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, with Web-based access to distributed data sources such as BLAST sequence homology search interfaces. However, finding the desired scientific information can still be 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. Sharing these new genomics results is hindered by the lack of a common interface or data exchange mechanism. Moreover, the number of autonomous genomics sources and their rate of change outpace 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 in accessing 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 inherent in the design of the data source. We propose a service class description (SCD)-a meta-data approach for classifying 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. Our results show that an SCD-based approach successfully classifies two thirds of BLAST sites with 100% accuracy and two thirds of bioinformatics keyword search sites with around 80% precision.

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 W-7405-ENG-48. UCRL-JC.

** This work was performed while the author was a summer faculty scholar 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 dissemination has increased multi-institution collaboration, which produces dramatically larger data sets than previously available, and requires advanced data management techniques for full utilization.

Some tools that are shared among a large number of institutions have become *de facto* standards. 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 [23, 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 those alreday known, to reduce duplication, and to have their work validated by other members of the community. The rapid addition of new sequences [22, 21] 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. 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:

- The scientist may not query the most relevant sites for their search.
- The search must be entered multiple times.
- Results of the search must be merged together by hand to obtain an integrated set of results.
- If an interface changes or moves, the scientist must ascertain where the new interface is and how to query it appropriately.

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 navigation path.
- Construct a wrapper for the source.

- 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].

We observe that the correct classification of a Web source involves identifying the capability of the source as well as its correct interaction pattern. 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 the end-user receives the answer page. Section 1.1 illustrates a scenario where a sequence of indirection pages are used to present the result from a typical BLAST source. In our initial experience in the classification of BLAST Web sources, at least 10% of these 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 main contributions of this paper are:

- A practical, heuristic approach to classifying arbitrary Web sources using a service class description (SCD) driven by the application need of users. This alleviates the need for a global ontology and offers the flexibility to incorporate the needs of individual information seekers.
- 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. None of the current source classifiers deal with automatic navigation of multiple pages. Even current state-of-the-art wrapper generators do not deal with intermediate cross page navigation during data extraction.

This system is part of an ongoing research effort to build an automated integration system for Web sources at Lawrence Livermore National Laboratory. To concretely demonstrate the approach, we focus on BLAST interfaces. However, these flexible techniques are generic and can be applied to other domains. In particular, we also demonstrate how a bioinformatics keyword-based search interface for protein and nucleotide sequences can be defined.

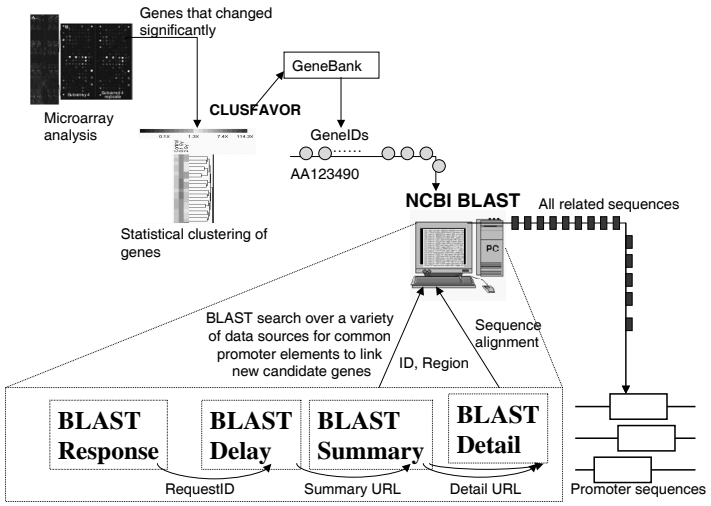

Fig. 1. A scientific data integration example with indirection.

1.1 Example Scenario with Indirection Page

The scenario in Figure 1 illustrates the need to deal with a complex interaction pattern for large-scale data integration in genomics. It illustrates a routine task that a biologist performs on a regular basis. In the first step, the biologist uses a program called CLUSFAVOR to cluster genes that changed significantly in a micro-array analysis experiment. After extracting all gene IDs from the Clusfavor result, the biologist needs to check the sequence of each of those genes using a source like GenBank. Once the complete sequence for a relevant gene has been retrieved, it is sent to a gene matching service (e.g., NCBI BLAST) that will return homologs-other genes with similar sequences. There are a large number of BLAST sources (over 500). Each source might specialize in a particular type of species, with a particular quality of data content, form of input and output, and pattern of interactions. For example, performing a BLAST search using the popular NCBI BLAST (http://www.ncbi.nlm.nih.gov/blast) requires fours steps. The NCBI server will first return a response page with a request ID. When the biologist asks for the results by the request ID, the Website will present a delay page if the result is not yet ready to display. Once the search results are delivered, they are displayed in a summary page that contains a summary of all genes matching the search query condition and links to detail pages of each found gene. If the summary page does not include the information of interest, the biologist has to visit each detail page. The same request on another known BLAST source located at the bioinformatics laboratory at Stanford University (http://www-sequence.stanford.edu:8080/bncontigs6.html) only requires a single step to get the summary page. Both sites produce similar summary pages, but the NCBI BLAST site involves

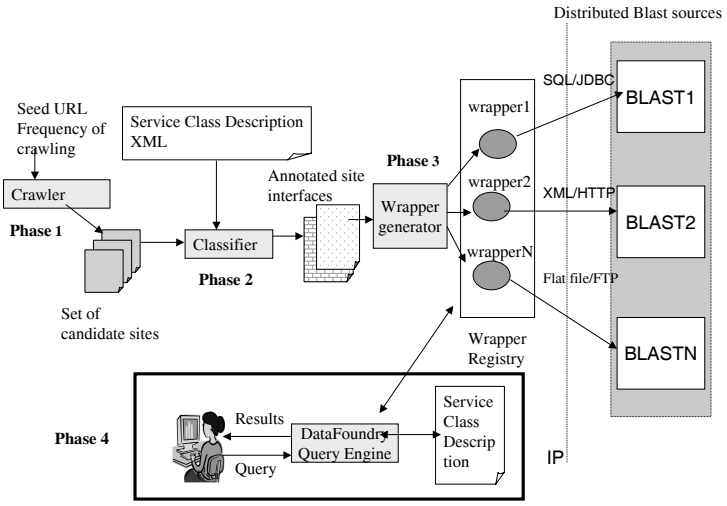

Fig. 2. Architecture of DataFoundry.

navigation of multiple pages (indirection pages) to get to the summary page. It is important to be able to access the summary page of both BLAST sources regardless of site specific differences in navigation and data access.

2 Overview of DataFoundry-A Large-Scale Data Access System

The automated classifier described here is a part of the large-scale data access system called DataFoundry, developed at Lawrence Livermore. A goal of DataFoundry is to produce an automated system that can find, classify, interact with, and wrap new and changed Web sources without costly human intervention. Figure 2 shows the overall architecture of the project. We divide the process of integrating and accessing a large number of heterogeneous bioinformatics data sources into four phases.

- The first phase is a *discovery service* that locates candidate information sources from the Web.
- The second phase is a *classification service* that filters candidate sources by using an SCD to check client requirements such as capability of the source and the interaction patterns.
- The third phase is an *extraction service*, bringing desired information from the relevant sources in a convenient format to users.
- The fourth phase is an *interface service*, consisting of an integrated application programmer interface (API) or graphical user interface (GUI) that provides a uniform interface to extracted information from various information sources of interest.

Our focus here is on the classification service, which is further divided into discovering the source's capability and interaction pattern. The discovery service is implemented as a spider or a crawler agent that is capable of following links and interacting with HTML forms. To maintain modularity in the system architecture, we separate the discovery service from the classification service. This allows us to incorporate a better crawler, if available. We are **n**ot addressing the extraction service which is reponsible for generating the wrappers; however, the output from our classifier is intended to serve as input to XWRAP [18], an existing wrapper generation system. The integration of data across the different Web sources is performed by the interface service, and is also not the focus here. However, the SCD used for the classification will serve as the "global ontology" for this phase.

3 Discovering a Web Source's Capability

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 a 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 [24].

3.1 Service Class Descriptions

Service classes are specified by an SCD, which uses an XML format to define the relevant functionality of a category of Web sources from an application's perspective. The SCD format supports four categories of information used to define an interface of interest:

- Data types.
- Input parameters.
- Control flow.
- 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] and includes constructs for building *atomic types* and *complex types*. Atomic types are simple, valued data elements such as strings and integers. The XML Schema system provides several built-in atomic types that can be used to create user-defined types defined by restriction. The `DNASequence` type in Figure 3 is an example of an atomic type

defined by restriction in the nucleotide BLAST SCD. Figure 3 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.

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

<type name="AlignmentSequenceFragment" >
    <element name="AlignmentName" type="string" pattern=".{1,100}:" />
    <element type="whitespace" />
    <element name="beginMatch" type="integer" />
    <element type="whitespace" />
    <element name="Sequence" type="DNASequence" />
    <element type="whitespace" />
    <element name="endMatch" type="integer" />
</type>

<type name="AlignmentString">
    <element DNAAlignmentString" type="string" pattern="\s+\|+[| ]*" />
</type>

<type name="Alignments" >
      <element name="QueryString" type="AlignmentSequenceFragment" />
      <element type="string" pattern="\s*" />
      <element type="AlignmentString" required="true"/>
      <element type="string" pattern="\s*" />
      <element name="SequenceString" type="AlignmentSequenceFragment" required="true" />
</type>
<type name="SummaryResults">
      <choice>
          <element type="Alignments" />
          <element type ="EmptyDNABLAST"/>
      </choice>
</type>
```

Fig. 3. Sample nucleotide BLAST type definitions.

Composition of elements into complex types can be in the form of a simple sequence of elements, such as the `AlignmentSequenceFragment` definition. List definitions are also allowed using

the constraints `minOccurs` and `maxOccurs`, which define the expected cardinality of a particular sub-element within a type. The `choice` operator allows types to contain a set of possible sub-elements from which one will match. Figure 3 shows examples of complex types called `Alignments`, `SummaryResults`, and `AlignmentSequenceFragment`. The `SummaryResults` type is defined as a choice of either `EmptyDNABLAST` or `Alignments`. Figure 4 shows an instance of the `Alignments` type for a BLAST source, and Figure 5 shows an instance of the `EmptyDNABLAST` type.

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

Fig. 4. Example of a nucleotide alignment.

```
*** No Hits ****
```

Fig. 5. Example of an instance of EmptyDNABLAST.

Control flow graphs enumerate 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. To be as general as possible, the control flow graph does not include either delay pages (e.g., "Your 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 6 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 with a nucleotide sequence input parameter. 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 service class member inferred during interface classification.

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

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

Fig. 6. Control flow graph definition.

Figure 7 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" in its name. The DNA Sequence argument in a nucleotide BLAST service class therefore includes hints of "sequence," "query_data," and "query," with type "text."

If a user wishes to discover a more complex type of BLAST servers, that is, one that includes choice of alignment parameters, additional arguments with suitable hints need to be added to the example specification.

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. We summarize the approach that we take for identifying members of the service class in the following paragraphs. Refer to Rocco and Critchlow [24] 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

```
<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. 7. A nucleotide BLAST example.

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 SCD. For each response, the classifier follows the outbound links and tests the responses of the source against possible states in the control flow graph. This process continues until either the site matches one of the end states in the flow graph or there are no more possible queries to try. Our initial prototype of this source classifier was unable to infer source-specific interaction patterns that were more complex than a start 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 (described in Section 4).

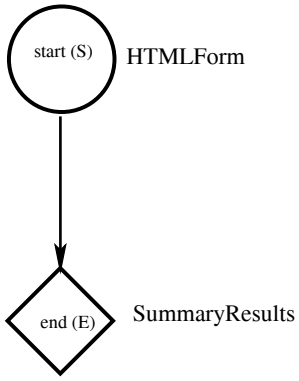

Fig. 8. Control flow graph for a nucleotide BLAST service.

The control flow graph for a nucleotide BLAST service is shown in Figure 8. In the control flow graph, start and end states are represented with a circle and diamond respectively. For the classification of BLAST sources, only the start and end states are required to be specified in the SCD. 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`, as shown in Figure 3.

If the Web source's start page 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 end state result. 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 9 is an example

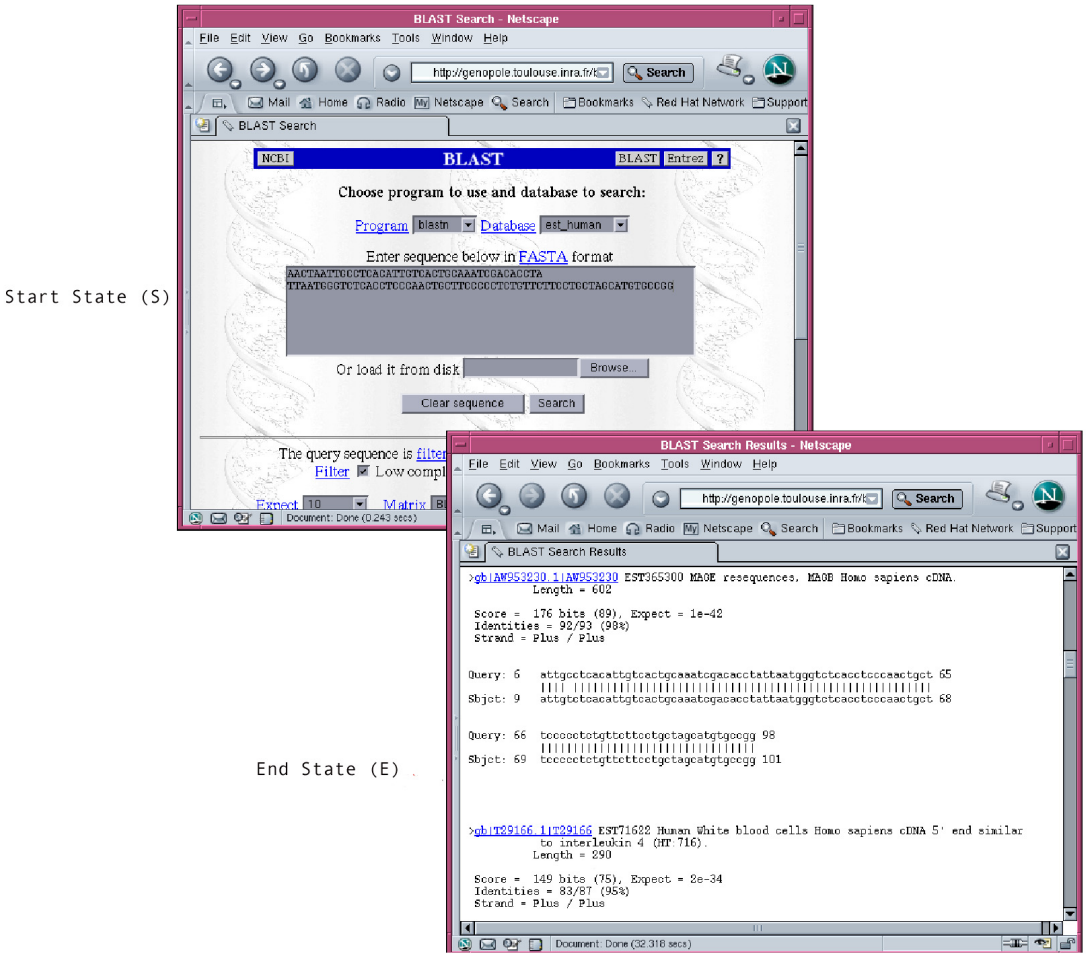

Fig. 9. A potential Web source that matches the nucleotide BLAST service control flow graph.

of a BLAST interface that matches the control graph definition shown in Figure 8. Note that 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 the next section, this task can be significantly complicated by the presence of indirection pages that 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 SCD 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 the initial effort to classify DNA BLAST sites using our prototype implementation, we found that many sites, including the popular NCBI BLAST, 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, and initially propose a simple heuristic-based approach to identify those pages. The shortcomings of this approach are then discussed, which 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* that will eventually lead to the expected results. The pointer could be a form that needs to be submitted (Figure 10) or an HTML reference link (Figure 11) that needs to be clicked on. After conducting an in-depth

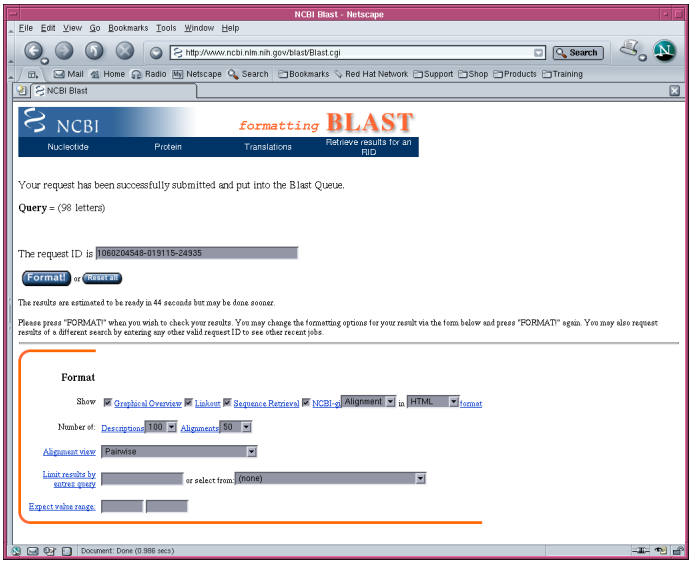

Fig. 10. An indirection Page from the NCBI BLAST site (clicking on the format button will lead to the result summary page).

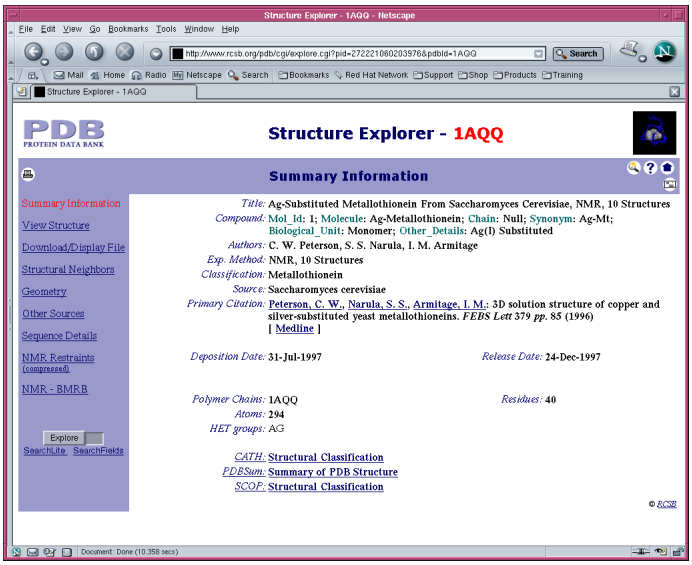

Fig. 11. An indirection page from the Protein Data Bank site (clicking on the download/Display File link will lead to the result summary page).

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.
- An indirection page can be as simple as an HTML page that contains a single HTML form (site 7 in Table 1) 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 a 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 site 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.

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 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 to by each link, and check the 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 SCD is found. Such an approach is infeasible because:

- It is computationally very expensive. The example indirection page shown in Figure 11 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 the website denying access.
- 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 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 interaction with the Web interface. Note that we are dealing with dynamic Web pages or the hidden Web, which current Web-crawling technology does not cover. The result of our research here can be used to complement technology for crawling the deep Web.

4.2.1 Simple Heuristic Approach to Discovery of Dynamic Links

Observing that a typical indirection page has both static and dynamically generated information, we developed the following heuristics for identifying *dynamic links*. The goal is to eliminate those links in an HTML page that will not contribute to a correct interaction.

1. An HTML form is a dynamic link that should always be followed first. This assigns an HTML form with a higher priority dynamic link than an HTML reference link.
2. An HTML reference link that ends with a file extension such as ".css," ".ps," ".jpeg," ".zip," ".pdf," ".gif," etc. is not a dynamic link,
3. An HTML reference link that does not begin with an 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.

The first heuristic 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. The second uses our observation that dynamic links are generated on the fly and any HTML reference links that ends with one of the above listed extensions typically are not dynamic links and can be eliminated. The third heuristic eliminates links that point to content that cannot be downloaded using HTTP protocol and hence cannot be checked. The fourth 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 sites (5 and 9 in Table 1) 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 3 failed because the indirection page requires specialized human intervention. Using this simple heuristic for identifying indirection pages, we were able to reduce the potential set of HTML pages to check from 421 to 83 for the PDB example shown in Figure 11. Despite its performance advantage 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 develop 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 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 12. It consists of three main steps: the first chooses and posts queries, the second computes the page differences, and the third step recursively identifies sets of dynamic links to follow. During classification, 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

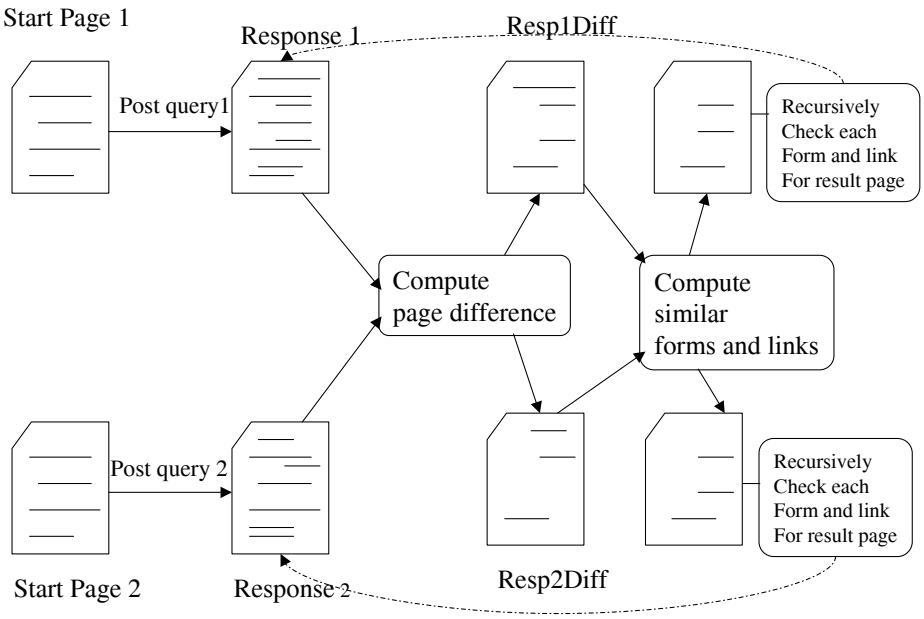

Fig. 12. PageDiff algorithm for computing minimal sets of dynamic links to check for indirect pages.

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 values 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 recursively in pairs 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 page difference computation between two uncorrelated HTML pages might result in a set of dynamic links that equal all the outbound

links in an HTML page. This degenerates to a naive approach for identifying indirection pages. The criteria for computing similarity for HTML forms and HTML reference links are discussed below.

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, 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 13 shows an example of a URL and its parts.

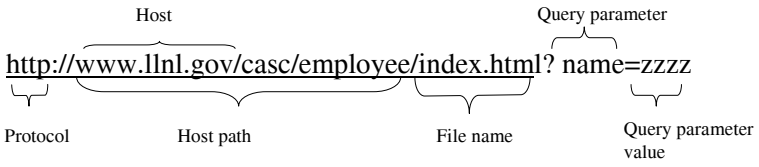

Fig. 13. Format of a URL.

We define two HTML reference links that have a list of query parameters to be similar if their host paths, file names, and 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: First, if their host paths match, their file extensions match, and the label of their links match (e.g., both are displayed as "blast" in the browser), we consider the two links to be similar. The following two URLs are an example of such a match:

```
<a href="http://www.xxx.yyy.org/blast/A123456.html">blast</a>
<a href="http://www.xxx.yyy.org/blast/A125678.html">blast</a>
```

In the second case, 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:

```
<a href="http://www.xxx.yyy.org/blast/tmp/A123456/index.html">blast</a>
<a href="http://www.xxx.yyy.org/blast/tmp/A234567/index.html">blast</a>
```

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 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

Here we briefly compare robustness and efficiency of the simple heuristic and PageDiff approach to identifying 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 paired comparisons of HTML pages until the result summary page is found. It also incurs higher network and disk input/output because two HTML pages must be downloaded each time. To evaluate the amount of overhead, we conducted experiments using the ten BLAST sites identified in 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 an 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 the heuristic approach by 336% as shown by sites 1, 4 and 8. In the case of the Protein Data Bank 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, while the 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 Evaluation

Our source classifier prototypes are implemented in Java and can examine a set of supplied URLs obtained from our crawler in the *discovery service phase*. Interaction with the Web is handled by the HttpUnit user agent library [13].

Experiment 1 shows the results of identifying BLAST sites using the BLAST SCD, while Experiment 2 shows the results of identifying bioinformatics keyword search sites using the BioKey SCD. The standard recall and precision in information retrieval will be used to evaluate the effectiveness of our approach. The goal is to achieve high recall without sacrificing precision. Recall is computed according to Equation 1:

$$Recall = \frac{\text{relevant sites identified}}{\text{total relevant sites}} \tag{1}$$

`Relevant sites identified` is the same as the sites that are labeled as "True Positive" in our results tables below. False negative sites are those that are relevant, but which the classifier failed to identify. `Total relevant sites` is the sum of the true positive sites and the false negative sites in our experiment. Precision is computed according to Equation 2:

$$Precision = \frac{\text{relevant sites identified}}{\text{total sites identified}} \tag{2}$$

`Total sites identified` is the sum of the true positive sites and the false positive sites. The higher the number of false positive sites, the lower the precision.

5.1 Experiment 1: BLAST Service Class Descriptions

The data for this experiment consists of 35 URLs (of which 27 are BLAST) we used as a test bed and another 150 distinct URLs used for the final experiment. The use of distinct URLs is important to ensure that we have a random set from different sites. There were 12 sites that were manually determined to be nonfunctional or that returned results exclusively via email and were excluded. Of the 138 URLs used in the final experiment, 92 were BLAST URLs. Those URLs were gathered from Web crawling, and their interfaces vary widely in complexity: some have forms with fewer than five input parameters, while others allow minute control over many 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 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 BLAST SCD of approximately 150 lines.

5.1.1 Results from Experiment 1

Table 3 shows the results of our experiments. The initial test set is Web sources that were tested repeatedly as the prototype matured and helped shape its design. The remaining sources of the

Data Set	True Negative	True Positive	True Positive with Indirection	False Negative		False Positive	Recall	Precision
				Specialized	Processing			
Initial test set	8	18	3	1	5	0	77.7%	100%
Experimental set	46	64	5	1	22	0	75.0%	100%

Table 3. Sites classified using the nucleotide BLAST SCD.

experimental set were classified once. Sites listed as true positive were those that could 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 (Figure 5). Sites listed under "True Positive with Indirection" are those additional ones 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.

False negative were sites that failed to be classified correctly. They fall into two categories: specialized interaction and processing failures. A specialized interaction source is an indirection page that requires further specific user input 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.

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 our prototypes. 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.

We do not encounter any false positive cases in Experiment 1. This is attributed to the regularity and uniqueness of DNA alignments that are generated exclusively by BLAST servers. We are able to classify the functionality of a BLAST source with above 75% recall and 100% precision.

5.2 Experiment 2: BioKey Service Class Descriptions

The second experiment was conducted to identify bioinformatics keyword-based search sites for protein and nucleotide sequences. For ease of reference, we use BioKey to refer to this class of sites. The interfaces for BioKey sites are radically different from BLAST sites. A Web source is

considered to be a BioKey site if it has a keyword-based HTML form for input and returns the result in an HTML page with a list of pointers to files that have detailed information for the protein or nucleotide sequences. Each pointer is an HTML link indexed by either protien ID code (PDBID) or accession number. Figure 15 shows an example of a protein BioKey site and Figure 16 shows an example of a nucleotide BioKey site in response to a search using "HIV" as the keyword. The HTML links can come in a variety of formats. Some BioKey sites displayed the HTML links as a vertical list, other displayed them as rows in a table, and some others displayed them as a horizontal list of items. Figure 14 shows a partial definition of a BioKeyword SCD. Note that `PDBIDType` is a four-letter alphanumeric code that starts with a digit, and `AccessionNumberType` is either a two-character code followed by six digits or a one-character code followed by five digits. The `SummaryResults` page is defined as a choice of either `PDBLink` or `NucleotideLink`. We recognize that in the bioinformatics domain, a keyword search can return the results in an HTML page with a list of pointers indexed by other IDs (e.g. MMDBID, Genome ID, MIM number, CDS identifier). If the user wants to identify results pointed to by other IDs beside protein id code and accession number, then additional types can be added to the BioKeyword SCD.

5.2.1 Results from Experiment 2

The number of available BioKey sites on the Internet are much less than for BLAST. The test bed used consists of 20 URLs, of which 9 are relevant BioKey sites. The rest are popular keyword search sites such as Yahoo, Google, Amazon, etc. We use this test bed to fine tune the initial definition of our BioKey SCD. Three sets of URLs are then used for our classification experiments.

- Set 1 contains 89 URLs crawled with http://www.infobiogen.fr/services/dbcat as a seed URL.
- Set 2 contains 89 URLS crawled with http://www.rcsb.org/pdb as a seed URL.
- Set 3 contains 46 URLs crawled with http://www.ncbi.nlm.gov as a seed URL.

For each set, only a minority of the URLs actually pointed to the the type of sites we were looking for. Table 4 shows the results of Experiment 2. Notice the presence of false positive cases in this experiment. False negative sites are mainly because of the inability of of the HttpUnit agent to deal with complex JavaScript. None of the BioKey site requires specialized interaction.

Recall and precision are computed using Equations 1 and 2. We encountered lower precision in Set 1 and the initial test experiments because of the false positive cases. We found that unlike Experiment 1, it is not possible to define a BioKey SCD that can eliminate all false positive cases without using an exhaustive list of all valid protein codes and accession numbers in this domain. The `PDBIdType` and `AccessNumberType` defined using regular expressions are not unique to BioKey sites.

```
<type name="AccessionNumberType" type="string" pattern="([A-Z]{2}_?\d{6}|[A-Z]\d{5})"/>

<type name="AccessionNumber">
    <element type="string" pattern="[^A-Za-z0-9]*?" />
    <element type="AccessionNumberType" />
    <elemen type="string" pattern="[^A-Za-z0-9]"/>
</type>

<type name="PDBIDType" type="string" pattern="[1-9][A-Za-z]{1}[A-Za-z0-9]{2}"/>

<type name="PDBID">
    <element type="PDBIDType"/>
    <element type="string" pattern="[^A-Za-z0-9]"/>
</type>

<type name="HREF">
    <element type="string" pattern="\x3C(a|A)[^\x3E]*(href|HREF)=[\x22\x27]"/>
    <element name="Link" type="string" pattern="[^\x22\x27]+"/>
    <element type="string" pattern="[\x22\x27][^\x3E]*\x3E"/>
</type>

<type name="PDBLink">
    <element type="HREF"/>
    <element name="proteinCode" type="PDBID"/>
</type>

<type name="NucleotideLink">
    <element type="HREF"/>
    <element type="ValidPrefix" optional="true"/>
    <element name="accessionNo" type="AccessionNumber"/>
</type>

<type name="SummaryResults">
      <choice>
          <element type="PDBLink"/>
          <element type ="NucleotideLink"/>
      </choice>
</type>
```

Fig. 14. Sample BioKey service-class-description type definitions.

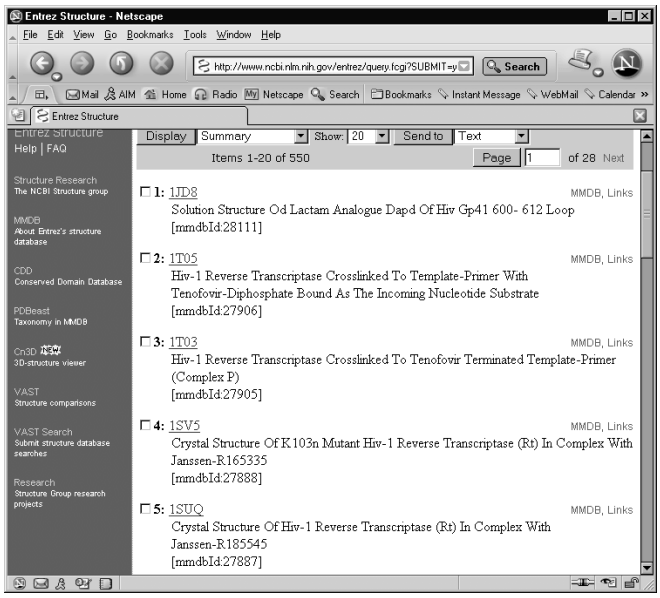

Fig. 15. A Summary Page for Protein BioKey site

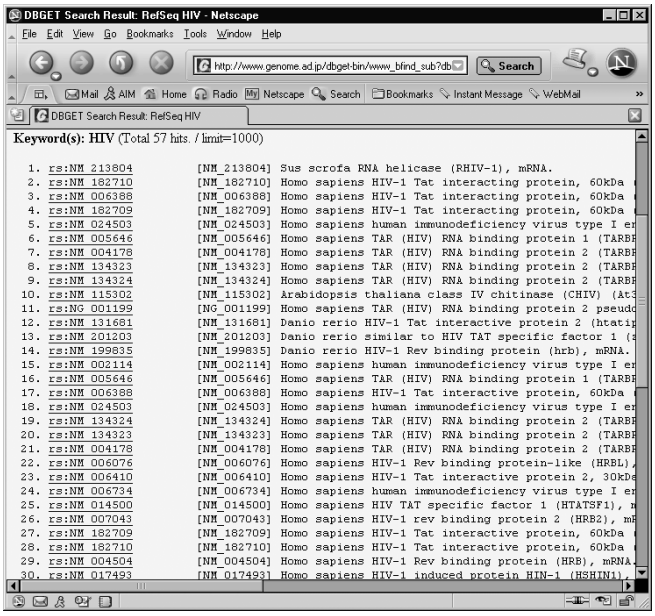

Fig. 16. A Summary Page for Nucleotide BioKey site

For example, a Google search site could return an HTML link pointed to by a four-letter acronym. Despite the lower precision, our recall rate is still acceptable. For all three of the experimental sets, we are able to discover new BioKey sites.

Data Set	True Negative	True Positive	True Positive with Indirection	False Negative		False Positive	Recall	Precision
				Specialized	Processing			
Initial Test Set	10	7	1	0	1	1	87.7%	87.7%
Experimental Set 1	81	3	2	0	2	1	71.4%	83.3%
Experimental Set 2	82	5	0	0	0	0	100.0%	100%
Experimental Set 3	43	2	0	0	1	0	63.3%	100%

Table 4. Sites classified using the BioKey SCD.

6 Application of Service Class Descriptions to Other Domains

BLAST site interfaces and BioKey site interfaces are completely different in the input requirement and the result summary page, but we can use the same mechanism to classify both kinds of sources. The only requirement is the ability to define an SCD that represents the meta information about sites of a specific class. The use of SCDs has proved to be very effective for classifying sites that have a unique and regular output format. For example, the DNA alignments are very unique to BLAST sources, and thus we can achieve 100% precision in identification.

The protein code and the accession number are not unique only to BioKey sites. There is nothing to prevent an arbitrary dynamic Website from generating HTML links that are pointed to by a four-letter acronym. Moreover, the output format from BioKey sites is not as regular as for BLAST sites. While every BLAST site that matches the input DNA sequence will output the alignments in a regular pattern, the output from a BioKey site can appear as a horizontal or vertical list of HTML links as well as appearing as rows in a table. No consistent presentation template can be identified across different BioKey sites. We have shown that even with these constraints, an SCD with 120 lines of code can still achieve reasonable classification precision.

The SCD is based on a regular expression. It describes not only the type of input and output of a site but also the functionally (the input queries and the control graph) of a site. The ability to capture the functionally of a site together with the input and output requirements make it a powerful paradigm for classification of Web sources. However, SCDs are not suitable for Web sources whose output does not exhibit a regular pattern and cannot be described uniquely using a regular expression, such as a site whose output consists of a list of publications where each item

(title, author, journal/proceeding, date, location) can all literally be described using a character string, and listed in different order. Moreover, it is impossible to use a regular expression to describe the title of a scientific paper in a unique way.

7 Related Work

Our work is inspired by the ShopBot agent [9], whose purpose is to assist with online shopping. ShopBot uses a domain description that lists useful attributes about the services in question. The ShopBot authors addressed the problems of discovering unknown vendor sites and integrating a set of learned sources into a single interface. Our work addresses the related problem of automatically classifying services from an arbitrary set of sites. Our SCD format provides greater descriptive power than ShopBot's domain descriptions. For example, we can specify composition of different data types as well as functionalities of a Website in terms of its input and output requirements. Shopbot assumes that all potential sites interact in a simple request-response paradigm. Our approach also addresses the varied interaction patterns embedded in vendor sites of a particular domain.

Machine-generated ontologies are used in [20] to allow a more focused approach to seeking information on the Web for a specific domain. The methodology is based on a set of heuristics from an information-retrieval domain, and is used to infer the type (schema) of a large collection of Web interfaces of a particular domain. This approach is similar in spirit to using an SCD to discover a Website's capability. However, in their approach the interaction pattern, which they called precedence relationships, cannot be inferred automatically. It requires manual annotation of hidden Website interaction patterns. They also do not address generation of ontologies from Web interface output. It is dangerous to assume that interfaces with the same input ontologies will also contain the same output ontologies.

ExALG [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. It is an algorithmic approach that deduces the template for data extraction from a given set of template-generated HTML pages. The generated template is useful for automatic generation of a wrapper for a specific site, but it does not provide a semantic description of a class of Websites. Manual postprocessing labelling is required to use the template for discovering site capability. The result of our classification process is an annotation map that is comparable to ExALG's generated template, without the assumption of being useful for only template-generated HTML pages. From our empirical observation of genomics sites, no regular template exists across different sites of the same class. Our SCD is used to describe the common properties across a class of Websites in a specific domain. It is invariant to a site-specific template.

The focus of ROADRUNNER [26] is on an automatic wrapper generation for a specific Web source by probing the site and comparing generated HTML pages. ROADRUNNER assumes that potential Websites are known a priori for wrapper generation. Our goal is to discover potential sites, using our SCD to interact with them and generate site-specific profiles (annotation map) as input for automatic wrapper generation tools. Our approach, when coupled with a wrapper generator tool, can generate a large number of wrappers for a class of Web sources without any manual intervention.

The work by Knoblock [16] also addressed the automatic wrapper generation by exploiting formating information in dynamically generated HTML pages. Our classifier also exploits the regular output pattern in dynamically generated HTML pages, however our approach does not depend on any HTML-formating information, which varies widely across Websites and changes frequently.

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. [27] describe a wrapper and mediator system for limited-capability Web sources that includes query planning and rewriting capabilities. Information Manifold [17] targets myriad 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 system [6] 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 nontrivial to define a common ontology (one that is used by all members of the community) for integrating different Web sources. This requires semantic differences between different sources in a particular domain to be resolved first. In bioinformatics, a global and unified ontology[1] is not available because of rapid advancement in biotechnology. Again, in the SIMS project [3], a common ontology and query language are used to facilitate the combination of information from heterogeneous data sources. The cost associated with creating and maintaining a global ontology outweigh its advantages when dealing with large and dynamic sources such as the WWW.

Researchers have also examined federated heterogeneous data integration in the domain of biological data. DiscoveryLink [14] uses the relational model and SQL language for providing 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] adopts a nested relational model, extends SQL, and provides access to complex sources with structured

[1] There are fragmented and distinct ontologies available in bioinformatics, but no single global ontology available.

data. Both DiscoveryLink and BioKleisli have the advantage of providing local autonomy, accessing up-to-date data from different sources at run-time, and giving the user a single unified point of access. However, both these approaches still require manual coding of source-specific wrappers, and thus present a scalability problem when used for a large-scale integration effort. They also assume that sources that need to be integrated are identified a priori.

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. We focused 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. An automatic classification of sources is critical to the construction of a large-scale data access system.

Automatic classification of Web sources apropos of a particular domain involves both locating sources and determining their relevance to the domain; we address 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. For conciseness, we don't elaborate on the crawling aspects of Web sources. The research challenges and implementation for Web crawling have been addressed in the literature and commercially [5, 19, 15]. There is also active research into topic-driven or focused crawlers; Srinivasan et al. [25] present such a crawler for biomedical sources that includes a treatment of related systems.

8 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 bioinformatics 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 an SCD driven by the functionality of a class of applications. 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 verified our claims experimentally by using a BLAST SCD to identify BLAST servers and a BioKey SCD to identify bioinformatics keyword-based sites.

Our initial results are very encouraging: 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 the 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 additional sites that could be correctly classified. The remaining sources that could not be classified included sites that require specialized human intervention or use JavaScript, and a few with quirky interfaces. We have provided a short discussion on the kind of sites that work particularly well using our approach and those that do not. In general, regular expressions are not a good mechanism to use when no regular pattern can be observed in the output page or in the data to be matched. We are currently investigating machine-learning techniques to generate rules that can express the wide variation and yet similar format of various Websites of a particular class.

We are also developing a 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 a common class of indirection pages with specialized interaction. While our PageDiff algorithm is working well with the current set of BLAST sites, we need to extend it to handle dynamic pages that are generated based on cookies, which requires posting of two different queries for PageDiff computation. Currently, the SCD used for identifying source capability needs to be created manually by domain experts who have knowledge of both XML and regular expression. The manual creation of SCDs is difficult and error-prone. A graphical tool that allows automatic SCD generation from a selection of example sites, similar to that described in [20] will improve usability.

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. Craig A. Knoblock, Steven Minton, Jose Luis Ambite, Naveen Ashish, Ion Mulsea, Andrew G. Philpot, and Sheila Tejada. The ariadne approach to web-based information integration. *International Journal of Cooperative Information Systems (IJCIS)*, 10(1-2):145–169, 2001.
17. 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.
18. Ling Liu, Calton Pu, and Wei Han. XWrap: An XML-enabled Wrapper Construction System for Web Information Sources. *Proceedings of the International Conference on Data Engineering*, 2000.
19. 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.

20. 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.
21. National Center for Biotechnology Information. GenBank Statistics. http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html, 2003.
22. NIAS DNA Bank. Growth of daily updates of DNA Sequence Databases. http://www.dna.affrc.go.jp/htdocs/growth/D-daily.html, 2003.
23. NLM/NIH. National Center for Biotechnology Information. http://www.ncbi.nih.gov/, 2002.
24. Daniel Rocco and Terence Critchlow. Automatic discovery and classification of bioinformatics web sources. *Bioinformatics, Oxford University Press*, 19(15):1927–1933, 2003.
25. 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.
26. G. Mecca V. Crescenzi and P. Merialdo. Towards automatic data extraction from large web sites. In *Proceedings of the 27th International Conference on Very Large Data bases*, September 2001.
27. 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.