Automatic Composite Wrapper Generation for Semi-Structured Biological Data Based on Table Structure Identification*

Liangyou Chen‡ Hasan M. Jamil† Nan Wang‡
‡Department of Computer Science and Engineering, Mississippi State University, USA
†Department of Computer Science, Wayne State University, USA
cly@cse.msstate.edu, jamil@cs.wayne.edu, nw43@cse.msstate.edu

ABSTRACT

Biological data analyses usually require complex manipulations involving tool applications, multiple web site navigation, result selection and filtering, and iteration over the internet. Most biological data are generated from structured databases and by applications and presented to the users embedded within repeated structures, or tables, in HTML documents. In this paper we outline a novel technique for the identification of table structures in HTML documents. This identification technique is then used to automatically generate composite wrappers for applications requiring distributed resources. We demonstrate that our method is robust enough to discover standard as well as non-standard table structures in HTML documents. Thus our technique outperforms contemporary techniques used in systems such as XWrap and AutoWrapper. We discuss our technique in the context of our *PickUp* system that exploits the theoretical developments presented in this paper and emerges as an elegant automatic wrapper generation system.

1. INTRODUCTION

The importance of automatic wrapper generation for biological database interoperation has been well recognized in recent research by a number of leading research groups. The massive efforts in database integration were motivated by several important factors including the need for large scale data analysis and distributed resource integration in post-genomic era.

The need for automation can be justified in many different ways. Two major factors stand out – the lack of technical sophistication of the end-users of such databases, and by the ad hoc nature of the applications or queries run by them. At the application level, automation helps end-users to approach distributed resource integration on their own possibly with the aid of intelligent tools. These tools could generate wrappers in a stepwise fashion resolving any ambiguities along the way with the help of the user. It was shown in many research that in such a set up, an intelligent tool can perform well with minimal help from an average user when faced with system limitations in resolving ambiguities. It is our thesis that in a nearly homogeneous application domain, such as genomics, this assumption[1] holds true and is supported by experimental evidence.

At the technical level, automation can be justified as follows. First, manual wrapper generation is a tedious task and prone to error. Despite risks of error, manual wrapper generation can be made much more effective and reliable. But the cost of generating practical wrappers manually may be prohibitive. Many reliable techniques now exist for concept identification and matching in digital documents needed for wrapper induction. But the caveat is that there exists situations when all such techniques fail. Human intelligence can usually be applied to handle such situations and improve the functionality of the autonomous system. Secondly, when ad hoc queries over distributed resources are concerned, integration itself becomes practically ad hoc, and in such situations manual integration or wrapper generation is impractical due to cost factors. So, a user assisted automatic integration and wrapper generation is highly preferred if high precision at a throw-away cost can be guaranteed.

Our goal in this paper is twofold. First to demonstrate that automatic composite wrapper generation is feasible for homogeneous domains, and second, to present a procedure to identify table structures in HTML documents. We argue that table structure recognition is an important ingredient for the generation of any composite wrapper. In an attempt to convince the reader of the importance of our goals, let us consider an application that finds the Homo sapiens genes for the ovarian tissues from The Cancer Genome Anatomy Project (CGAP) database at NCBI for some cancer research. A search for such genes using the *gene finder* tool at CGAP site may display the table shown in figure 1.

If we are interested in picking up all the corresponding gene sequences from the GenBank, we would need to collect all the Sequence IDs in the third column, and read the sequences from the database. If, however, we are required

* This research was supported in part by grants EPS-0082979 and EPS-0132618 from the National Science Foundation, USA.

[1]The assumption being that automated and accurate wrapper generation with little or no help from the end-user is possible in homogeneous domains.

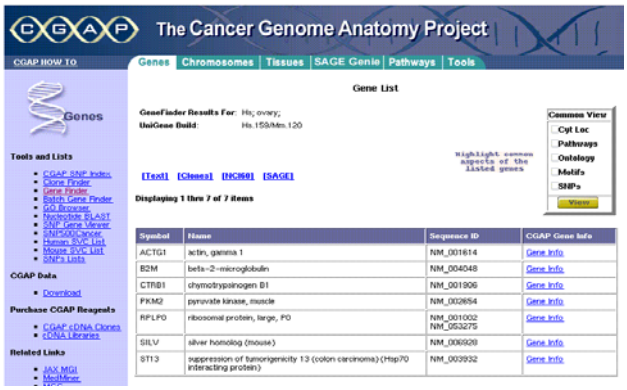

Figure 1: The set of Homo sapiens ovarian tissue genes found in NCBI CGAP Database.

to collect all the sequences corresponding to the full-length MGC clones for each of the genes, we need to follow the links in the CGAP Gene Info column to visit the subsequent pages to collect the accession numbers, and then the sequences from the GenBank database. Figure 2 shows one such table for the first sequence *NM_001614* in figure 1. Readers may have noticed that the figure 2 contains yet another table along with other information.

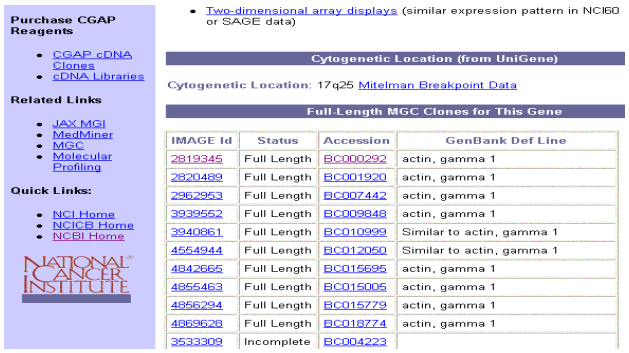

Figure 2: MGC Clones for Gene NM_001614 shown in figure 1.

In both the pages, identification and manipulation of the table structures are the key operations. Consequently any wrapper generation algorithm must be equipped to do just that. It is important to remark here that the apparent simplicity of the HTML documents[2] is really deceptive. It is well known that tables in general can be captured in many different ways in HTML, possibly using non-standard techniques. Hence identification of HTML record structures in general is not a trivial problem, as it will be evident from the discussions in this paper.

This paper introduces a fully automatic wrapper generation system, called *PickUp* [4], for HTML documents based on table structure identification. We organize the presentation of the techniques involved in table structure identification as follows. In section 2, we discuss relevant recent research and compare our approach with several leading research on intuitive grounds. We introduce our *PickUp* system in section 3 and discuss its relationship to life sciences. In particular, in section 3.2 we discuss how table structure identification helps integrate life sciences resources in an ad hoc fashion. Then in section 4, we introduce the so called hierarchical repeated structure identification technique for table structure identification. In this section we discuss relevant theoretical backgrounds and a method that exploits these concepts for automated wrapper generation. This section also reports our automatic wrapper induction system called *PickUp*. We also present two examples to show the effectiveness of our method over leading techniques. We then finally summarize in section 6.

2. RELATED RESEARCH

Automatic and manual wrapper generation has been a subject of intense research for several years. However, in the context of world wide web and web services, research in wrapper generation have received renewed attention. For the sake of brevity, we will not present a comprehensive discussion on recent advances in this area. Instead, we refer interested readers to surveys such as [7, 16] for an introduction to this subject. In this paper, we will only discuss most relevant research on automatic and semi-automatic wrapper generation and compare our approach with these research because we too take an automatic approach.

Although many other systems also induce wrappers automatically, the induction methods either rely on a large set of training pages, complicated induction rules, or pattern discovery from candidate sources. We induce wrappers from a single web sources without the application of induction rules, training, or pattern discovery. We believe our approach is much more in the line of systems such as XWrap [17], AutoWrapper [11], BYU tool [8, 9] and Island Wrappers [12]. All these systems except AutoWrapper and BYU tool generate wrappers from single web documents with substantial help from the user. Although some of the systems are not fully automatic, it is instructive and pertinent to compare PickUp with these systems as we too generate wrappers for table structures from single web sources but in a fully automatic fashion.

XWrap uses an autonomous heuristic driven boundary discovery method for the recognition of meaningful objects in a source document. It then encodes the information in XML and generates meta-data. The heuristics used can be selected by the user to influence the discovery process of XWrap. But XWrap requires significant amount of user input and guidance and often fails to generate correct wrappers, specially when inappropriate heuristic is selected making it an almost manual and trial-and-error based system. In contrast, PickUp does not require user guidance for non-ambiguous tables[3], and hence can induce wrappers much faster than XWrap.

AutoWrapper on the other hand uses Smith-Waterman algorithm [22] based textual similarity learning for repeated

[2]It is important to note here that most web documents, including all the documents at NCBI site, are still written in HTML for many practical purposes. Hence it is important that algorithms manipulating such pages are capable of handling HTML peculiarities. We also believe that translation of HTML to XML does not completely remove the problems associated with HTML documents and hence is not the solution to this problem.

[3]A table structure is considered ambiguous if there exists more than one candidate tables in a web document. Users may optionally mark the intended table structure in PickUp to disambiguate the identification correct structure.

structure identification. However, unlike PickUp, AutoWrapper cannot identify table structures represented without HTML table tags, nested tables, empty tables, or even single row tables. Finally, the Island Wrapper system requires that users mark an appropriate set of example texts (training set) for it to generate the wrappers. It uses advanced elementary formal system (AEFS) to automatically learn wrappers. But the quality and success of the induction depends entirely upon the ability of the users to appropriately select sufficiently expressive examples so that the system can learn patterns correctly. The strength of Island Wrapper system lies in the fact that it is based on a formal system and the correctness of the induced wrappers can be reasoned and predicted.

Finally, the system proposed by Embley et. al [8, 9] (called the BYU tool) takes a different approach. To be able to construct a wrapper BYU tool require an accurate ontology designed by expert users manually. Once the ontology is supplied, the system can map and consolidate records from multiple sources to the ontological schema. The approach is restrictive in many ways. We discuss several major limitations. First, applications involving sites with wide variations in ontological structures will extract far less information. On the other hand, PickUp will perform better in applications where accurate ontologies are not available or are difficult to construct. Second, this system will not support ad hoc extraction which is the focus of our approach. Third, the approach is not suitable for table structures at higher depths. PickUp is not limited by nesting depth of tables.

3. WRAPPER GENERATION IN THE CONTEXT OF GQL

Data processing in life sciences applications are complicated by two major factors - the distributive nature of the repositories and resources, and specialized tool applications needed to interpret data. As data become abundant, and application tools for interpreting them get increasingly sophisticated, researchers are focusing on new applications that require not just a single data source, or a set of sequences and a set of tools, but are part of a substantial array of complex interacting resources that comprise a high level application delivering useful information. For example, consider an application in protein function prediction. A step in protein function prediction may involve the identification of distant homology of a given query probe as described below.

1. For a given probe, retrieve similar sequences (using PatternHunter [19], Canada) from a sequence database (PDB [20], USA).
2. Filter unwanted sequences (use a popular homology filter [e.g., Blixem [2], UK).
3. If more than one sequences survived the filtering, do a multiple sequence alignment (using ClustalW [6], UK). Otherwise, perform a pairwise alignment (using Blast2 [1], USA).
4. Perform a profile generation based on the alignment (use profile generators, e.g., PROFILE [21], UK).
5. Use the profile as a probe, and retrieve similar sequences from the sequence database.
6. If no new matches, then the profile and the sequences in the previous pass are the solution. Otherwise, perform multiple sequence alignment over the profile and the new matches.
7. Go back to profile generation in step 4.

The simple and partial application above adequately highlights the need for a true workflow enabled query processor that is capable of executing the steps involved, using the resources spread over multiple online sites across several continents. It is our contention that it is perhaps unwise to develop a customized application for this process if the application is unstable, short lived, has a potential to change, or if the sites involved change often without notice. But the traditional approach to this application has been either by developing customized applications that implement the workflow, maintaining it and coping with the changes in sites, or by downloading all the required resources and maintaining them locally, and thereby dealing with loss of currency issues and view maintenance related complexities.

3.1 Declarative Querying of Life Sciences Databases

At Wayne State University, we are developing an SQL-like declarative query language, called the *genomic query language* or GQL [13, 15], that is aware of the distributive nature of life sciences resources – data and applications alike. In short, the envisioned language will support ad hoc, declarative and workflow-like querying of "arbitrary" resources. To be able to support such ad hoc declarative querying, we believe two fundamental operations are needed. (i) Since resources in this domain are inherently distributed and heterogeneous, we need a method to extract the target data from multiple sources and combine them in a unified format, and then, as a next step (ii) submit the set of gathered data to an application and extract the desired information from the application output. The extraction and combination of data require semantic reconciliation in several axes – at the structural level of data and at the logical meaning of data. We are using an approach based on ontology generation and merging using *OntoBuilder* [10, 18] for semantic reconciliation. The discussion on semantic reconciliation is not relevant for this paper and is therefore, omitted. We focus only on the extraction of target data, which plays a vital role in both the cases – data gathering and tool application.

3.2 Remote User Defined Functions

Resources published on the web can be classified into two general categories - plain documents, and documents with forms. Plain documents usually contain data, structured or otherwise, and forms usually capture abstraction of databases (often known as the hidden web), and functionalities of applications as black boxes. Thus, for any form, we are only knowledgeable about its input-output behavior. So, for the purpose of modeling, we can treat them uniformly and think that both types return a defined type of data given a defined set of input data. In this sense, we can consider forms as the recently introduced concept of remote user defined functions [3, 4] in extended SQL, which in turn, is a concept similar to wrappers.

Even when the heterogeneity and disparities of component data repositories are resolved, effective query processing requires customized communication and querying of participating sites, response gathering, reverse transformation of the responses to global view, and often secondary processing at the originating site. We have developed a semi-automatic wrapper generator system, called PickUp [4], that can generate wrappers for web based (plain) documents or forms.

These wrappers use our own HTML query language called hyper text query language or HTQL [4], and an SQL extension called the remote user defined function (RUDF) that facilitates direct communication between an SQL engine and the external web sources. This SQL extension basically allows treating web resources as relational tables and is defined using syntax similar to the one shown below. In the extended SQL expression below, a function blast is being defined that accepts an input parameter called query, and returns an output called *req_id*. The function is at the URL `http://www.ncbi.nlm.nih.gov/blast/Blast.cgi`, and the result req_id can be extracted by executing the HTQL expression <*form*> . <*input*> on the returned page.

define function *blast*
href *http://www.ncbi.nlm.nih.gov/blast/Blast.cgi*
parameters *query* varchar(1000)
results *req_id* varchar(40)
htql value: <*form*> . <*input*>;

This function in NCBI database returns BLAST search results when a DNA/Protein sequence is supplied as the query constant (condition). RUDF is actually designed to define functions for form type documents, but it is possible to omit the parameter clause and make it work like a function for extracting data from a plain document. Once defined, the function *blast* can be used anywhere in an SQL statement where a built-in function can be used. Our objective in this paper is to show that expressions such as <*form*> . <*input*> needed in the htql value: clause in the above RUDF expression can be generated in a fully automatic manner. This automation is essential if we insist on ad hoc querying of arbitrary online resources. In our PickUp system, we essentially do just that – generate the RUDF expressions fully automatically for ad hoc integration of remote resources at query run time.

4. HIERARCHICAL REPEATED STRUCTURE RECOGNITION

In this section we introduce the so called hierarchical repeated structure recognition (HRSR) method for automated table structure identification. The HRSR method is primarily based on the observation that records in table structures in web documents share certain structural regularity. Intuitively, HRSR technique involves the identification of cells in a conceptual table, a reconstruction of rows or records from fragmented cells, and generation of a table model from a set of similar records. Cell identification is facilitated by the path expressions using the hyper text query language (HTQL). These path expressions are used also to model the table structures and the wrappers. In the following sections we will discuss each of these concepts in some details. However, at this point we refer the readers to [4] for an introduction to HTQL and its associated tag-tree data model[4] on which our subsequent discussions are based.

[4]In the context of the discussions presented in section 4.1, it is important to remark that the fluidity of HTML documents and its tag structures do not break down our data model. Hence it is not necessary, as we tacitly assume in this paper, that HTML documents are somewhat regular – have matching start and end tags.

4.1 Structural Relationships of HTML Elements in Tag-Tree Data Model

An HTML document in our tag-tree data model is a sequence of items. Items are of two types – tag items (HTML tags) and text items. As usual, tags items are of two types – *start tags* (or s-tags) and *end tags* (or e-tags). For any two s-tags s_1 (short form of $< s_1 >$) and s_2 in any document d, s_2 is said to be *reachable* from s_1, written $s_1 \rightarrow s_2$, if s_2 follows s_1, and the corresponding e-tag of s_1 follows s_2 in d.

Thus, by definition, the s-tags in d naturally induces a partial order on all the s-tags of d – i.e., for any three s-tags a, b, c, $a \rightarrow a$, $a \rightarrow b \wedge b \rightarrow c \Rightarrow a \rightarrow c$, and $a \rightarrow b \wedge b \rightarrow a \Rightarrow a = b$ hold. Given two s-tags a and c, $path(a, c)$ holds if $a \rightarrow c$ holds, and $path(a, c) = a.b_1 \ldots b_n.c$ when $a \rightarrow b_1, b_1 \rightarrow b_2, \ldots, b_{n-1} \rightarrow b_n, b_n \rightarrow c$.

For any two distinct s-tags s_1 and s_2 such that $s_1 \rightarrow s_2$ holds, s_1 induces a natural indexing on all s such that $s_1 \rightarrow s$, and $s_2 = s$. For all such s, $s_1.sk$ represents the kth s after s_1 such that $s_1 \rightarrow s$ holds. Notice that $a.c$ always holds when $a.b.c$ holds. For example, when $a.b.c.d.c.e$ holds, $a.c1$ means $a.b.c$ and $a.c2$ means $a.b.c.d.c$ while $a.c$ or $a.c0$ means all c such that $a \rightarrow c$ holds.

Furthermore, it is assumed that every HTML document begins with a special *null* tag, or n-tag. So, for any s-tag s, $null \rightarrow s$ always holds. Since the n-tag is a special tag, for any s-tag s, $path(s) \equiv path(null, s)$. As such, $a.b.c.d.c.e \equiv a1.b1.c1.d1.c1.e1$.

A path p is called complete if all its tags are indexed, i.e., $a1.b1.c1.d1.c1.e1$ is a complete path. The tags in a complete path with the indices are called qualified tags. The trailing qualified tag of a complete path is called the target tag. For any complete path p, $seq(p)$ denotes the sequence of tags in p. i.e., for an arbitrary $p = a1.b.c1.d.c1.e1$, $seq(p) = a.b.c.d.c.e$.

Let $p_1, \ldots, p_n$ be a set C of complete paths with identical trailing tags (may differ in indices). Let 2^C be all possible subsets of C such that for each $c \in 2^C$, and for all $p_1, p_2 \in c$, $seq(p_1) = seq(p_2)$ and all $p \in c$ share a common suffix σ_c of qualified tags. Each such c is called a family of *structurally similar* paths.

For a given pair of sets $c_1, c_2 \in 2^C$, c_1 is preferred over c_2 if the length of every paths in c_1 is longer than the length of the paths in c_2, and the length of σ_{c_1} is longer than length of σ_{c_2}. A set $c \in 2^C$ is *most preferred* if there exists no c' such that c' is preferred over c.

However, for any $c \in 2^C$, c is called a *related* family of paths if all $p \in c$ share a common prefix π_c of qualified tags. For a given pair of sets of related family paths $c_1, c_2 \in 2^C$, c_1 is preferred over c_2 if the length of π_{c_1} is longer than length of π_{c_2}. A set $c \in 2^C$ of related family paths is *most preferred* if there exists no c' such that c' is preferred over c.

The problem of automatic table structure identification is thus stated as the identification of the largest set S of most preferred structurally similar paths and the largest set R of most preferred related family paths in document d at item offset[5] δ such that the length of the common prefix of paths in $R \cup S$ is maximized.

[5]The offset δ is used to discover target table structure using the tag preceding immediately before the offset. Thus using a zero offset is tantamount to the discovery of structure appearing anywhere in the document (from the start to end of document). We will take up the discussion on the importance of δ again in section 4.2.1.

4.2 Discovery of Regular Structures

A major assumption we exploit in our system is that record structures in a document share a structural pattern, and thus all paths to its components (columns) share a common prefix. To discover such common substructures, we exploit the idea of repeated pattern mining in gene sequences [14]. The difference is that we now mine HTML tag sequence instead of gene sequence. The entire process is explained in the next few sections.

4.2.1 Target Structure Recognition

In a given document d, it is possible that it includes several tables. Every such table structure (not necessarily represented using HTML table tag) will have repeated tag structure or sub-trees (captured in the form of paths). With the help of a variant of SeqMiner tool [14], we discover all repeated patterns and their repeat count in the tag trees of d. Then we rank all the repeated patterns using a simple function R. The function R returns an integer value for every repeated pattern given the length of the repeat sequence and the frequency of repeat. Ranking of repeats (and thus identification of record structure) is somewhat tricky since unintended identification is possible. For this paper we assume that ranking of a pattern must be high if it has high frequency (many records) and long pattern length (many attributes or columns). So we use the function $R(n, m) = n*m$ where n is the length of the repeated pattern and m is the frequency of repeat. It is possible that several patterns will be assigned the same ranking using this formula. So, we choose the first pattern as a candidate for the record structure since each one of them are equally valid.

In situations where a wrong table structure becomes the candidate, PickUp allows marking a target table on the HTML document to disambiguate the identification. This marking of an element, a row, or a table virtually marks the first tag t that precedes the marked element in d. This t is now used as a candidate for target repeated pattern identification where t is the leading tag. We then use the techniques described in the following section to generate path expressions for t and move on to generate a record structure involving t. If t falls within the boundary of the target table structure, our method is guaranteed to find the intended structure. However, it is easy to notice that in an unmarked document, the special *null* tag is assumed to be the leading tag of a candidate repeat pattern outlined above. It implies that marking is well defined and robust, and its use or omission does not break down our procedure.

4.2.2 HTQL Path Expression Generation

Once the repeated tag pattern determined (or the leading tag of a candidate pattern is known through marking), we need to generate the HTQL expressions for the purpose of wrapper generation. The leading tag of the repeated pattern is used to generate all possible path expressions in HTQL. Details of HTQL language and a procedure for the efficient generation of *complete path* expressions can be found in [4].

4.2.3 Structural Relationship Recognition

Notice that the repeated pattern identification alone is not a guarantee that a correct table structure will be discovered. So, we use the leading tag t of the repeated pattern to generate all possible HTQL path expressions in which t appears as the trailing tag. By doing so, we include more candidates for the table structure by inflating the set of path expressions. However, the table structure identified by the repeat pattern is still included in the set. The set of paths obtained at this stage is used to compute the set S of most preferred structurally similar paths as explained in section 4.1. And finally, the set S is used to compute the set R of most preferred related family paths. This is accomplished by iteratively generating the path expressions as explained in section 4.2.2 for each of the tags in the repeated pattern, and maximizing the most preferred relationship with respect to $S \cup R$. The combination of S and R in essence captures the most significant table structure in d.

4.2.4 Model Generation and Validation

The most preferred sets S and R do not actually account for missing columns in rows, or missing columns in tables specially when rows are spread over a document space. This is a consequence of the assumption that the table structures need not be represented using HTML table tags. We essentially allow any regularity to be identified as a table structure. This means that two candidate rows may differ with respect to a column. So, we generate a model of the table by conservatively generalizing a row to fit an intended table structure. This is achieved by adding a column to a row only if that column's repeat frequency is more than half of the row frequency. We also accept the type of a column as the type of the majority of the row types in that column. Finally, we generate a composite HTQL expression to generate the wrapper. Once the wrapper is generated, it is validated by recreating each of the elements in the table structure in d by the unit expressions in the wrapper. If the test is positive for all the elements, the validation is considered successful.

4.3 Examples

We now present two examples to demonstrate the capabilities and the effectiveness of the PickUp system. The figure 2 shows a CGAP page containing a table, and figure 3 shows the table generated by a wrapper induced by PickUp from the page in figure 2. As shown in figure 2, the table structure is represented using table tag and is somewhat obvious, and PickUp had no trouble correctly wrapping this table.

COLUMN1	COLUMN2	COLUMN3	COLUMN4
2819345	Full Length	BC000292	actin, gamma 1
2820489	Full Length	BC001920	actin, gamma 1
2962953	Full Length	BC007442	actin, gamma 1
3939552	Full Length	BC009848	actin, gamma 1
3940861	Full Length	BC010999	Similar to actin, gamma 1
4554944	Full Length	BC012050	Similar to actin, gamma 1
4842665	Full Length	BC015695	actin, gamma 1
4855463	Full Length	BC015005	actin, gamma 1
4856294	Full Length	BC015779	actin, gamma 1
4869628	Full Length	BC018774	actin, gamma 1
3533309	Incomplete	BC004223	
3538275	Incomplete	BC017450	
3897065	Incomplete	BC009544	
4053240	Incomplete	BC010417	Similar to actin, beta, cytoplasmic
4109821	Incomplete	BC023548	Similar to actin, beta

Attributes=4, **tuples**=15, **total**=60

Figure 3: Table of figure 2 recreated by the wrapper.

The next example (in figures 4 and 5) demonstrates that PickUp is capable of effectively identifying table structures even when the structure is not represented using HTML table tags. This example goes to show that PickUp is more

versatile than XWrap, Island Wrapper and AutoWrapper in identifying and wrapping table structures.

Figure 4: List of books on humans represented using loose table structures without table tags.

COLUMN1	COLUMN2	COLUMN3	COLUMN4	COLUMN5	COLUMN6
	695 items	Cancer Medicine.	Bast, Robert C.; Kufe, Donald W.; Pollock, Raphael E.; Weichselbaum, Ralph R.; Holland, James F.; Frei, Emil, editors.	Canada: BC Decker Inc; c2000	BC Decker Inc;
	577 items	Medical Microbiology.	Baron, Samuel, editor	Galveston: University of Texes Medical Branch; c1996.	University of Texes Medical Branch
	218 items	Molecular Cell Biology.	Lodish, Harvey; Berk, Arnold; Zipursky, S. Lawrence; Matsudaira, Paul; Baltimore, David; Darnell, James E.	New York: W H Freeman & Co; c2000.	W H Freeman & Co
	207 items	Molecular Biology of the Cell.	Alberts, Bruce; Bray, Dennis; Lewis, Julian; Raff, Martin; Roberts, Keith; Watson, James	New York and London: Garland Publishing; c1994	Garland Publishing

Figure 5: Faithful recreation of the books table in figure 4 by the wrapper generated by PickUp.

5. COMPOSITE WRAPPERS

Our over arching goal for PickUp is to be able to generate a wrapper that can iterate over the rows of an HTML table and navigate to the next page pointed to by one of the cells so that users do not need to simulate the iteration manually. This can be achieved in two principal ways. Hyperlinks can be identified automatically if all the links are of interest, or by letting the users mark a column on the generated wrapper output that contains a hyper link in order to avoid the generation of uninteresting wrappers. Once identified, the same technology as applied in the current page may be applied to identify tables in the subsequent pages. The result would be something similar to a nested table.

However, in the context of web documents, nesting can be indefinite and hence termination remain a serious issue. In our implementation, we require that the users specify a depth of nesting for the wrapper. For any depth n, PickUp will only extract data corresponding to links that are nested n levels deep. We also allow users to choose the set of interesting hyperlinks for extraction. The default is the set of all hyperlinks in the document.

The figures 6 and 7 show the idea behind the composite wrapper generation for depth $n = 1$. Notice that the links in figure 6 are chased and the table in the page pointed to by the link is extracted and inserted in the cells as shown in figure 7.

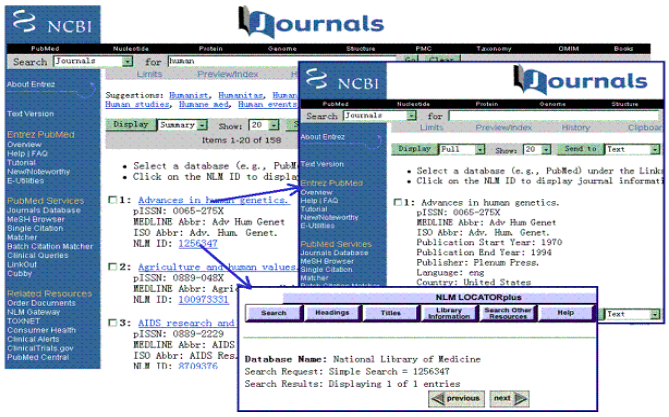

Figure 6: NCBI journals database page with links to subsequent pages.

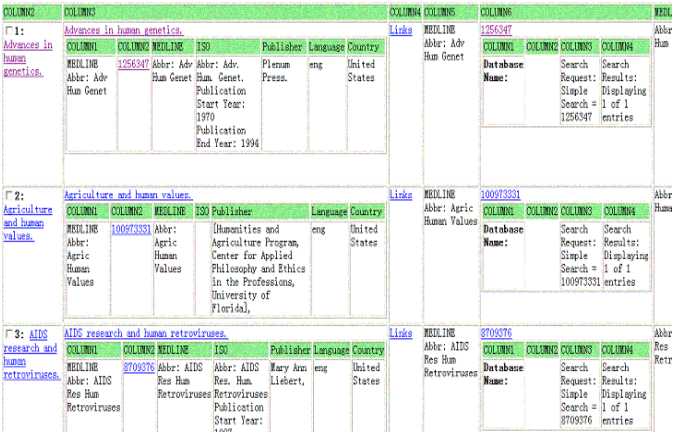

Figure 7: Nested table of depth 1 extracted from the page in figure 6 using a wrapper generated by PickUp.

6. SUMMARY

In this paper we explored a new technique for automatic wrapper generation for table structured data in semi-structured documents. The approach relies on several key ingredients – a formal model of path expression based characterization of tables, repeated pattern discovery and reconstruction of table structures from fragmented path expressions using a preference relationship. We believe we have intuitively demonstrated through experiments and examples that our method is more effective and robust than the leading systems that generate wrappers for table structures in automatic ways.

Due to space limitations, we could not discuss many technical details. An extended version of this paper [5] will contain all the algorithms and a detailed description of our

method, as well as an objective comparative discussion of our tool with available contemporary approaches. Another important issue, one which is intimately related to wrapper technology, we could not address for want of space is maintenance of generated wrappers. We believe that the wrappers generated by PickUp are more suitable for incremental maintenance. This is not true for most other systems. We believe that it is possible to detect changes in target documents and tweak the existing wrapper to adapt to the new document. This process may be facilitated by maintaining the set of candidate wrappers we generate during the wrapper induction phase. However, developing a method for incremental wrapper maintenance remains part of our future research.

7. REFERENCES

[1] Blast2. http://www.ncbi.nlm.nih.gov/blast/bl2seq/bl2.html.

[2] Blixem. http://www.hgmp.mrc.ac.uk/registered/option/blixem.html.

[3] L. Chen and H. M. Jamil. Supporting remote user defined functions in heterogeneous biological databases. In *Proceedings of the 2nd IEEE Bioinformatics and Bioengineering Symposium*, pages 144–152, Bethesda, Maryland, Nov 2001. IEEE Press.

[4] L. Chen and H. M. Jamil. On using remote user defined functions as wrappers for biological database interoperability. *International Journal on Cooperative Information Systems*, March 2003. Special Issue on Biological Databases.

[5] L. Chen and H. M. Jamil. A table structure extraction operator for relational databases from semi-structured data. Technical report, Wayne State University, June 2004. In Preparation.

[6] ClustalW. http://www.ebi.ac.uk/clustalw/.

[7] L. Eikvil. Information extraction from the world wide web: a survey. In *Technical Report 945*. Norwegian Computing Center, 1999.

[8] D. Embley, D. Campbell, Y. Jiang, S. Liddle, D. Lonsdale, Y.-K. Ng, and R. Smith. Conceptual-model-based data extraction from multiple-record web pages. *Journal of Data & Knowledge Engineering*, 31(3):227–251, November 1999.

[9] D. W. Embley, Y. S. Jiang, and Y.-K. Ng. Record-boundary discovery in web documents. In *Proceedings of the ACM SIGMOD Conference*, pages 467–478, Philadelphia, PA, 1999.

[10] A. Gal, G. Modica, and H. Jamil. Ontobuilder: Fully automatic extraction and consolidation of ontologies from web sources. In *Proceedings of the 20th IEEE International Conference on Data Engineering (ICDE 2004)*, Boston, USA, 2004.

[11] X. Gao and L. Sterling. Autowrapper: automatic wrapper generation for multiple online services. In *Asia Pacic Web Conference '99, Hong Kong*, 1999.

[12] G. Grieser, K. P. Jantke, S. Lange, and B. Thomas. A unifying approach to html wrapper representation and learning. *Discovery Science,DS 2000, Kyoto, Japan, December 4-6, 2000*, 1967:50–64, 2000.

[13] Z. Guan and H. M. Jamil. Streamlining biological data analysis using bioflow. In *Proceedings of the 3nd IEEE Bioinformatics and Bioengineering Symposium (BIBE 2003)*, pages 258–262, Washington, DC, Mar 2003. IEEE Press.

[14] J. Han, H. M. Jamil, Y. Lu, L. Chen, Y. Liao, and J. Pei. DNA-Miner: A system prototype for mining DNA sequences. In *Proceedings of the ACM SIGMOD Conference*, Santa Barbara, CA, 2001.

[15] H. M. Jamil, G. Modica, and M. Teran. Towards a visual query interface for phylogenetic databases. In *Proceedings of the 10th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 57–64, Atlanta, Georgia, Nov 2001. ACM Press.

[16] A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. A brief survey of web data extraction tools. *SIGMOD Record*, 31(1):84–93, 2002.

[17] L. Liu, C. Pu, D. Buttler, W. Han, H. Paques, and W. Tang. Aqr-toolkit: An adaptive query routing middleware for distributed data intensive systems. *SIGMOD Record*, 29(2):597, 2000.

[18] G. Modica, A. Gal, and H. Jamil. The use of machine-generated ontologies in dynamic information seeking. In C. Batini, F. Giunchiglia, P. Giorgini, and M. Mecella, editors, *Cooperative Information Systems, 9th International Conference, CoopIS 2001, Trento, Italy, September 5-7, 2001, Proceedings*, volume 2172 of *Lecture Notes in Computer Science*, pages 433–448. Springer, 2001.

[19] PatternHunetr. http://www.bioinformaticssolutions.com/software/ph/doc/index.php.

[20] PDB. http://www.rcsb.org/pdb/.

[21] PROFILE. http://bmbsgi11.leeds.ac.uk/bmb5dp/home.html.

[22] T. Smith and M. Waterman. Identification of common molecular subsequences. *Molecular Biology*, 147:195–197, 1981.