and edges
- specify search conditions (such as keywords) on nodes and edges

We extended the concept of search conditions in our work on checkpointing. While existing search engines use keywords primarily as conditions for searching, we allow other attributes of a WWW document to be specified as well. For instance, we may attach a condition $A.URL = "http://www.whoweda.com"$ to checkpoint A to associate WWW document(s) with URL $http://www.whoweda.com$. (Clearly, there is only one such document.) Other attributes such as document size, date of last update, document title, document type, etc. can be used as conditions on a checkpoint.

2.2 Analysis Phase: Web Query Processing

Given a directed graph representing a web query in the format described above, we need a technique to match and capture WWW documents corresponding to the checkpoints in the query graph and satisfying the topological relationships among the checkpoints. In this section, we look at issues in the processing of a web query.

2.2.1 Finding a Starting Checkpoint

A start node is required in any graph traversal. A web query, however, may not have a well-defined start node. In Figure 2, N_1, N_3, N_4 are potential start nodes for the bottom-left web query. Hence, a technique is required to find suitable start nodes. Below, we present the factors, in order of priority, to be taken into considerations when determining start nodes.

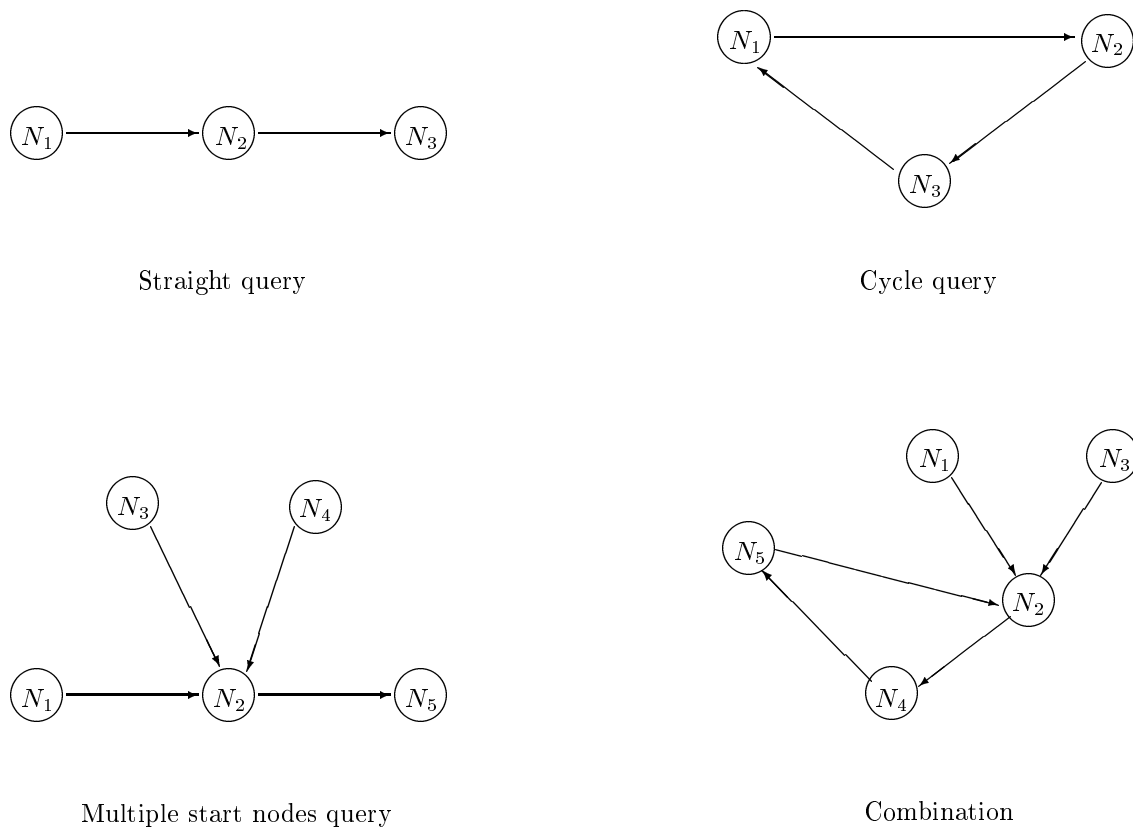


Figure 2: More examples of web queries.

- *Indegree of nodes*
 A node with minimum indegree can be chosen as the start node. Ideally, a start node should be one with zero indegree.
- *Using 'URL' attribute*
 If two or more nodes have equal indegree and outdegree, they are all candidate start nodes, and an ordering must be determined to indicate the order of *evaluating* them. Their URL attributes can be used. A URL like `www.yahoo.com` is easier to access than a specific one like `www.virtual.com/diseases/cancer/treatment.html` because the former has a higher probability of generating more links. Hence, if the latter is evaluated first, then it has a higher probability of failure (not finding any relevant links). As all candidate start nodes must be evaluated, a failure in one start node means the entire query graph results in no WWW documents, thus, eliminating the need to evaluate other start nodes.
- *Using title attribute*
 If the URL attribute does not help in finding an order of evaluation among multiple start nodes, we check their `title` attribute. If two or more nodes are defined with a `title` attribute, they may be evaluated in any order.

- *Using text attribute*

If the `title` attribute does not help in finding an order of evaluation among multiple start nodes, we check their `text` attribute. If two or more nodes are defined with a `text` attribute, they may be evaluated in any order.

If all nodes satisfy the above evaluation criteria to the same extent, the start node can be chosen from among them randomly. Web queries exist in many forms depending on a user's search requirement. There may be web queries with more than one nodes of zero indegree. In this case, we have more than one starting nodes to start the query evaluation process.

2.2.2 Traversal Order

With the start node identified, we may traverse the graph from checkpoint to checkpoint to identify and match WWW documents corresponding to each checkpoint and satisfying the search conditions defined on that checkpoint. A breadth-first or depth-first traversal may be used.

2.2.3 Matching WWW Documents and Checkpoints

We are ready to retrieve information from the WWW that satisfies query graph. If a start node's URL is known, then the corresponding document is retrieved and examined to determine the corresponding document(s) for adjacent checkpoints in the graph. In the retrieval process, existing search engines are used to obtain the set of WWW documents corresponding to start node(s) whose URL(s) are unknown, that is, the search condition does not use the URL attribute.

We may not want to rely on a single search engine to identify the initial documents as it may not have a good index of the WWW documents that are of our interest. Therefore, we can widen the range of documents from different search engines that better match the search conditions in the web query.

The search engine we have experimented is `METAFind` (<http://www.metafind.com>). It is chosen because it submits keyword search request to a few popular search engines such as AltaVista, Lycos, and WebCrawler simultaneously. The results obtained from `METAFind` are formatted into a single page. This simplifies the task of having to visit all result pages.

When a set of candidate documents for a checkpoint (say A) is obtained, each document is analyzed as follows: A parser extracts hyperlinks from the document. These links are checked against the search conditions specified on the out-edge(s) of A . If a hyperlink satisfies the condition on an out-edge $E = (A, B)$, then checkpoint B (which is adjacent to A) will be examined next.

Note that if a user does not specify any search condition on a node, the node becomes *unbound*, that is, any document on the WWW may potentially *satisfy* the node. Similarly, a edge without search condition is unbound. Unbound nodes and edges in a query graph are useful when user is unsure of the exact hyperlink structure of a collection set of documents. In Figure 1, a user may simply let the edge between checkpoint A and B be *unbound*, so that the user can retrieve documents satisfying B through some sequences of hyperlinks from documents satisfying A without knowing exactly what the sequences are. Such imprecision is useful when searching on the WWW as nobody has a complete picture of the WWW's hyperlink connectivity.

```

Inputs :    N := set of nodes in the web query
              A, B, C, D  $\subseteq$  N

Algorithm body :
  A := {x  $\in$  N | indegree(x) is minimal in N}
  if sizeof( A ) = 1
    start node := x
  else
    B := {y  $\in$  A | y.url != NULL}
    if sizeof( B ) = 1
      start node := y
    else if sizeof( B ) > 1
      C := EvaluateURLs( B )
      start node := max_url_depth( C )
    else
      D := {z  $\in$  A | z.title != NULL}
      if sizeof( D ) = 1
        start node := z
      else if D =  $\emptyset$ 
        start node := first node of A
      else
        start node := first node of D
      endif
    endif
  endif
endif

Output :   start node

```

Figure 3: Algorithm for finding a starting checkpoint.

2.2.4 Materializing Search Results

If a query graph is successfully evaluated, a set of graph instances will be obtained. Each of these instances is a graph of WWW documents satisfying the search conditions of the query graph. Hence, each of these instances has a similar topology to the query graph. We refer to an instance as a *web tuple*. The resultant set of instances can be materialized into a *web table* to be access and manipulated further, in the same manner as relational tables.

As new websites are added everyday and existing web pages are modified, the WWW is constantly expanding and changing. Hence, the set of graph instances for a query graph is potentially large. When stored as a table, we need good storage and indexing techniques and strategies so that the contents can be accessed efficiently. Furthermore, existing web tables must be refreshed so that changes in source documents are reflected in the local copy of web tables.

2.3 Execution Phase: Algorithm Description

Having examined the issues in the incorporation of checkpointing for WWW search, we now describe various algorithms for processing different modules of the overall checkpointing procedure.

2.3.1 Finding a Starting Checkpoint

As discussed in Section 2.2.1, there are a few attributes that we examine to determine the start node. The algorithm for getting the start node is shown in Figure 3. The algorithm attempts to look for a node with minimum indegree (preferably zero). If there are more than one nodes satisfying the condition, it looks at the URL attribute of the node. The URL attribute will be evaluated based on the directory depth. If the list of nodes with minimum indegree does not have any URL attribute, the algorithm examines the title attribute. The title attribute is not evaluated as it does not contribute to the performance of the evaluation process.

2.3.2 Matching WWW Documents and Checkpoints

The algorithm is shown in Figure 4. Before the matching begins, some initializations are performed. The start node is determined and a connection is established with the proxy server in order to retrieve web documents. Then matching begins. First, a set of web document(s) are retrieved from the WWW. If a URL has been defined for the start node, only one document will be retrieved. The retrieved document(s) for the start node are then processed and stored. (Additionally, we handle the case of multiple start nodes as discussed earlier. For each of the nodes with zero inlinks, we derive a set of WWW document(s) corresponding to the node. These document(s) will be processed and stored.)

After obtaining the set of start documents(s), the algorithm is now ready to traverse the query graph. For each start document obtained, the query graph will be traversed once. The algorithm returns a set of WWW documents that satisfy the query graph. An empty set indicates a failed traversal. With these set of WWW documents, a cleanup module (see below) is invoked to remove invalid links in the documents. An invalid link arises as a result of retrieving an invalid document corresponding to that link. For example, suppose a document D has three links that satisfy some edge conditions. In the course of evaluating the target document of the link, only two out of the three links satisfy the edge conditions. Therefore, an invalid link is present in document D , and it should be removed.

The `dfsTuple` module is invoked for each initial documents in the multiple start nodes case. Eventually, the algorithm finds a set of WWW documents for the web query and stores them in a web table.

2.3.3 Query Graph Traversal

The algorithm is shown in Figure 5. The starting document is provided by the calling module. Together with the traversal order, the algorithm retrieves documents from the WWW that satisfy the web query. With the initial set of starting documents, it begins by evaluating links in the documents. The algorithm checks whether a document has more than one outlinks; if so, it ensures


```

Inputs :    Q := web query

Variables : S,R,Z := set of materialized node instances
              T := set of nodes of Q in order of traversal
              W := web table

Algorithm body :
start node := GetStartNode( Q )
T := GetTraversalOrder( start node, Q )
S := GetNodeset( start node )
for i := 1 to n
  if indegree(  $N_i$  ) = 0
     $Z_i$  := GetNodeset(  $N_i$  )
  endif
endfor
for s := 1 to n
  R := TraverseQuery( Q, T,  $S_s$  )
   $R_0$  := Cleanup( R,  $S_s$  )
  for z := 1 to n
     $R_z$  := Cleanup(  $R_{z-1}$  )
  endfor
  if  $R_n \neq \emptyset$ 
    StoreNodes(  $R_n$ , W )
  endif
endfor
if content( W ) = empty
  FormTuples( W )
endif

Output :    W

```

Figure 4: Algorithm for matching WWW documents with query checkpoints.

that all the outlink conditions are satisfied before it proceeds. If the current document corresponds to a terminating checkpoint in the web query, then all the links from it are removed. A terminating node is identified as one with zero outlink. If the current evaluation results in an empty set of links, the document will be deleted as an indication that it does not satisfy the web query. Otherwise, the URL of the links will be inserted into a pool that holds the URLs of all subsequent documents to be evaluated. Before inserting the URL into the pool, a check is also performed to ensure that the URL has not been visited in the traversal.

2.3.4 Cleaning Up Nodes

After traversing through all the nodes in a query graph, a preliminary set of web tuples (graph instances) are formed. However, these tuples are not stored immediately during the traversal since we do not know whether any links of a document would still be valid after the evaluation. Therefore,

Inputs : Q := web query
 L := set of links in Q
 s := start node for the query
 T := a set of nodes of Q in order of traversal

Variables : E,R := set of materialized node instances

Algorithm body :

```

for i := 1 to n
  E := GetNodeSet(  $T_i$  )
  for j := 1 to m
    if outdegree(  $T_i$  ) > 1
      evaluate  $E_j$  on all outlinks
      if  $\exists$  a link that fails
        E := E - { $E_j$ }
      endif
    endif
    if outdegree(  $T_i$  ) = 0
      remove all links in  $E_j$ 
      R := R  $\cup$  { $E_j$ }
    endif
    for k := 1 to p
      if LeftHandSideOf(  $L_k$  ) =  $T_i$ 
        evaluate  $E_j$  on  $L_k$ 
        if sizeof(  $E_j$  linkset ) = 0
          E := E - { $E_j$ }
        else
          add url of links to URLPool
          R := R  $\cup$  { $E_j$ }
        endif
      endif
    endfor
  endfor
endfor
if T =  $\emptyset$ 
  R :=  $\emptyset$ 
endif

```

Output : R

Figure 5: Query graph traversal algorithm.

a second traversal is performed to determine the valid documents that can be formed into a tuple. The traversal is now performed in a depth-first manner. The algorithm is shown in Figure 6.

This module uses a recursive approach to determine whether the links in each document leads to another valid document. If the link points to an invalid or deleted document, the link is removed. If this results in an empty set of links; and the document is not a terminating document, the document will be marked for deletion. Since the action is recursive, it will replicate the removal of links and nodes all the way to the starting point. That is, if there is a valid node for the beginning of a web query, but there does not exist a valid node for the ending node in the web query, all the node coupled will be removed and this will result in an empty webtable, i.e., no tuple are formed.

3 WWW Searching with Multiple Checkpoints—A Walkthrough

In this section, we illustrate the use of multiple checkpoints for searching WWW information; from the creation of a web query to the display of results. To materialize information from the web, web query must first be created. A graphical user interface is implemented to provide this need.

3.1 Graphical User Interface

The graphical user interface (Figure 7) for WWW searching using checkpoints is a simple drawing pad which allows user to create web queries using the predefined stencil. The GUI provides a set of common features that are typical in conventional applications. These features are provided through the use of a menu bar and a tool bar. A status bar is also required to provide visual feedback to the user on the current status of the execution.

The features required by the GUI for WHOWEDA can be broadly categorized as *file manipulation* and *drawing operations*. File manipulation includes opening a new or existing web query, saving and closing the web query. Drawing operations includes insertion and deletion of nodes and links, moving nodes around, and undoing the previous action.

The menu bar provides an interface for the user to execute all the commands supported through the GUI. Menu options available are grouped according to their operation. They include a **File** option for the file manipulation, an **Edit** option for creating and editing the web query, a **Tools** option to insert and delete nodes and links; and a **About** option to show the status on the development of WHOWEDA.

The tool bar contains icons of tasks that are performed frequently. The features provided in the tool bar are File New, File Open, File Save, Undo, Redo, Select, Move, Add Node, Add OR Link, Add AND Link, Delete Node, Delete Link and Execute.

A status bar is placed at the bottom of the interface to display information about the status of the program. It can also be used to show short and simple instruction to guide the user in using of the interface. Information like the next action to be taken, position of the mouse cursor and the execution status could be displayed on the status bar.

```

Inputs :   s := materialized node instance
            R := set of materialized node instances
            T := set of materialized node instances processed by this function

Variables : N := materialized node instance
              status := boolean variable

Algorithm body :
  if outdegree( s ) = 0
    T := T  $\cup$  {s}
    status := true
  else if s  $\in$  T
    status := true
  else
    status := false
  endif
  if status = true
    L := linkset of s
    for i := 1 to n
      N := node pointed by  $L_i$ 
      if state( N ) = valid
        T := T  $\cup$  {N}
        if Cleanup( N ) = false
          L := L - { $L_i$ }
        endif
      endif
    endfor
    if L =  $\emptyset$ 
      T := T - {N}
      status := false
    else
      status := true
    endif
  endif
Output :  status

```

Figure 6: Algorithm for cleaning up of nodes

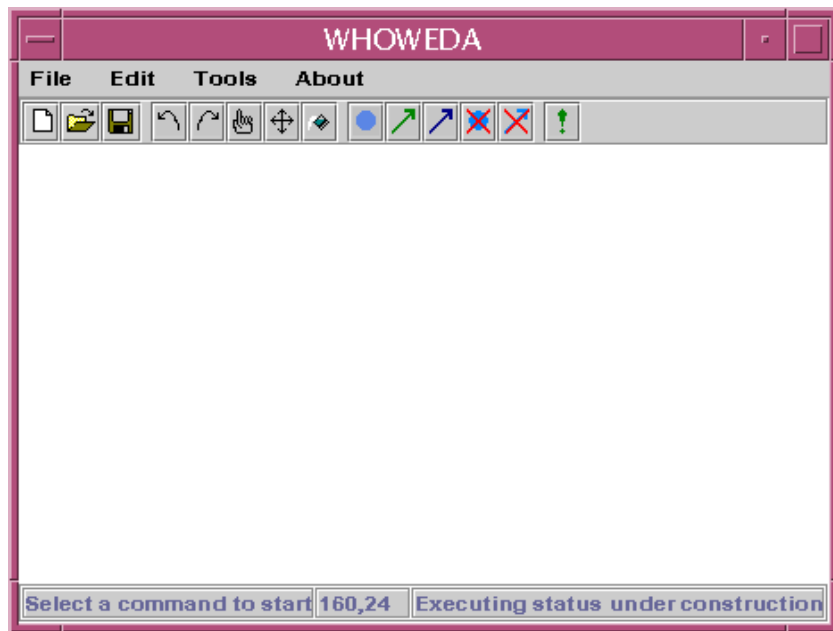


Figure 7: Graphical user interface for query specification.

3.2 Painting a Web Query

Suppose we would like to find information on web computing from the WWW. We can start our search by probing computer science departments worldwide and look at the papers that they have published. There are websites that maintains such a list. Therefore, we can define a checkpoint with this website as the URL. An example is the site

`http://sunsite.doc.ic.ac.uk/bySubject/Computing/UniCompSciDepts.html`.

With the list of institutions, we can go to the individual websites, which can be represented by an unbound checkpoint.

Now, we need a link to join the two checkpoints. This link is unbound since we would like to have a more extensive search of all the institutions available. At the website of these institutions, there is a likelihood of two links that we can use to probe further in order to find the information we need. We can follow the link that leads to the research projects undertaken by the department; or we can go to the faculty directory of the department. At this node, it is very likely that we would get a list of publications belonging to their research projects or staff. In order to reduce the amount of redundant results, we can make sure that the publications of interest obtained from the links given by the research project node and that of the faculty checkpoint is the same. This is achieved by creating two links, one from the research project checkpoint and the other from the faculty checkpoint, to point to the publication checkpoint.

Figure 8 shows the final representation of this web query. In the interface, only the node label is shown to avoid confusion when the query becomes too complex. As shown, there are five nodes

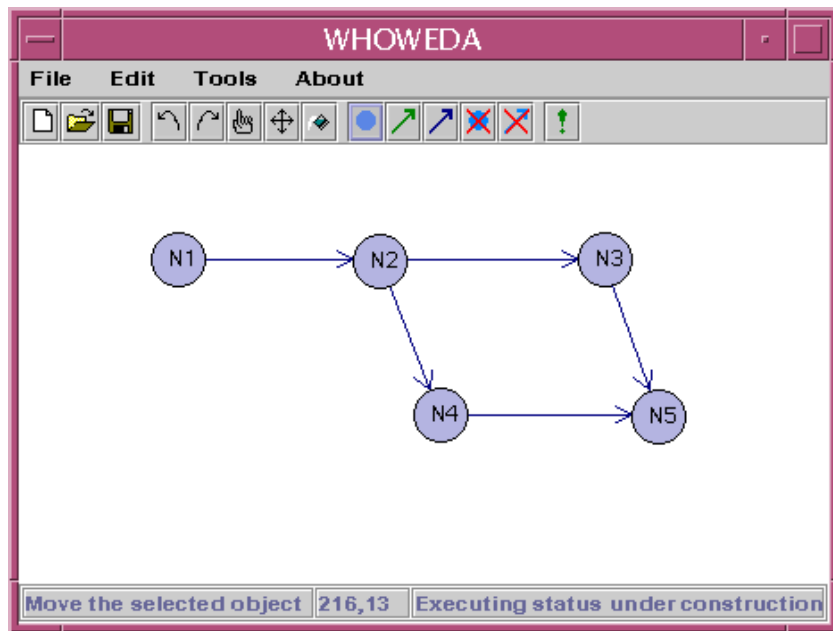


Figure 8: Graphical representation of web query example.

and edges. The search conditions on the nodes and edges are as follows:

```

N1.URL EQUALS "http://sunsite.doc.ic.ac.uk/bySubject/Computing/UniCompSciDepts.html"
N4.TEXT CONTAINS "web computing"
N5.TEXT CONTAINS "web computing"
L2.LABEL EQUALS "staff"
L3.LABEL EQUALS "research projects"
L4.LABEL CONTAINS "publications"
L5.LABEL CONTAINS "publications"

```

Note that N2, N3, L1 are unbound. Dialog boxes are provided to insert search conditions on nodes and edges. Figure 9 and 10 shows the dialog boxes for entering node and edge search conditions respectively.

To complete the query creation process, the web query is saved. The information saved are the search conditions for both the nodes and edges and the topological information. In addition, the graphical information of the nodes and edges are also saved in order to display the same graphical representation of the web query later.

3.3 Query Execution

We are now ready to execute the web query. The interface provides a **Execute** button to perform this operation. A dialog box as shown in Figure 11 appears once the query starts execution. The dialog box shows the status of the execution at the server end. Once the query execution completes, it will prompt the user to view the result.

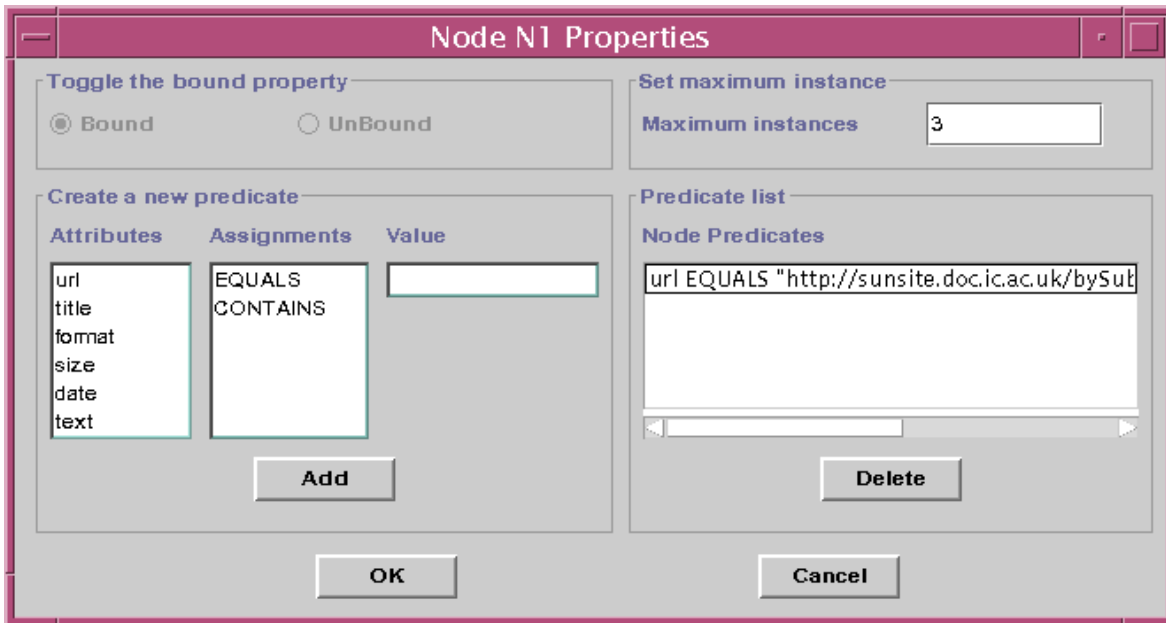


Figure 9: Dialog box for node predicate input

3.4 Query Results

Figure 12 shows the interface for displaying the results of the web query. The results are shown in three panels, the **Schema View**, **Tree View** and the **Tuple** panel. The **Schema View** panel displays the web query that has been defined earlier. The **Tuple** panel shows a list of properly formed web tuples retrieved from the WWW that satisfy the web query; in this case, there are ten of them.

Since we have defined a starting checkpoint for the web query, the graph is transformed into a tree. However, topological information are still retained. In order to give a better picture of the query structure and results, the interface also provides a tree view option through the **Tree View** panel. By doing this, more information of the results can be displayed. As shown in the figure, the **Tree View** panel shows the tree structure of the first tuple in the tuple list. Note that the URLs of retrieved documents are shown.

4 Performance Issues

Evaluating the performance of the WWW searching through checkpoints is difficult for many reasons. First, the speed of information retrieval depends heavily on network traffic. If the network is heavily utilized, the time delay in retrieving the information is longer. This delay will be added to the total elapsed time of the overall retrieval process, resulting in poorer performance.

The complexity of the web query affects the performance too. A simple query takes a shorter time to realize. Web queries with more checkpoints (and more search conditions) are more complex and takes a longer time to materialize. This applies to the complexity of the query's topological

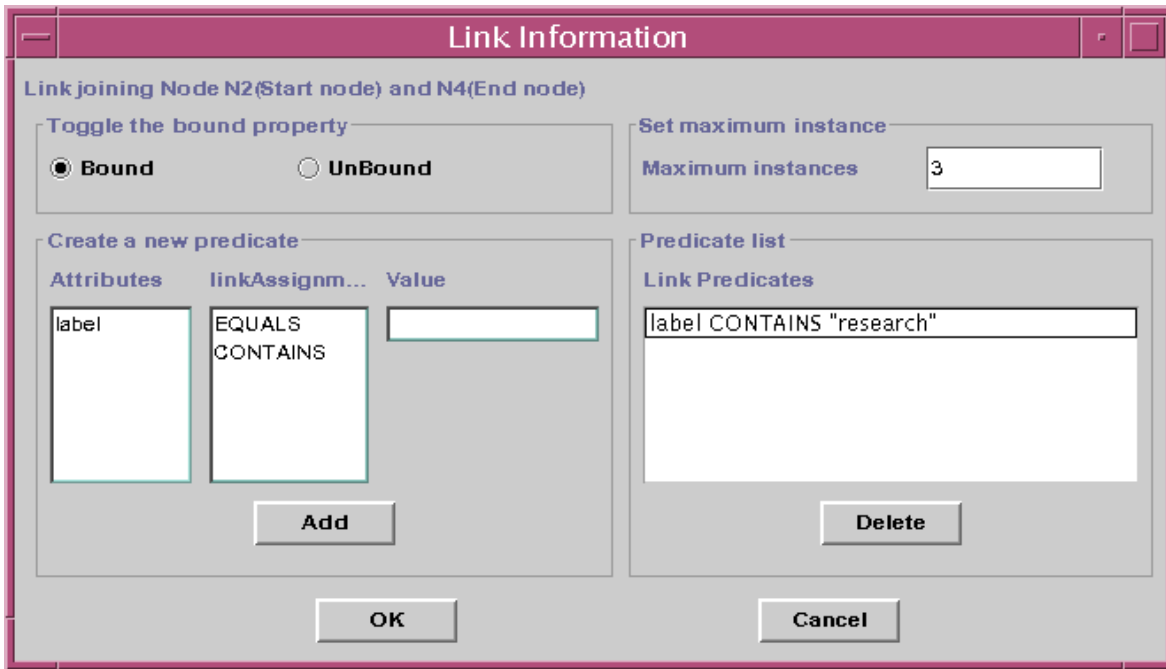


Figure 10: Dialog box for link predicate input

structure. Web queries that are more refined will be materialized faster.

Since we employ search engines to provide us with a starting point for queries that do not have a specified URL for its start node, we must consider the performance of search engines too. The reliability and effectiveness of search engines contribute to the performance of the retrieval process.

5 Related Work

As a form of web querying technique, the checkpointing concept is related to existing work in web querying. In this section, we give a brief overview of some of these works. Mendelzon, Mihaila and Milo [5] proposed a WebSQL query language based on a formal calculus for querying the WWW. Their objectives and motivations are similar to ours—to permit more complex and expressive queries on the WWW. The major difference between their work and ours is that the result of WebSQL query (a set of web tuples) is flattened immediately into linear tuples. This causes the structure information of the web tuples to be lost permanently, and thus the resultant table cannot be used further in the WebSQL query. This limits the expressiveness of queries to some extent as complex queries involving operators such as local web coupling are not possible. In our work, both structure and content are intact as web tuples in a webtable. Furthermore, they can be manipulated by web operators to satisfy new queries.

Konopnicki and Shmueli proposed a high level querying system called W3QS [6] for the WWW whereby users may specify content and structure queries on the WWW and maintain the results

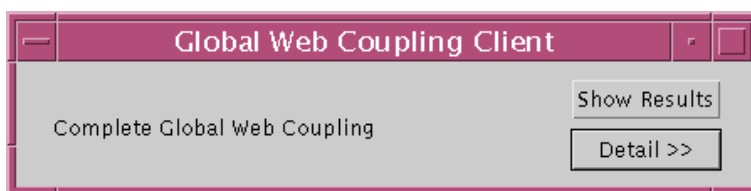


Figure 11: Executing the web query.

of queries as database views of the WWW. In W3QL, queries are always made to the WWW. Past query result are not used for the evaluation of future queries. This limits the usage of web operators like local web coupling to derive additional information from the past queries.

Fiebig, Weiss and Moerkotte extended relational algebra to the World Wide Web by augmenting the algebra with new domains(data types), and functions that apply to the domains. The extended model is known as RAW [7] (Relational Algebra for the Web). Only two low level operators on relations, *scan* and *index-scan*, have been proposed to expand an URL address attribute in a relation and to rank results returned by web search engine(s) respectively. RAW made minor improvements on the existing relational model to accommodate and manipulate web data and there is no notion of a coupling operation similar to the one in WICM.

Inspired by concepts in declarative logic, Lakshmanan, Sadri and Subrmanian designed WebLog to be a language for querying and restructuring the web information. But there is no formal definition of web operations such as web coupling.

Other proposals, namely Lorel [8] and UnQL [9], aim at querying heterogeneous and semistructured information. These languages adopt a lightweight data model to represent data, based on labeled graphs, and concentrate on the development of powerful query languages for these structures. Moreover, in both proposals there is no notion of web coupling operation similar to the one in WICM.

The WebOQL system supports a general class of data restructuring operations in the context of the Web. It synthesized ideas from query languages for the Web, semistructured data and web site restructuring. The data model proposed in WebOQL is based on ordered trees where a web is a graph of trees. This model enables us to navigate, query and restructure graphs of trees. In this system, the *concatenate* operator allows us to juxtapose two trees which can be viewed as the manipulation of trees. But there is no notion of web operation similar to ours.

6 Conclusions

Existing search engines provide a rudimentary mechanism to search for web information. In particular, it provides only one checkpoint for searching. In this paper, we extended and proposed a multi-checkpoint approach for a more advanced search process. By establishing multiple web checkpoints, a richer and controllable search procedure can be constructed to obtain more relevant Web information.

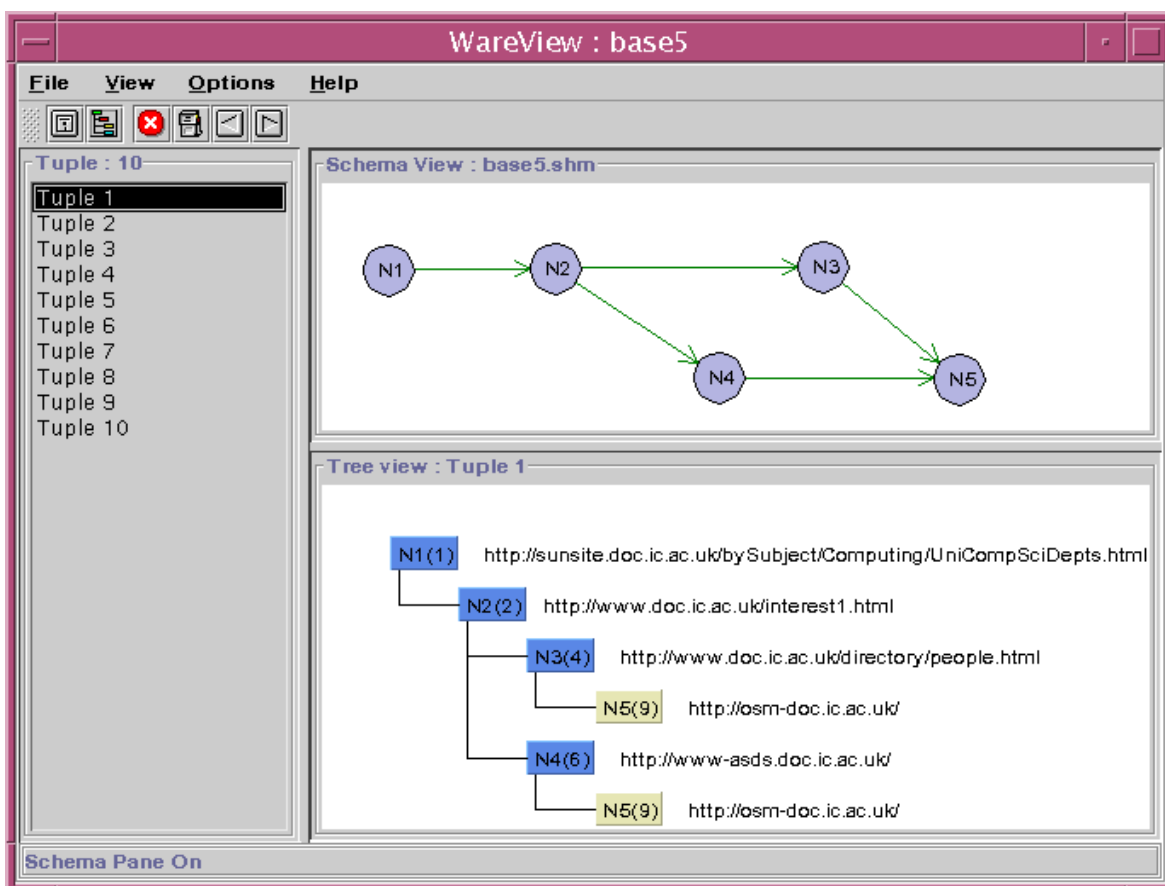


Figure 12: Web query results.

While search conditions may be defined on checkpoints to constrain and focus the searching process, unbound checkpoints (those with no conditions) are useful when a user is unsure of the exact hyperlink structure of a collection set of documents he wants to retrieve. Unbound checkpoints offers a level of imprecision in WWW searching that is useful because nobody has a complete picture of the WWW's hyperlink connectivity.

Locating WWW documents through web checkpointing is performed as part of the WHOWEDA project. We have implemented a preliminary version of this technique. and is currently performing a rigorous investigation of the performance of checkpointing in WWW searching. In particular, we are interested in the optimization of the search process under various configurations of checkpoints. These results will be reported in future papers.

References

- [1] S. BHOWMICK, S. K. MADRIA, W.-K. NG, E.-P. LIM. Web Bags: Are They Useful in A Web Warehouse?, *Technical Report*, CAIS-TR-98-13, Center for Advanced Information Systems,

Nanyang Technological University, Singapore, 1998.

- [2] S. BHOWMICK, S. K. MADRIA, W.-K. NG, E.-P. LIM. Pi Web Join in A Web Warehouse, *Proceedings of 6th International Conference on Database Applications for Advanced Applications (DASFAA'99)*, Taiwan, April 19-21, 1999.
- [3] S. BHOWMICK, W.-K. NG, E.-P. LIM. Join Processing in Web Databases. *Proceedings of 9th International Conference on Database and Expert Systems Application(DEXA98)*, Vienna, Austria, August 24–28, 1998.
- [4] S. BHOWMICK, W.-K. NG, E.-P. LIM. Information Coupling in Web Databases. *Proceedings of International Conference on Conceptual Modelling (ER98)*, Singapore, November 1998.
- [5] A. O. MENDELZON, G. A. MIHAILA, T. MILO. Querying the World Wide Web. *Proceedings of the International Conference on Parallel and Distributed Information System (PDIS'96)*, Miami, Florida, 1996.
- [6] D.KONOPNICKI, O. SHMUELI. W3QS: A Query System for the World Wide Web. *Proceedings of the 21st International Conference on Very Large Data Bases*, Zurich, Switzerland, 1995.
- [7] T. FIEBIG, J. WEISS, G. MOERKOTTE. RAW: A Relational Algebra for the Web. *Workshop on Management of Semistructured Data (PODS/DIGMOD'97)*, Tucson, Arizona, May 16, 1997.
- [8] S. ABITEBOUL, D.QUASS, J. MCHUGH, J. WIDOM, J. WEINER. The Lorel Query Language for Semistructured Data. *Journal of Digital Libraries*, 1(1):68-88, April 1997.
- [9] P. BUNEMAN, S.DAVIDSON, G. HILLEBRAND, D.SUCIU. A query language and optimization techniques for unstructured data. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Canada, June 1996.
- [10] G. A. MIHAILA WebSQL–A SQL-like Query Language for the World Wide Web. Master's Thesis, Department of Computer Science, University of Toronto, 1996.
- [11] W.-K. NG, E.-P. LIM, C.-T. HUANG, S. BHOWMICK, F.-Q. QIN. Web Warehousing: An Algebra for Web Information. In *Proceedings of IEEE International Conference on Advances in Digital Libraries (ADL'98)*, Santa Barbara, California, April 22-24, 1998.
- [12] Whoweda Web Warehousing Project, <http://www.cais.ntu.edu.sg:8000/~whoweda>, 1997, 1998, 1999.