LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-JRNL-201611

Automatic Discovery and Inferencing of Complex Bioinformatics Web Interfaces

Anne H.H. Ngu, Daniel Rocco, Terence Critchlow, David Buttler

December 24, 2003

World Wide Web

Automatic Discovery and Inferencing of Complex Bioinformatics Web Interfaces *

Anne H.H. Ngu **[1], Daniel Rocco[2], Terence Critchlow[3], and David Buttler[3]

[1] Department of Computer Science, Texas State University, San Marcos, TX 78666

[2] College of Computing, Georgia Institute of Technology, Atlanta, GA, 30332

[3] Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94551

Abstract. The World Wide Web provides a vast resource to genomics researchers in the form of web-based access to distributed data sources—e.g. BLAST sequence homology search interfaces. However, the process for seeking the desired scientific information is still very tedious and frustrating. While there are several known servers on genomic data (e.g., GeneBank, EMBL, NCBI), that are shared and accessed frequently, new data sources are created each day in laboratories all over the world. The sharing of these newly discovered genomics results are hindered by the lack of a common interface or data exchange mechanism. Moreover, the number of autonomous genomics sources and their rate of change out-pace the speed at which they can be manually identified, meaning that the available data is not being utilized to its full potential. An automated system that can find, classify, describe and wrap new sources without tedious and low-level coding of source specific wrappers is needed to assist scientists to access to hundreds of dynamically changing bioinformatics web data sources through a single interface. A correct classification of any kind of Web data source must address both the capability of the source and the conversation/interaction semantics which is inherent in the design of the Web data source. In this paper, we propose an automatic approach to classify Web data sources that takes into account both the capability and the conversational semantics of the source. The ability to discover the interaction pattern of a Web source leads to increased accuracy in the classification process. At the same time, it facilitates the extraction of process semantics, which is necessary for the automatic generation of wrappers that can interact correctly with the sources.

UCRL number: UCRL-JRNL-201611

Contact: hn12@txstate.edu; rockdj@cc.gatech.edu; critchlow1@llnl.gov; buttler1@llnl.gov

* This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48. UCRL-JC.

** This work was performed while the author was a summer faculty at LLNL

1 Introduction

The World Wide Web provides a mechanism for unprecedented information sharing among researchers. Today, scientists can easily post their research findings on the Web or compare their discoveries with previous work, often spurring innovation and further discovery. The value of accessing data from other institutions and the relative ease of disseminating this data has increased the opportunity for multi-institution collaboration, which produces dramatically larger data sets than were previously available and require advanced data management techniques for full utilization.

Some tools become *de facto* standards in the communities as they are shared among a large number of institutions. An example is the BLAST [1] family of applications, which allows biologists to find homologues of an input sequence in DNA and protein sequence libraries. BLAST is an application that has been enhanced through a *Web interface* to provide dynamic access to large data sets. Many genomics laboratories provide a Web-based BLAST interface [21, 12] to their sequence databases that allow scientists to easily identify homologues of an input sequence of interest. This capability enhances genomics research by allowing scientists to compare new sequences to known sequences, to reduce duplication effort, and to have their work validated by other members of the community. The addition of new sequences at a frequent rate [20, 19] further increases the value of this capability.

Unfortunately, while the underlying program on many of these sites is the same, there is no common interface or data exchange mechanism for the established BLAST sources currently on the Web. To perform a BLAST search against multiple sources, a scientist must manually select the set of sites to query, enter their query into each site, and integrate the results. There are numerous problems with this approach, including: the scientist may not query the most relevant sites for their search, the search must be entered multiple times, the results of the search must be merged together by hand to obtain an integrated set of results, and if an interface changes or moves, the scientist must ascertain where the new interface is and how to query it appropriately.

Previous work in integration of heterogeneous data sources [3, 6, 16] focused on providing integrated access to a myriad of web sources based on the use of common agreed vocabularies or ontologies and manual-coding or semi-automatic generation of source specific translators or gateways. However, the number of web sources and their rate of change out-pace the speed at which translators for the various heterogeneous sources can be built and made available for integrated access. Moreover, manually maintaining a wrapper library will not scale to accommodate the growth and the rapid change of scientific data sources on the Web. Rapid technological advances in ge-

nomics also make it impossible to define a common set of concepts which is fundamental to the heterogeneous data source integration.

Providing integrated access to a large number of BLAST Web sources is a difficult and important problem in genomics. The major challenges are to locate new Web sources, evaluate them to determine if they provide a BLAST interface, interact with them to determine their process semantics, construct a wrapper for the source, and integrate the wrapper into a multidatabase system that can provide a single point of access to all known sources conforming to the interface. Source autonomy complicates this problem: a cursory Web search yields hundreds of sources that provide a BLAST interface, many of which do not appear in bioinformatics directories [8]. Our goal is to produce an automated system that can find, classify, interact with, and wrap new and changed web sources without costly human intervention.

During the evaluation of a web source, it is important to be able to identify both the capability of the source as well as the correct interaction pattern with the source. For example, a web interface might accept input parameters that span a sequence of HTML pages and return the result in stages depending on the nature of the input parameters and the user profile. We use the term *indirection page* to refer to all the intermediate pages that contribute to the correct interaction pattern of a Web source after the first query submission and before end-user receives the answer page. In our initial experience in the classification of BLAST Web sources, at least 10% of these sources failed to be classified correctly because of indirection pages. The ability to detect indirection pages early in the classification process avoids an unnecessary and expensive re-classification process. Moreover, indirection pages are an essential part of general Web interface design. For example, successfully completing a sale at a Web source might consist of a sequence of pages: a login page, an item selection page, a checkout page, a payment page, and a purchase confirmation page. A novel feature of our automatic classifier is the ability to infer not only the capability of the source, but also the associated indirection pages (interaction patterns) if there are any. The contributions of this paper are:

- A practical, heuristic approach to classifying arbitrary Web sources using a service class description driven by the need of the users. This alleviates the need to agree on a common ontology and offers the flexibility to incorporate the need of individual information seeker.
- A robust and efficient approach to infer the interaction pattern or hidden states of a given Web source that improves the success rate of the classification process.

This classification system is part of an ongoing research effort to build an automated integration system for Web sources. To concretely demonstrate the approach, our discussion in this paper

focuses on BLAST interfaces, but these flexible techniques are generic and can be easily applied to other domains. In Section 2, we discuss research related to our work. We present our service description format in Section 3.1 and the automatic classification system in Section 3.2. Section 4 describes how we derive a robust and efficient approach to identify a variety of indirection pages for BLAST sites. Section 5 outlines our experimental evaluation using results obtained from applying our techniques to a set of BLAST sequence search services on the Web. As part of this discussion, we identify characteristics of sources that our prototype cannot currently handle, and highlight the improvement in the classification process by incorporating detection of indirection pages. We conclude with an examination of this work and future research opportunities.

2 Related Work

Our work is inspired by the ShopBot agent [9] whose purpose is to assist users in the task of online shopping. ShopBot uses a domain description that lists useful attributes about the services in question. The authors addressed the problems of discovering unknown vendor sites and integrating a set of learned sources into a single interface. Our present work addresses the related problem of automatically classifying services from an arbitrary set of sites. The service class description format we describe provides greater descriptive power than ShopBot's domain descriptions and can specify complex data types and source capability information. Furthermore, our approach goes beyond Shopbot's domain description and also addresses the interaction semantic embedded in the vendor sites.

In [18], machine generated ontologies are used to allow a more focused approach to seeking information on the Web for a specific domain. A methodology based on a set of heuristics is used to infer the type of a large collection of web interfaces of a particular domain. This approach is similar in spirit to our approach of using a service class description to discover capability of a web site. However, in their approach, the discovery of interaction pattern, which they called precedence relationships cannot be done automatically. It requires human input to discover the hidden interaction patterns of a web site. The work in [2] addressed the problem of automatically extracting structured data (schema or type) encoded in a collection of web pages without any human input or training data sets. This is different from our approach where given a structured data (service class description), we want to automatically identify sites that conform to it.

Related to this work is the problem of heterogeneous data source integration. There are several research and commercial systems for querying heterogeneous data sources. Zadorozhny et al. [24] describe a wrapper and mediator system for limited-capability Web sources that includes query planning and rewriting capabilities. Information Manifold [16] targets the myriad of Web interfaces

to general purpose data, using a declarative source description for these sources combined with a set of query planning and optimization algorithms. The TSIMMIS [6] system provides mechanisms for describing and integrating diverse data sources while focusing on assisting humans with information processing and integration tasks. InfoSleuth [4] is an agent-based system for information integration, discovery and retrieval in a dynamic and open WWW environment. In InfoSleuth, agents are used to wrap Web data sources. While it is easy to wrap a data source using the agent metaphor, it is non-trivial to define a common ontology for integrating different Web sources. This requires semantic differences between different sources in a particular domain to be resolved first. In the bioinformatics domain, such an ontology is not available due to the rapid advancement in technology in this area. In the SIMS project [3], again, a common ontology and a common query language is used to facilitate the combination of information from heterogeneous data sources.

Researchers have also examined heterogeneous data integration in the domain of biological data. DiscoveryLink [14] provides access to wrapped data sources and includes query planning and optimization capabilities. Eckman et al. [10] present a similar system with a comparison to many existing related efforts. BioKleisli [7] provides access to complex sources with structured data but does not include query optimization.

Our goal is to construct a system that can automatically discover, describe, and integrate bioinformatics Web sources. We seek to unify a class of sources such as BLAST behind a single interface that will maintain a current set of sources without manual intervention. This paper focuses on the *automatic* classification of source capability and interaction pattern, which the above systems do not address. Many of these mediation systems could utilize the results of our classification system to identify sources to wrap.

Automatic discovery of Web sources apropos to a particular domain involves both locating sources and determining their relevance to the domain; this paper addresses only the second step. Locating sources in the context of the Web typically involves a crawler that treats sites as nodes in a graph connected by hyperlink edges. Starting from a set of root pages, a crawler traverses the graph in some order specific to its goals and processes the sites it encounters. Part of this process involves extracting new hyperlinks to crawl from the encountered sites. While simple on the surface, Web crawling presents several research and implementation challenges, many of which have been addressed in the literature and commercially [5, 17, 15]. There is active research into topic-driven or focused crawlers; Srinivasan et al. [23] present such a crawler for biomedical sources that includes a treatment of related systems.

3 Discovering a Web Source's Capability

In order to provide a unified single-point of access to a large number of distributed and autonomous Web sources, it is necessary to discover the capabilities of the sources automatically. Our approach to discovery and classification of the capabilities of Web sources is based on the concept of *service classes* that share common functionality but not necessarily a common interface. By describing the salient characteristics of a class of services and an essential set of examples, in a generic format, we are able to use this description to evaluate specific sources. The details of the discovery process can be found in [22].

3.1 Service Class Descriptions

Service classes are specified by a *service class description*, which uses an XML format to define the relevant functionality of a category of Web sources, from an application's perspective. The service class description format supports four categories of information used to define an interface of interest: data types, input parameters, control flow, and examples.

Data Types are used to describe the input and output parameters of a service class and any data elements that may be required during the course of interacting with a source. The service class type system is modeled after the XML Schema [11] type system and includes constructs for building atomic and complex types. *Atomic types* are simple valued data elements such as strings and integers. The XML Schema type system provides several built-in atomic types that can be used to create user-defined types defined by restriction. The `DNASequence` type in Figure 1 is an example of an atomic type defined by restriction in the nucleotide BLAST service class description. Figure 1 also shows the specification of a nucleotide BLAST alignment sequence fragment, which is a string similar to:

```
Query: 280 TGGCAGGCGTCCT 292
```

The above string in a BLAST result would be recognized as an `AlignmentSequenceFragment` type.

Atomic types can be composed into complex types which allow us to define types such as `SummaryResults` (refer to Figure 1) that make use of a complex type called `Alignments` or `EmptyDNABLAST`. Figure 2 shows an instance of the `Alignments` type for a BLAST source.

Control flow graphs are used for enumerating the expected navigational paths used by all members of the service class. A control flow graph consists of a set of states connected by edges. In order to be as general as possible, the control flow graph does not include either delay pages (e.g. your

```
<type name="DNASequence"
      type="string"
      pattern="[GCATgcat-]+" />

<type name="AlignmentSequenceFragment" >
    <element name="AlignmentName"
             type="string"
             pattern="[:alpha:]+:" />
    <element type="whitespace" />
    <element name="m"
             type="integer" />
    <element type="whitespace" />
    <element name="Sequence"
             type="DNASequence" />
    <element type="whitespace" />
    <element name="n"
             type="integer" />
</type>
<type name="Alignments"
      <element name="QueryString" type="AlignmentSequenceFragment"
      <element type="AlignmentString" required="true"/>
      <element name="SequenceString" type="AlignmentSequenceFragment" required="true"/>
</type>
<type name="SummaryResults">
      <choice>
          <element type="Alignments" />
          <element type ="EmptyDNABLAST"/>
      </choice>
</type>
```

Fig. 1. *Sample nucleotide BLAST type definitions.*

```
Query:   179 GGCTTCTACACCAAAGTGCTCAACTACGTGGACTGGAT 142
                ||  | ||||||||| ||| || | ||||||||||||||
Sbjct:     3 GGTGTTTACACCAACGTGGTCGAGTACGTGGACTGGAT 40
```

Fig. 2. *Example of a nucleotide alignment.*

browser window will automatically refresh in 15 seconds) or intermediate pages that point to the ultimate results (e.g. click here to view results). We consider both delay pages and intermediate pages as indirection pages. Figure 3 shows the XML specification of the control flow graph used by our nucleotide BLAST service classifier. Our example has a single start state that defines the type of start page a class member must contain: in this case, any member of the nucleotide BLAST service class must have a start page that includes an HTML form. This simple description has the advantage that it can be used to match with the maximum number of BLAST sources, with actions required to traverse intermediate and delay pages specific to a given instance of the service class member inferred during the interface classification.

```
<controlgraph name="BLASTN">
     <vertices>
       <vertex name="start" type="HTMLform"/>
       <vertex name="end" type ="SummaryResults" />
    </vertices>
    <edges>
      <edge origin="start" destination="end" />
    </edges>
</controlgraph>
```

Fig. 3. *Control Flow Graph definition.*

Examples contain queries that can be executed against an instance of the service class. In our current prototype, input parameters of a service class is specified as arguments of examples. Specifically, examples can be used to determine if a site accepts input (data) as required by the service class.

Figure 4 shows an example used in a nucleotide BLAST description that illustrates the components of an example argument. The attribute `required` states whether the argument is a required input for all members of the service class; all members of the nucleotide BLAST service class are required to accept a DNA sequence as input. The argument type is listed as well as a specific value that can be used during interaction with the site. This example also includes an optional argument called `BLASTProgram` for conducting the blast. This is specified as an optional argument because some BLAST sources do not have a program selector input.

The optional `hints` section supplies clues to the site classifier that help select the most appropriate form parameters on a Web source to match an argument. For example, a DNA sequence is always entered into a text input parameter, usually with “sequence”, “query_data”, or “query”

```
<example>
  <arguments>
    <argument required="true">
      <name>sequence</name>
      <type>DNASequence</type>
      <hints>
        <hint>sequence</hint>
        <hint>query</hint>
        <hint>query_data</hint>
        <inputType>text</inputType>
      </hints>
      <value>TTGCCTCACATTGTCACTGCAAAT
             CGACACCTATTAATGGGTCTCACC
      </value>
    </argument>

    <argument required="false">
      <name>BlastProgram</name>
      <type>string</type>
      <hints>
        <hint>program</hint>
      </hints>
      <value>blastn</value>
    </argument>
  </arguments>

  <result type="SummaryPage" />
</example>
```

Fig. 4. *A nucleotide BLAST example.*

in its name. The DNA Sequence argument in a nucleotide BLAST service class therefore includes hints of "sequence", "query_data", "query", with type "text."

3.2 Source Capability Classification Process

The automatic classification of Web sources consists of two steps: locating interfaces and determining if they are instances of the service class. Locating interfaces can be achieved by a spider agent that is capable of following links and interacting with forms. This issue is not the focus of this paper. We summarize the approach that we take for identifying members of the service class in the following paragraphs. Refer to [22] for further details.

The service classifier begins the analysis of a Web source by attempting to match the start page of the source against one of the start nodes in the control flow graph. If no matches are found, the source cannot match the service class and is discarded. If the start page matches, the classifier generates a series of queries using the examples provided in the service class description. For each response, the classifier then follows the outbound links and tests the responses of the source against the possible states in the control flow graph. This process continues until either the site matches one of the end states in the control flow graph or there are no more possible queries to try. The first prototype implementation of this source classifier was unable to infer source-specific interaction patterns that were more complex than the a start state and end state (i.e. a single request-response interaction); our new prototype has enhanced processing capabilities that allow it to discover other interaction patterns is described in section 4.

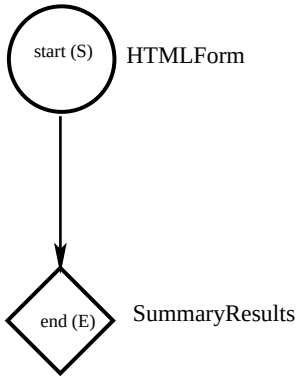

Fig. 5. *Control Flow Graph for a nucleotide BLAST service*

The control flow graph for a nucleotide BLAST service is shown in Figure 5. In the control flow graph, start and end states are represented with a circle and diamond respectively. For the

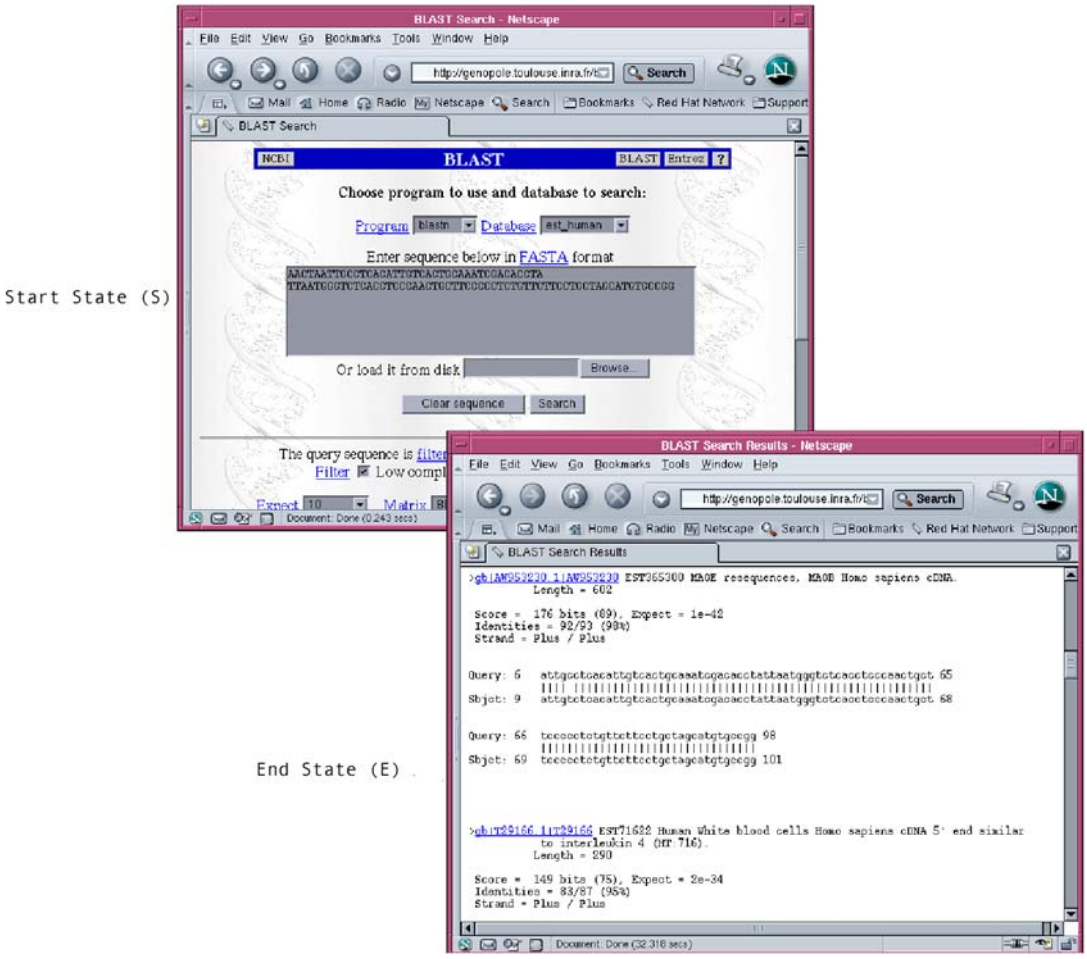

Fig. 6. *Potential Web source that matches nucleotide BLAST service control flow graph*

classification of BLAST sources, only the start and end states are required to be specified in the service class description. However, our control graph specification can be extended to specify complex control flows such as those that require unique intermediate states when the need arises. The type associated with each state in the control graph is listed next to it: `HTMLform` is the type for the start state of a nucleotide BLAST sequence search, `SummaryResults` is the type for the end state of a BLAST sequence search. `SummaryResults` is a complex type that can match with either `Alignments` or `EmptyDNABLAST` type as shown in Figure 1.

If the Web source's start page (S) does not match the type of this control graph's start state, the classifier returns a negative result. If the start page matches, the classifier uses the examples to query the site, which returns its result (E). If this result matches the `SummaryResults` type for a nucleotide BLAST (i.e. it is either `Alignments` or `EmptyDNABLAST`), the classifier returns a positive result with the details needed to execute a query against the site. Figure 6 is an example

of a BLAST interface that matches the control graph definition shown in Figure 5. Note that a A significant part of the classifier's task is to infer the control graph for the source using the example queries. As we discuss in Section 4, this task can be significantly complicated by the presence of indirection pages which have to be discovered during the classification process.

Generating queries with which to test a candidate interface is a significant challenge when analyzing a Web source but is vital to verifying whether the interface is an instance of the service class. Our query generator takes the examples from the service class description and produces a set of test queries that matches each argument in an example with a parameter in the interface's forms. Each test query is assigned a priority using a simple function that assigns points to a query for each parameter that matches the hints of its example argument. The queries are then executed in priority order until either one leads to an end state or there are no more queries to execute.

4 Discovering a Web Source's Interaction Pattern

In our initial effort to classify DNA BLAST sites using our prototype implementation, we found that many sites, including the popular NCBI BLAST site, failed to be classified correctly because their control flow is more than just simple `start` and `end` states. These sites utilized indirection pages that needed to be traversed to reach the query results. In other words, it takes multiple interactions to arrive at the expected end state. During the evaluation of a Web source, it is important to discover these interaction patterns because they are an integral part of the semantics of the source. In the following sections, we first define the characteristics of an indirection page. We then propose a simple heuristic-based approach to identify indirection pages. The shortcomings of this approach are then discussed. This leads to the more robust and efficient approach called PageDiff for identification of indirection pages.

4.1 Indirection Page

Technically, an indirection page is an HTML response page that contains a *pointer* which will eventually lead to the expected results. The *pointer* could be a form that needs to be submitted (Figure 7) or an HTML reference link (Figure 8) that needs to be clicked on. After conducting an in-depth analysis of ten different genomics indirection pages shown in Table 1, we arrived at the following observations:

- there is no single, consistent page layout or template that can be used to identify indirection pages from different genomics sites or even within a single class of genomics interfaces,

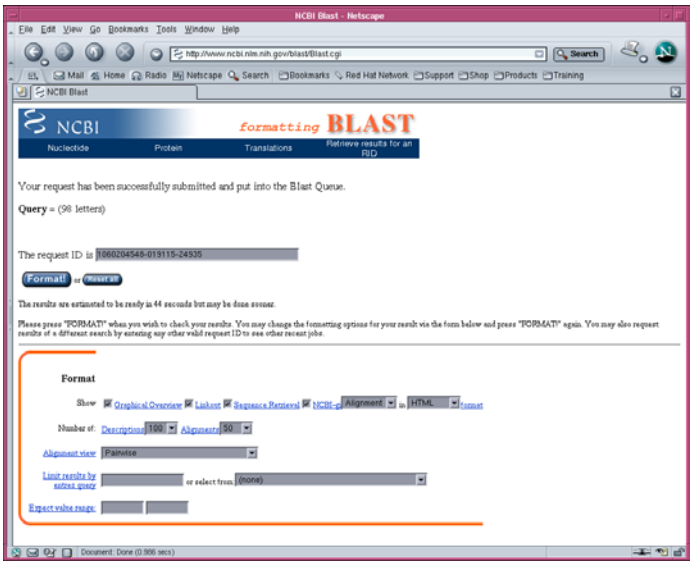

Fig. 7. *An Indirection Page from NCBI BLAST site (clicking on the format button will lead to the result summary page)*

- an indirection page can be as simple as an HTML page that contains a single HTML form (site 7) or a single HTML link (site 2),
- an indirection page can be a complex HTML page with multiple forms and links whose options are controlled by JavaScript (site 10),
- a typical indirection page contains both dynamic and static information. Dynamic information is populated and generated during query processing while static information is part of the original html template used to generate the indirection page,
- an indirection page can be nested. For example, site 4 first points to a refresh page, which leads to an HTML page that contains pointer to the results page,
- an indirection page could be automatically refreshed by the Web browser, or could require human intervention to be refreshed (site 3).

We also analyzed the Protein Data Bank (PDB) site in addition to the ten sites listed in Table 1. This is used to confirm the characteristics of indirection pages within genomics sources as well as ensuring that our solution for indirection page identification is applicable to non-BLAST sites.

4.2 Strategies for Identification

A naive approach to identify an indirection page or a sequence of indirection pages is to traverse all the outbound links of a given HTML page, download the content pointed by each link and check the

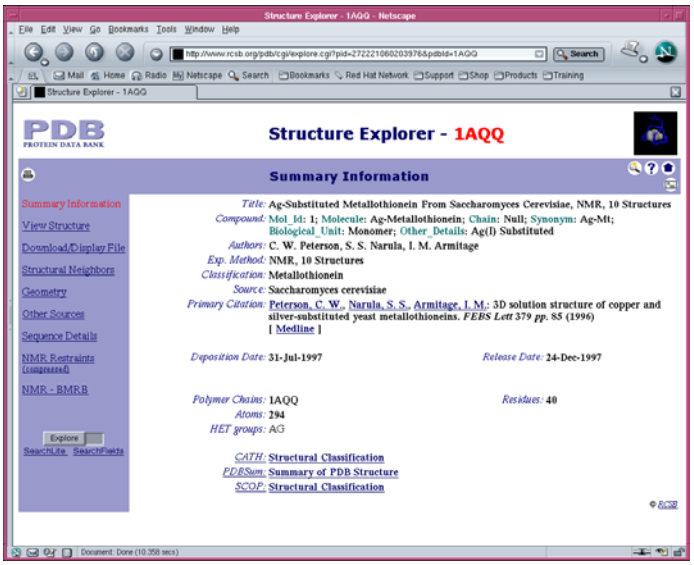

Fig. 8. *An Indirection Page from PDB site (clicking on the download/Display File link will lead to the result summary page)*

type of the content page. The traversal process terminates when a content page that corresponds to the type of the expected result summary as defined in the service class description is found. Such an approach is infeasible because:

- it is computationally very expensive. The example indirection page shown in Figure 8 has 28 outbound links to traverse, and a total of 421 pages to download and check if we set the search level to two,
- this approach treats all outbound links as having the same chance of leading to the result summary page. Clearly there are certain outbound links, such as an HTML reference link to a help manual or home page, that simply maintain the consistency of the Web interface. Such a link will not lead to any interesting results pages,
- following every possible outbound links in a particular HTML page has the effect of flooding a particular Web source with too many requests and this may lead to denial of service, and,
- following every link blindly might end up looping through the same set of pages forever.

Our goal is to prioritize the outbound links in an HTML page in order to identify an indirection page efficiently and robustly without resorting to expensive text understanding and information extraction processes. We use the term *dynamic links* to refer to the set of outbound links that are generated during the interaction with the Web interface.

Site No.	Site URL	Type of Indirect page
1	http://www.genedb.org/genedb/dicty/blast.jsp	Retrieve button and links
2	http://www.sgn.cornell.edu/cgi-bin/SGN/blast/blast_search.pl	Click to view result link
3	http://pbil.iniv-lyon1.fr/BLAST/blast_nuc.html	Click here to see your results → Click Reload button to check status
4	http://zeon.well.ox.ac.uk/git-bin/blast2	forms &links → Refresh → Click here → output page
5	http://www.rtc.riken.go.jp/jouhou/HOMOLOGY/blast	Check your entry → new pop-up window for email result
6	http://www.ncbi.nlm.nih.gov/blast/Blast	Format button & links
7	http://www.bioinfo.org.cn/lmh/blastlmh.html	Press it button
8	http://www.sanger.ac.uk/HGP/blast_server.shtml	Retrieve result button & links
9	http://genoplante-info.infobiogen.fr/pise/blast2_gpi.html	Multiple forms & links → email
10	http://www.ebi.ac.uk/blast2	Refresh → forms, links & result summary page

Table 1. BLAST Indirection Pages

4.2.1 Simple Heuristic Approach to Discovery of Dynamic Links

The observation that a typical indirection page contains both static and dynamically generated information leads to the development of the following heuristics for identification of *dynamic links* in an HTML page. The goal is to eliminate those links in an HTML page that will not contribute to a correct interaction. These heuristics are listed below:

1. an HTML form is a dynamic link that should always be followed first. This assigns HTML form to be of a higher priority dynamic link than a HTML reference link,
2. an HTML reference link that ends with a file extension such as ".html", ".htm", ".css", ".ps", ".jpeg", ".zip", ".pdf", ".gif" etc is not a dynamic link,
3. an HTML reference link that does not begin with a HTTP protocol is not a dynamic link,
4. an HTML reference link that has a different domain from the URL used to make the initial request is not a dynamic link.

Heuristic one is based on the observation that traversals of indirection pages on the ten identified BLAST sites are frequently achieved by pressing submit buttons on HTML pages. Heuristics two uses our knowledge that dynamic links are generated on the fly and any HTML reference links that ends with one of the above listed extensions cannot generate dynamic links and can be eliminated. Heuristic 3 eliminates links that point to content which cannot be downloaded using HTTP protocol

and hence cannot be checked. Heuristics 4 assumes that the result page cannot come from a different URL domain.

Using those heuristics, we are able to successfully identify six of the initial ten manually identified indirection BLAST pages. Two of the indirect BLAST sources (5, 9) require summary results to be emailed back and thus do not conform to the type of interfaces that we are looking for. One of the sites (10) failed because it has complex JavaScript. Site number 3 failed because the indirection page requires specialized human intervention. Using this simple heuristic for the identification of indirection pages, we are able to reduce the potential set of HTML pages to check from 421 to 83 for the PDB example shown in Figure 8. Despite its performance advantage as compared to the naive approach, this heuristic approach has three serious shortcomings:

- It could miss a genuine dynamic link that ends with an ".html" extension. For example, the result summary page could be pointed to by http://www.xxx.yyy.gov/blast/tmp/A123456/index.html. The string "A123456" is a session-id that is generated on the fly.
- It could mis-classify a static link such as http://www.xxx.yyy.gov/blast/query_form.cgi as a dynamic link because of the ".cgi" file extension.
- It assumes that an HTML form is always a dynamic link. This is not always true. Multiple forms can exist in an HTML page. Some of those forms are there purely for the convenience of user navigation. This results in submission of forms that will never lead to the result summary page.

4.2.2 PageDiff Approach for Discovery of Dynamic Links

In this section, we describe a more robust and efficient approach, called PageDiff, to identify an indirection page. The fundamental challenge is to automatically distinguish the HTML reference links and forms in an HTML page that are truly *dynamic* regardless of the syntactic nature of the links or forms. On the other hand, we want to avoid using expensive text understanding mechanisms to figure out the semantics of an HTML page. Dynamic information is usually generated during query processing based on either the underlying data sets or part of the state information that needs to be maintained for each request.

We can identify dynamic links by focusing on how dynamic information is encoded in an HTML page. In genomics data sources, state information for a request is typically passed using URL re-writing, hidden fields, or session-ids. This leads to our hypothesis that if we post two queries to the same Web interface and compute the difference in the HTML reference links and forms between the two response pages, the resulting sets will represent the links and forms that are dynamically

generated during the query processing for each query. Such an approach results in more accurate detection of dynamic links without the high overhead of having to understand the semantics of the HTML page.

The algorithm for PageDiff is shown in Figure 9. It consists of three main steps. The first

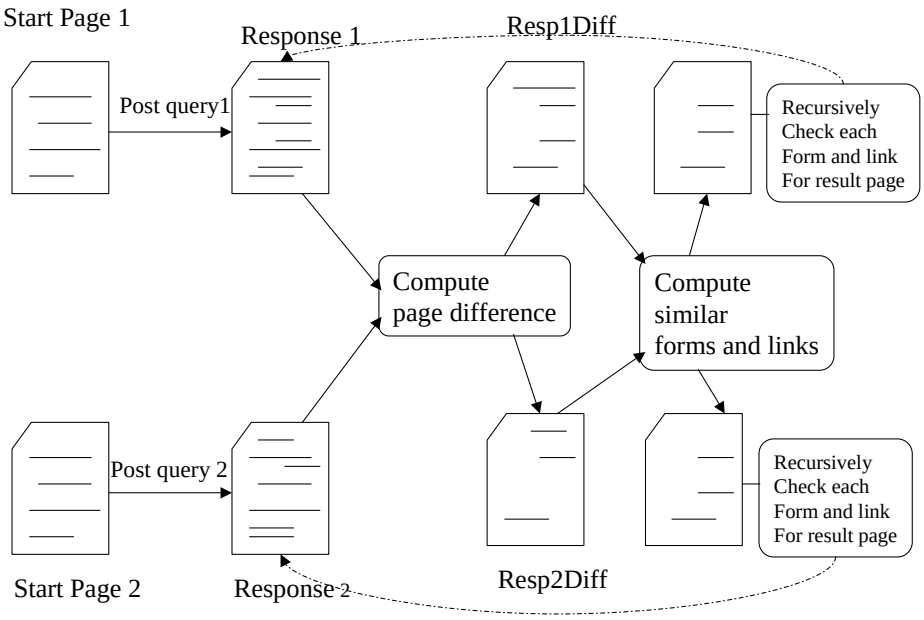

Fig. 9. *PageDiff Algorithm for computing minimal sets of dynamic links to check for indirect pages*

step chooses and posts queries. The second step computes the page differences and the third step recursively identifies sets of dynamic links to follow. During the classification process, a set of queries based on different examples described in the service class are generated and attempted on the site. An n number of the top ranked queries are cached. The PageDiff algorithm first chooses the top ranked query from the cache and posts this query twice to the same website.

Two HTML response pages are obtained as the result of posting the two queries. These response pages will serve as input for the difference computation. In computing the page difference, we only focus on the outbound links and the forms. We consider two HTML reference links different if their string values are different. Two HTML forms are considered different if their actions are different or if any of the hidden value of their hidden parameters are different. The resulting two sets (`resp1Diff`, `resp2Diff`) from the difference computation form the input to the similarity computation step. Note that the difference computation is not symmetric. The first set `resp1Diff`

represents the set of dynamic links to follow from the first response page, and the second set `resp2Diff` represents the set of dynamic links to follow from the second response page.

The goal of the similarity computation is to identify the comparable set of dynamic links that can be followed pair wise recursively for nested indirection pages. We want to avoid doing a page difference between two totally unrelated indirection pages. The mechanics of the similarity computation is the same as the *join* operator in relational algebra. The basic idea is to compute the items that match certain conditions or are related from the two given sets. The criteria used for defining *matching conditions* for HTML forms and HTML reference links determine the dynamic links to follow at each search level. Currently, those *unmatched* HTML forms and HTML reference links are not followed when the same query is posted twice. Performing a pair wise page difference computation between two uncorrelated HTML pages might result in a set of dynamic links which equal to all the outbound links in an HTML page. This degenerates to naive approach for identification of indirection pages. The criteria for computing similarity for HTML forms and HTML reference links are discussed in the following two sub-sections.

4.2.3 Criteria for matching two HTML forms

An HTML page can contain multiple forms, each having an index and a name. However, this index and name are not unique. Posting two queries (different input values) to the same Web interface might end up with two HTML pages that differ in the number of forms. Thus the first form in the first HTML page might not correspond to the first form in the second HTML page. We define two HTML forms to be *similar* if they satisfy the following conditions:

- the form action tags match,
- the number and the type of hidden parameters match (the values of the hidden parameters can be different),
- the two forms have a common submit button

Using the above conditions, two forms that have different number of parameters, type of parameters (input, text, select-one, radio, check-box), and buttons but have the same action tags, hidden parameters, and a common button are considered to be *similar*.

4.2.4 Criteria for matching HTML reference links

An HTML reference link is a URL (Unified Resource Locater), which consists of the name of the protocol, the host, the host path, the file name and possibly a list of query parameters and their values. Figure 10 shows an example of an URL and its parts.

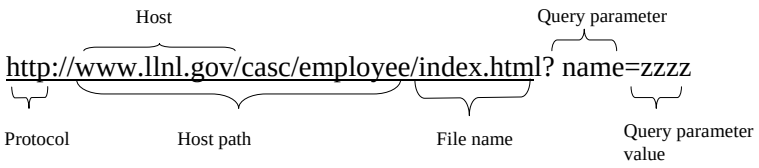

Fig. 10. *Format of an URL.*

We define two HTML reference links that have a list of query parameters to be *similar* if their host paths and file names are the same, and the list of parameters are the same. The following are two matching or similar HTML reference links:

http://www.rcsb.org/pdb/cgi/explore.cgi?pid=94041058908865&page=0&pdbId=1B20
http://www.rcsb.org/pdb/cgi/explore.cgi?pid=96011058909035&page=0&pdbId=1AOO

For HTML reference links that do not have any query parameter, we have two cases to consider: a) if their host paths match, their file extensions match and the label of their links match (e.g. both are displayed as blast.html in the browser), we consider the two links to be similar. The following two URLs is an example of such a match:

http://www.xxx.yyy.org/blast/A123456.html
http://www.xxx.yyy.org/blast/A125678.html

b) if their host paths are sub-strings of each other, their file names match, and the label of their links match, we consider the two links to be similar. The following is an example of matching HTML reference links that do not have any query parameter:

http://www.xxx.yyy.org/blast/tmp/A123456/index.html
http://www.xxx.yyy.org/blast/tmp/A234567/index.html

The above criteria are by no mean an exhaustive list for matching HTML reference links and HTML forms. We believe that this set of criteria will be fine-tuned as we experiment with more BLAST indirection pages. Currently, we find that this set of criteria is sufficient for the successful identification of a variety of BLAST indirection pages shown in Table 1.

4.3 Comparative Analysis of Simple Heuristics and PageDiff

In this section, we briefly compare the robustness and the efficiency of the simple heuristic and the PageDiff approaches in the identification of indirection pages. Robustness refers to the fact that we

do not discard dynamic links that will lead to identification of indirection pages. Efficiency refers to the time it takes to correctly identify an indirection page.

The PageDiff algorithm obviously incurs more overhead in computation because it performs pair wise comparisons of HTML pages until the result summary page is found. It also incurs higher disk I/O since two HTML pages must be downloaded each time. To evaluate the amount of overhead, we conducted experiments, using the ten BLAST sites identified in from Table 1. The results are summarized in Table 2. The experiments were conducted on an Intel Pentium-based system running the Linux operating system. The time shown in seconds is an average over five runs for each site. Site 5 fails to be classified by PageDiff because it requires the results to be emailed back exclusively. This site falls outside the type of Web interfaces that our classifier is looking for. Site 10 fails because of complex JavaScript. Site 3 fails because it requires further human intervention. PageDiff is able to correctly classify indirection pages pointed to by a link that ends with an ".html" file extension while the simple heuristic approach will mis-classify it as a static link. This demonstrates the robustness of PageDiff approach and explains why PageDiff is able to identify site 9 (results are returned through email as well as html link), while the simple heuristic failed to identify it.

Site No.	**BLAST Site**	**Indirection Page Identified**	**Time Taken in seconds**	
			PageDiff	**Heuristic**
1	http://www.genedb.org/genedb/dicty/blast.jsp	Yes	56	335
2	http://www.sgn.cornell.edu/cgi-bin/SGN/blast/blast_search.pl	Yes	46	28
3	http://pbil.iniv-lyon1.fr/BLAST/blast_nuc.html	No (manual refresh)		
4	http://zeon.well.ox.ac.uk/git-bin/blast2	Yes	60	240
5	http://www.rtc.riken.go.jp/jouhou/HOMOLOGY/blast	No (email)		
6	http://www.ncbi.nlm.nih.gov/blast/Blast	Yes	166	162
7	http://www.bioinfo.org.cn/lmh/blastlmh.html	Yes	46	29
8	http://www.sanger.ac.uk/HGP/blast_server.shtml	Yes	90	318
9	http://genoplante-info.infobiogen.fr/pise/blast2_gpi.html	Yes (by PageDiff)	360	
10	http://www.ebi.ac.uk/blast2	No (JavaScript error)		

Table 2. Result of Indirection Page Identification

For a simple indirection page that only has a single submit button or a single link to follow, the simple heuristic performs better than PageDiff. However, the heuristic approach only outperforms PageDiff by around 40%. When it comes to an indirection page that has a combination of different forms and HTML reference links, PageDiff outperforms Heuristic approach 336% as shown by site

1, 4 and 8. In the case of the PDB site, which has many outbound links to check, PageDiff is able to reduce the number of links to traverse from the initial 421 to 35 links while simple heuristics approach can only reduce it to 83 links. PageDiff avoids submitting forms that remain unchanged across the two queries while the heuristic approach will try all the forms regardless of whether they are static or dynamic.

5 Experimental Results

		Identified	**Failed Sites**			**Percent**
Data Set	**Identified**	**with Indirection**	Spec. Interaction	Processing	**Total**	**Identified**
Initial test set	18	3	1	5	27	77.7%
Experimental set	64	5	1	22	92	75.0%

Table 3. Sites classified using the nucleotide BLAST service class description.

We have constructed two prototypes of the source discovery system described here to test the validity of our approach. Both prototypes are implemented in Java and can examine a set of supplied URLs or crawl the Web looking for sources matching a description. Interaction with the Web is handled by the HttpUnit user agent library [13]. Our second prototype builds on the initial implementation to detect indirection pages.

5.1 Methodology and Data

The data for our experiments consists of a list of 27 URLs of BLAST interfaces which we used as test-bed and another 104 distinct URL's that were used for the final experiment. The use of distinct URL's is important to ensure that we have a random set of URLs from different blast sites for the experiment. Those interfaces were gathered from the results of a Web search, and vary widely in complexity: some have forms with fewer than 5 input parameters, while others allow minute control over many of the options of the BLAST algorithm. Approximately 10% of the interfaces utilize indirection pages (12 sites). This number does not include delay pages that get handled automatically by the HttpUnit user agent library. A significant minority of the sources use JavaScript to validate user input or modify parameters based on other choices in the form. Despite the wide variety of styles found in these sources, our current prototype is able to recognize a large number of the sites using a service class description of approximately 150 lines.

The small test-bed of the nucleotide BLAST URLs was used for evaluating our prototype implementation. The other 104 URLs were used for experimental testing after the prototype implementation was deemed ready. There were 12 sites that were manually determined to be non-functional or that returned results exclusively via email and were excluded from our experiments and do not appear as part of the reported results.

5.2 Results

Table 3 shows the results of our experiments. The initial test set is the set of Web sources that were tested repeatedly as the prototype matured and helped shape its design. The remaining sources were classified once. Sites listed as "identified" are those that can be correctly queried by the first prototype classifier to produce an appropriate result, either a set of alignments or an empty BLAST result. An empty result indicates that the site was queried correctly but did not contain any homologues for the input sequence. Sites listed under "identified with indirection" are those additional sites that were classified correctly after we enhanced the classifier with the ability to detect indirection pages using the PageDiff algorithm. Half of the indirection pages in the experiment failed to be identified because of JavaScript problems.

Failed sites are all false negatives that fall into two categories: specialized interaction and processing failures. A specialized (Spec.) interaction source is indirection page that requires further specific user input in order to continue to the next state. A page that asks the user to enter a request-id before the interaction can continue is such an example. From our experiment with BLAST sites, it appears that specialized interaction is not that prevalent. However, it might be common in other types of Web interfaces. Recognizing and moving past this type of interaction presents several interesting challenges because of their free-form nature and the number of possible parameters involved. Incorporating a general solution to account for interaction with user input is part of our future work in discovery of source specific interaction patterns.

The majority of the processing errors were failures in handling JavaScript commands found on some sources. A minority are problems in emulating the behavior of standard user agents. A few Web design idioms, such as providing results in a new window or multi-frame interfaces, are not yet handled by either prototype. We are working to make our implementation more compliant with standard Web browser behavior. The main challenge in dealing with processing failures is accounting for them in a way that is generic and does not unnecessarily tie site analysis to the implementation details of particular sources.

6 Conclusion

It is clear that the World Wide Web is an important tool for scientists and researchers. As the Web matures, we expect dynamic Web sources to continue proliferating while also adopting more robust data exchange standards like XML and RDF. We have explored the use of Web sources in the bioinformatics domain and have seen that the increased number of sources promises greater research potential if the data management issues can be overcome. We propose a practical, heuristic approach to classifying arbitrary Web sources using a service class description driven by the need of the users. This alleviates the need to agree on a common ontology and offers the flexibility to incorporate the specific needs of individual information seekers. To improve the accuracy of our classification process, we find it necessary to enhance our classifier to automatically discover source-specific interaction patterns. We have shown how these concepts can be applied in an existing application scenario, Web-based BLAST genome sequence searches. Finally, we have verified our claims experimentally by using a BLAST service class description to identify a group of Web sites.

Our initial results are very encouraging, as our categorization program consistently identified approximately two-thirds of the input URLs correctly. We attribute this success to the regularity of the returned data sets and the observed characteristics of Web sources. Many of the sources have complex interactions that go beyond the single request-response paradigm. We implemented and tested a robust and efficient mechanism call PageDiff to identify interaction patterns that do not require any human input. This improved the effectiveness of our classifier as shown by the additional sites that can be classified correctly as shown in Table 3. The remaining sources that cannot be classified by our approach included sites that require specialized human intervention, that use JavaScript, and a few with quirky interfaces.

We are continuing development of new heuristics for site processing and recognition. In particular, we plan to expand indirection page detection to those that require human input. This will involve simulating and modeling typical human input for common class of indirection pages. While our PageDiff algorithm is working well with the current set of BLAST sites that we know of, we need to extend it to handle dynamic pages that are generated based on cookies or that has a different domain from the URL used to make the initial request. Currently, the service class description used for identifying the source capability needs to be created manually by domain experts who have knowledge of XML Schema. The manual creation of service class descriptions is tedious and error-prone. A graphical tool that allows automatic generation of service class description from a selection of example sites as described in [18] will improve the usability of our system. For each of the sources, our current prototype classifier can only identify its capability and interaction pattern.

However, with the ever increasing number of bioinformatics web sources with similar capability being identified, there is a need to differentiate the source based on its quality. Our classification process needs to be enhanced to discover the quality of the source as well. The system will also be extended to support aggregation of data from hyperlinks—e.g. gene summaries commonly found in BLAST results. Longer-term work will examine applying existing and novel information retrieval techniques to increase the number of recognized sources and further improve performance. For example, an advanced classification system could compare new sources to those it has already classified: if the new source matches a previously discovered source, the information from the existing match can be used to guide analysis of the new source.

References

1. Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Meyers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, October 1990.
2. Arvind Arasu and hector Garcia-Molina. Extracting structured data from web pages. In *Proceedings of ACM/SIGMOD Annual Conference on Management of Data*, pages 337–348, 2003.
3. Y. Arens, C. Knoblock, and W. Shen. Query reformulation for dynamic information integration. *International Journal of Intelligent and Cooperative Information Systems*, 6(2):99–130, 1996.
4. R. Bayardo and et.al. InfoSleuth: Agent-based semantic integration of information in open and dynamic environments. In *Proc. ACM SIGMOD Int'l Conference on Management of Data*, 1997.
5. Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.
6. Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman, and Jennifer Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *16th Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, Japan, 1994.
7. Susan B. Davidson, G. Christian Overton, Val Tannen, and Limsoon Wong. BioKleisli: A digital library for biomedical researchers. *Int. J. on Digital Libraries*, 1(1):36–53, 1997.
8. DBCAT, The Public Catalog of Databases. http://www.infobiogen.fr/services/dbcat/, 2002.
9. Robert B. Doorenbos, Oren Etzioni, and Daniel S. Weld. A scalable comparison-shopping agent for the world-wide web. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 39–48, Marina del Rey, CA, USA, 1997. ACM Press.
10. Barbara Eckman, Zoe Lacroix, and Louiqa Raschid. Optimized seamless integration of biomolecular data. In *IEEE International Conference on Bioinformatics and Biomedical Egineering*, pages 23–32, 2001.
11. David C. Fallside. XML Schema Part 0: Primer. Technical report, World Wide Web Consortium, http://www.w3.org/TR/xmlschema-0/, 2001.
12. W. Gish. BLAST. http://blast.wustl.edu/, 2002.
13. Russell Gold. HttpUnit. http://httpunit.sourceforge.net, 2003.
14. L. Haas, P. Schwarz, P. Kodali, E. Kotlar, J. Rice, and W. Swope. Discoverylink: A system for integrating life sciences data. *IBM Systems Journal*, 40(2), 2001.

15. Allan Heydon and Marc Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, 1999.
16. Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the Twenty-second International Conference on Very Large Databases*, pages 251–262, Bombay, India, 1996. VLDB Endowment, Saratoga, Calif.
17. Robert Miller and Krishna Bharat. SPHINX: A framework for creating personal, site-specific web crawlers. In *Proceedings of the Seventh International World Wide Web Conference*, 1998.
18. G. Modica, A. Gal, and H.M. Jamil. The use of machine-generated ontologies in dynamic information seeking. In *9th International Conference on Cooperative Information Systems, CoopIS2001*, pages 433–448, September 2001.
19. National Center for Biotechnology Information. GenBank Statistics. http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html, 2003.
20. NIAS DNA Bank. Growth of daily updates of DNA Sequence Databases. http://www.dna.affrc.go.jp/htdocs/growth/D-daily.html, 2003.
21. NLM/NIH. National Center for Biotechnology Information. http://www.ncbi.nih.gov/, 2002.
22. Daniel Rocco and Terence Critchlow. Automatic discovery and classification of bioinformatics web sources. *Bioinformatics, Oxfoed University Press*, 19(15):1927–1933, 2003.
23. P. Srinivasan, J. Mitchell, O. Bodenreider, G. Pant, and F. Menczer. Web crawling agents for retrieving biomedical information. In *Proceedings of the International Workshop on Agents in Bioinformatics (NETTAB-02)*, 2002.
24. Vladimir Zadorozhny, Louiqa Raschid, Maria-Esther Vidal, Tolga Urhan, and Laura Bright. Efficient evaluation of queries in a mediator for websources. In *Proceedings of ACM/SIGMOD Annual Conference on Management of Data*, 2002.