QuASM: A System for Question Answering Using Semi-Structured Data

David Pinto, Michael Branstein, Ryan Coleman, W. Bruce Croft, Matthew King,
Wei Li and Xing Wei
Computer Science Department
University of Massachusetts
Amherst, MA 01003
{pinto, michaelb, ryanc, croft, matthewk, weili, xwei}@cs.umass.edu

ABSTRACT

This paper describes a system for question answering using semi-structured metadata, QuASM (pronounced “chasm”). Question answering systems aim to improve search performance by providing users with specific answers, rather than having users scan retrieved documents for these answers. Our goal is to answer factual questions by exploiting the structure inherent in documents found on the World Wide Web (WWW). Based on this structure, documents are indexed into smaller units and associated with metadata. Transforming table cells into smaller units associated with metadata is an important part of this task. In addition, we report on work to improve question classification using language models. The domain used to develop this system is documents retrieved from a crawl of www.fedstats.gov.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – *information filtering, search process.*

General Terms: Algorithms, Measurement, Performance, Design, Experimentation.

Keywords: Question answering, semi-structured, tables, question classification, content documents, metadata, language model

1. INTRODUCTION

Traditional information retrieval (IR) systems use a query to return a set of documents that may be relevant to the user’s information need. In the case of a user looking for a specific, short answer to a question, these documents must be scanned to achieve this goal. Answers may be buried deep in tables, or far down in the document, making this process tedious.

Question answering (QA) systems augment IR systems by taking over the scanning task. A separate search and scoring algorithm is executed to locate answers in the retrieved documents (Voorhees[10]). Most QA systems have dealt with TREC data, where the determination of an answer is based in part on the syntactic structure of the language in the documents. QA performed on web pages could potentially exploit the structure inherent in the HTML markup (semi-structured data) – especially for tabular data.

This paper describes a system called **Qu**estion **A**nswering using **S**emi-structured **M**etadata (QuASM). QuASM is built on previous work in cross language question answering [5]. Experiments were performed to determine if the semi-structured nature of web documents could improve the performance of these systems.

One way of taking advantage of the structure is to determine if a document contains suitable data for answering questions. Many of the documents obtained from the WWW do not contain useful content. Pages may, for example, simply provide navigational links to other documents. If these pages can be removed, the initial document retrieval phase of a QA system should be improved. Finn, Kushmerick and Smyth [4] developed a way of identifying text sections in documents. This method was adapted for selecting content documents.

Once content documents have been identified, they can be transformed based on their structure. QA systems look for answers in close proximity to query terms [10]. When a table is used to display data, a cell of the table may be at a distance from where the query terms are likely to reside (e.g. column and row headers). Pyreddy and Croft [7] developed heuristics for identifying text tables and their components. As an extension of this work, the desired components of a table can be matched with each cell. The same can be done for more structured tables found in Hypertext Markup Language (HTML) documents.

Like tables, the prose sections of a document can be transformed. In a document on farming, there is no need to search for answers to questions about potatoes in a section about cows. The semi-structured nature of web documents offer clues to where sections begin and end, as well as their subject matter. Using these clues, it is our belief that documents can be broken into smaller units that are topically homogeneous. Our hypothesis is that by indexing smaller parts of documents for IR, the answer search will be more effective.

The QuASM experiments also involved query classification. In order to find answers, a relationship must be established between the question being asked and possible target answers in the document. Questions need to be classified on the basis of the type of answer that is expected. These answer entities must also be recognized in retrieved documents so the answers can be located. Regular expressions have been commonly used to classify

JCDL’02, July 13-17, 2002, Portland, Oregon, USA.
Copyright 2002 ACM 1-58113-513-0/02/0007…$5.00.

questions, but we found them lacking as the number of classes of questions were expanded and as we attempted to classify real world questions. In IR, language models have been used to estimate the probability that a document generates a query (Ponte and Croft [6], Song and Croft [9]). Our idea for improving question classification is to use language models of question classes to estimate the probability that the answer class generated the question.

The rest of this paper looks in detail at the components of the QuASM system. Section 2 presents an overview of the components. Section 3 discusses content selection in detail. Section 4 explains table transformation, with 4.1 concentrating on text tables and section 4.2 dealing with HTML tables. Section 5 presents query classification and entity identification. Section 6 evaluates the overall system and Section 7 looks to future work.

2. OVERVIEW

Figure 1 is a diagram of the acquisition and data processing modules of the QuASM system. The components are discussed below and in more detail in later sections.

2.1 Database Acquisition

Database acquisition consists of two tasks. The first is obtaining pages by means of a web crawl. The second is extracting pages that contain content (and thereby answers) from these documents.

The goal of the crawl was to extract pages related to the www.fedstats.gov site, as the questions we desired to answer would be of a statistical nature. A tiered search was used to accomplish this. All pages in the fedstats.gov domain were collected first. As pages were collected, links to pages outside that domain were also collected. Once a domain was exhausted, the program could move on to the next tier and repeat the process. Scripts were developed to allow the stopping and restarting of the crawl at each level.

Other heuristics were added to ignore error pages, ignore binary files, (including .pdf files), ignore popular web domains, and compensate for duplicate pages that appear to be unique.

Not all documents retrieved from the web crawl will contain content appropriate for question answering. Documents are screened to separate documents with answer content from those that are navigational or interactive (query pages). Results are discussed in section 3.

2.2 Document Processing

Documents are processed to break them up into smaller information units. Two of the processing methods transform table cells into short documents consisting of cell data and metadata. A third process breaks text documents into smaller sections for faster processing of questions.

2.3 Text Tables

Tables present an interesting problem for question answering. QA systems look for potential answer entities in close relation to query terms. However, in a data table, the answer may be rows and columns away from the text that could contain the query terms; the row names, column headers, titles and captions.

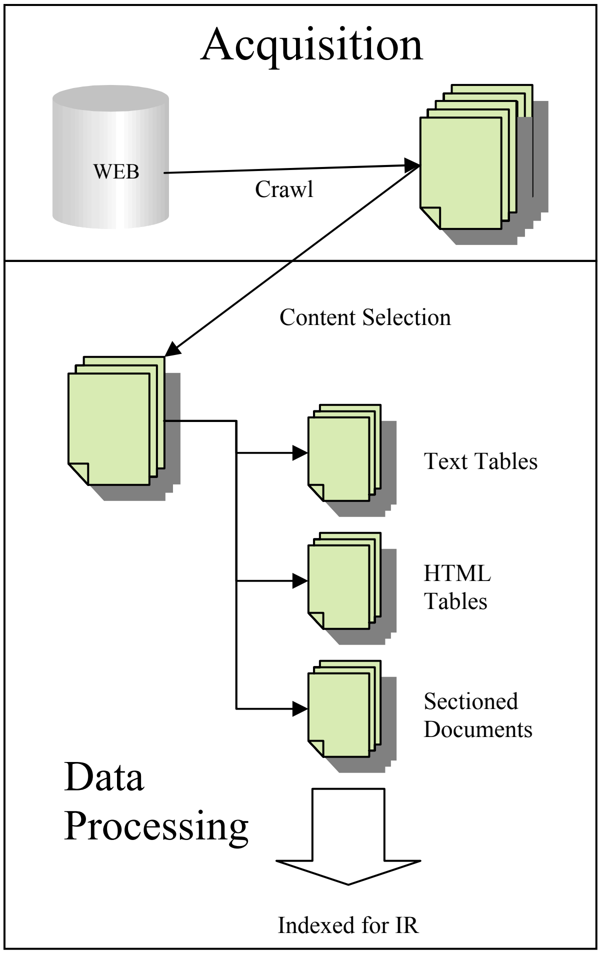


Figure 1. Data Acquisition and Processing

Text tables in particular make this difficult, since the layout of these tables is as varied as their composers. Cell spacing is random. Some use characters (-,|,!,+) to delineate rows and columns. Columns may have multiple headers, or one header may exist over a number of rows. Numerous heuristics have been developed to generally place the correct designators with the proper cell data. A full explanation of the heuristics and experimental results can be found in section 4.1.

2.4 HTML Tables

Unlike text tables, HTML tables offer a clearer structure. The beginning and end of these tables are marked with HTML tags, as are the column headers and individual cells. However, there is no standard that forces an HTML author to adhere to these.

An efficient algorithm was designed to make one pass through the data to join titles, row and column headers with cell data. The first row of the table is considered the header row and the first column of the table contains the row names. For any particular cell not in the first row or first column, the corresponding table title, row name and column header are written out with the cell data as a sentence. A more thorough discussion is found in section 4.2

2.5 Document Sectioning

The purpose of document sectioning is two-fold; to produce small snippets of documents on a single subject, and to find metadata for those snippets.

HTML tags provide clues about the organization of a document and its subject matter. Words tagged as bold or italics, or that appear in different levels of headings convey information about the section. Paragraph and body tags provide a good indication of the end of a logical section.

To find sections, the program creates and performs a series of operations on a *tag depth array*. The tag depth array contains two pieces of information about each line in the file; the text of the line and the depth of the line inside tags. Each time an important tag (BIG, BOLD) is encountered, we increase the depth by 1. When we find the end of that tag, the depth is decreased by 1, but never below zero. Certain other tags are reset tags (such as the start of a paragraph), and always reset the depth to zero. Each line of text is assigned the current depth.

Once each line is tagged with a depth value, the lines are arranged into sections. First, adjacent sections with the same depth are combined. Then consecutive sections are combined until a depth of 0 is reached, at which point a new section starts. Lines with depth greater than zero are saved as metadata.

Other heuristics are applied for special cases and better performance. An attempt is made to keep sections between 100 and 500 characters. For documents with no HTML markup, other clues must be used to section a document, such as capitalization and numbering. Once processing is complete, each section is written out as a separate document.

2.6 Answer Retrieval

Figure 2 represents the answer retrieval module of the QuASM system. Documents are indexed using the Inquery search engine [3]. The question classification and entity identifier are discussed below and in more detail in later sections.

2.7 Question Classification

In order to answer questions correctly, the query must be classified so an answer of the same type can be located in the retrieved documents. These question classes can be general (**NUMBER**, **LOCATION**) or more specific (**AREA**, **MASS**).

A number of techniques have been employed to identify the answer class of a question. Entity tagging, part of speech tagging, regular expressions and language models all combine to produce a class for a question. A more thorough discussion and evaluation of this work are in section 5.1.

Once passages have been retrieved, possible answer entities are tagged using BBN's Identifinder^{TM} (described in [2]) and a program developed as part of this project to find entities corresponding to answer classes not covered by Identifinder^{TM}. A more thorough discussion of this work is presented in section 5.2.

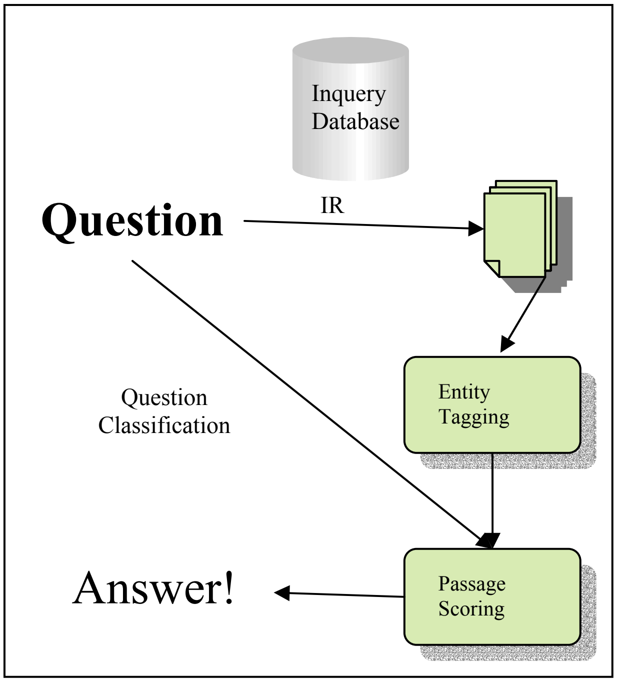


Figure 2. Answer Retrieval

3. CONTENT SELECTION EXPERIMENT

Rather than indexing every web page retrieved from the crawl, a filtering program was put in place to determine which documents were more likely to contain content. Content in this instance refers to prose and tables that convey information useful in answering questions, as opposed to navigational aids.

Our selection algorithm is based on work by Finn, Kushmerick and Smyth, [4] who explored text extraction from HTML files. A document is represented as a binary vector. Tokens representing HTML tags are given the weight one, all others weight zero. We modified this algorithm by setting certain HTML tags to have weight 0, since they were likely to occur in content sections (font changes, headings), or indicate content (table tags). From this vector, a document slope curve (DSC) is generated. The entries in the DSC array correspond to the total of the binary vector up to and including that token. Long, low sloping regions of this graph represent content (text without tags). If these regions are large enough the document is classified as containing content.

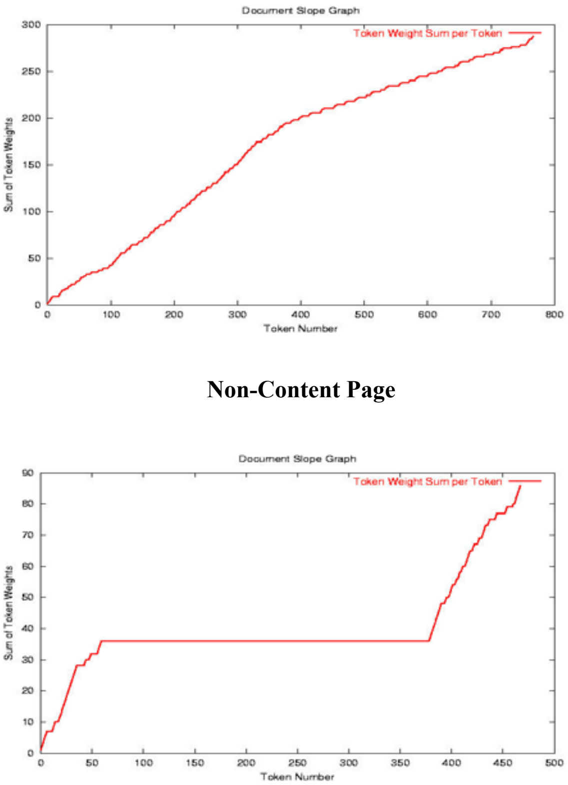
Figure 3 contains two examples of document slope curves. The graph labeled non-content page is from a document that contains a list of hyperlinks. The HTML tags in this document occur regularly, so the graph shows a steady rise. The graph labeled content page is an article from the EPA web site. The beginning and end of the document contain hyperlinks and HTML formatting tags, while the middle of the article is text. The curve has high slope at the beginning and end, but is flat in the middle.

The algorithm for determining content is as follows:

- Create the binary vector (BV).
- From this binary vector, create a DSC.
- The length of the document (in tokens) is used to determine a window size. This window will be used to

determine the average slope of each section. The minimum window size is 8 (for documents less than 200 tokens) and the maximum size is 50 (for documents greater than or equal to 5000 tokens.

- Record the average slope of the document
- Starting at the beginning of the DSC, and moving the window half the window length at a time, look for three consecutive sections that have an average slope less than 50% of the average slope of the document. A low slope area has now been located. Classify each of these areas as low slope until three consecutive sections where the slope is greater than 50% of the average document slope have been found.
- Once the low slope sections have been identified, determine the number of tokens in those sections. If this number is less than 10% of the tokens in the document, reject the document as having no content.
- If the document passes the above test, check the average slope of the low slope sections. If this average slope is less than 50% of the average slope of the document, accept it as having content.



Non-Content Page

Content Page

Figure 3. Document Slope Curves

After tuning the parameters on training data, documents were randomly extracted from the web crawl and judged by hand to be content or non-content. Then these documents were classified by the algorithm and compared to the judgments. Table 1 shows the result of this experiment.

Table 1. Experimental Results, Content Selection

Total Documents	188
Judged As Content	87
Identified as Content	109
Correctly Identified as Content	75
Recall	.862
Precision	.688

4. DOCUMENT PROCESSING

The goal in processing documents was to break the pages into multiple small information units with the following properties:

- The small units were about a single fact or related ideas.
- If needed, the data was transformed to associate potential query terms with the data.

Three separate filters were applied to the documents in succession. Text tables were transformed and removed, HTML tables were transformed and removed, and the remaining text was processed into sections. Experiments with text and HTML tables are discussed here.

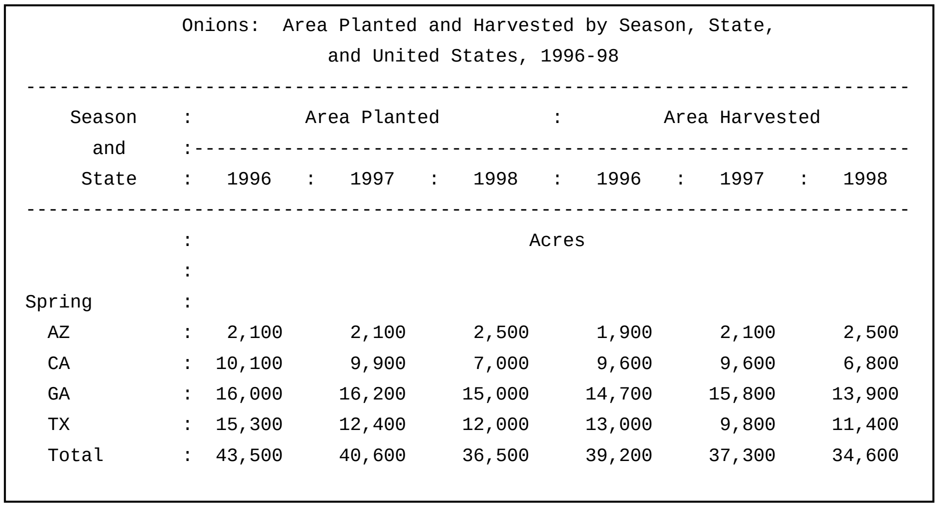
4.1 Text Table Experiments

There is a large amount of content information within our web crawl stored as non-HTML tables (text tables). Our goal was to build on previous work to associate the cells of these tables with metadata. The hypothesis is that transforming the tables in this way will lead to better answers for questions about the table data.

4.1.1 Table Identification

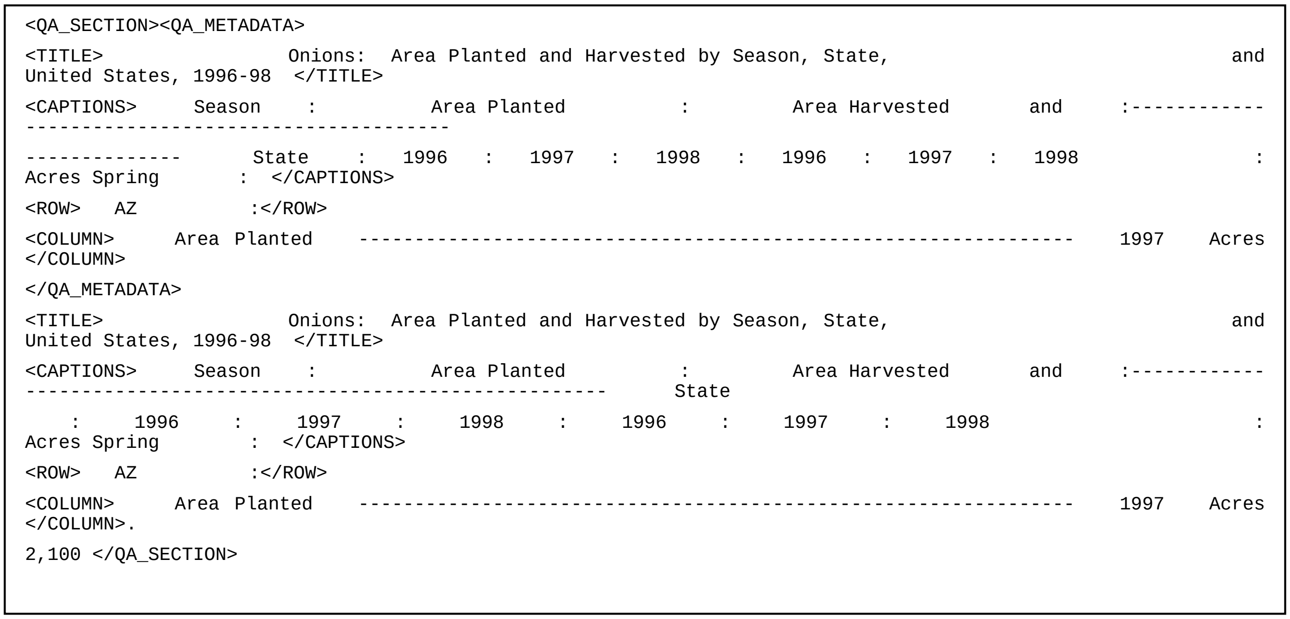
The first step in extracting table data is to locate the tables in the file. An example of a text table can be seen in Figure 4a. We built upon the TINTIN system by Pyreddy and Croft [7]. A Character Alignment Graph (CAG) is created to look for white space alignment in blocks of contiguous lines of texts. A number of heuristics are applied to this CAG:

- A row with more than two gaps may be a table row. Gaps are large areas of white space in a row, and may indicate column structure.
- A row with more than four continuous white spaces may be a table row.
- The density of table rows (number of rows marked as belonging to a table) indicate the presence of a table.
- Simple rows (with less than 3 cells) can indicate the beginning or ending of a table.
- Rows at the beginning of the table are likely to be less regular than the rest of the table.
- The number of simple rows in the table should be small.
- Non-table rows and blank rows are a sign of the end of a table.
- A number of consecutive signs of the end of a table will indicate the end of a table.



```
              Onions:  Area Planted and Harvested by Season, State,
                          and United States, 1996-98
-------------------------------------------------------------------------------
    Season    :          Area Planted          :         Area Harvested
      and     :----------------------------------------------------------------
     State    :   1996   :   1997   :   1998   :   1996   :   1997   :   1998
-------------------------------------------------------------------------------
              :                            Acres
              :
Spring        :
  AZ          :   2,100      2,100      2,500      1,900      2,100      2,500
  CA          :  10,100      9,900      7,000      9,600      9,600      6,800
  GA          :  16,000     16,200     15,000     14,700     15,800     13,900
  TX          :  15,300     12,400     12,000     13,000      9,800     11,400
  Total       :  43,500     40,600     36,500     39,200     37,300     34,600
```

Original Table (a)



```
<QA_SECTION><QA_METADATA>
<TITLE>                Onions:  Area Planted and Harvested by Season, State,                              and
United States, 1996-98  </TITLE>
<CAPTIONS>     Season    :          Area Planted          :         Area Harvested       and     :------------
-------------------------------------
--------------      State    :   1996   :   1997   :   1998   :   1996   :   1997   :   1998               :
Acres Spring       :  </CAPTIONS>
<ROW>   AZ          :</ROW>
<COLUMN>     Area  Planted    ----------------------------------------------------------------    1997    Acres
</COLUMN>
</QA_METADATA>
<TITLE>                Onions:  Area Planted and Harvested by Season, State,                              and
United States, 1996-98  </TITLE>
<CAPTIONS>     Season    :          Area Planted          :         Area Harvested       and     :------------
----------------------------------------------------     State
    :     1996     :     1997     :     1998     :     1996     :     1997     :     1998                       :
Acres Spring        :  </CAPTIONS>
<ROW>   AZ          :</ROW>
<COLUMN>     Area  Planted    ----------------------------------------------------------------    1997    Acres
</COLUMN>.
2,100 </QA_SECTION>
```

Cell Corresponding to Area Planted, 1997, AZ (b)

Figure 4. Text Table Examples

To test table identification, we selected files at random from the set of content files and identified by hand the number of table lines, and compared this to the output of the program. Totals are from 103 files containing 109 tables (Table 2).

Table 2. Experimental Results, Table Identification

Total number of lines	91578	--
Total number of table and caption lines	10944	--
Total number of lines extracted by table extractor	10786	--
Total number of table and caption lines missed	1075	9.8%
Total number of extraneous lines extracted	917	8.4%

While this result is not as good as TINTIN[7], it should be noted that TINTIN was tested on tables from Wall Street Journal news stories only, while the tables in this collection came from a more heterogeneous source; the Web.

4.1.2 Header Identification

The second step in extracting table data is to match data cells with their appropriate headers. This was new research with regard to table extraction. Figure 4b contains an example of a processed table cell, with the header information between the **COLUMN** tags.

Column headers are difficult to detect; they can span multiple lines. They can be split from the data by lines containing other information. They can have sub-headers and super-headers.

Whereas the TINTIN system was only interested in marking rows as tables or captions, our system extended this to find rows that contain header information. A number of heuristics were used to accomplish this:

- Simple rows at the beginning of a table are probably titles.
- Other rows at the beginning of a table are probably caption rows.
- Simple rows in the middle of a table are likely caption rows.
- Rows in a table with a similar number of cells are likely data rows.
- The most common number of cells in a row represents the number of cells in a data row, and rows with that number of cells are marked as such. Authors of text tables often leave cells without data blank. This makes these rows to appear to have fewer cells.
- Data rows with integer numbers in the range 1700 to 2100 are likely caption rows, since these number probably represent years. (These are likely year headers for US government studies, given the age of the country.)
- When data rows are made up of cells containing digits, data rows made up of alphabetic characters are probably caption rows.
- If a cell in a caption row can cover multiple cells in a data row, it is included in the headers for all those columns.
- Caption rows in the middle of a table indicate a new section of the table, and the headers from that row should only apply to the next section.
- Cells in the first column are used as row headers.

The same set of 103 documents was used to test caption identification. The results are in Table 3. Here we show an improvement over TINTIN.

Table 3. Experimental Results, Caption Identification

Actual number of captions	1049	--
Number of captions miss-tagged as table lines	37	3.53%
Actual number of table lines	9895	--
Number of table lines miss-tagged as captions	387	3.9%

One of the most important parts of the above heuristics was determining if a cell in a caption row covered multiple cells in a data row. The most common number of cells in a line is calculated by creating a histogram of the cells in each line and picking the entry with the highest value. For a line with this number of cells, the character position where each cell begins and ends is recorded. Then for a caption line with fewer cells, the beginning and ending character positions of each cell is recorded. Any columns that fall under the span of the caption row cell get this cell as a piece of its metadata.

To test this, we selected 62 tables that had column headers and table rows, and checked to see if we were capturing the column header information. The results are in table 4.

Table 4. Header Capture Experiment

Number of Columns	425
Columns, Header Text Missing	27
Columns, All Header Text Captured	398
Percent, All Header Text Captured	93.6

Of the 398 columns for which we captured all the header information, 327 of those included extraneous text. We believe QuASM will work better with a small amount of extra information than it would by missing relevant metadata. Once all the lines of the table have been marked, titles, caption, row and column data is written to a metadata section, and that same data is repeated with the cell data to be passed on to the scoring algorithm after retrieval.

As a check to see if text table processing would improve QA, the following experiment was performed. Sixteen questions were generated from the data in the table in Figure 4a, all variants on "In 1997, how many acres were planted with onions in the spring in AZ?" changing the state, the year, or substituting harvested for planted. The sixteen questions were run against two different databases. The first database contained the original content files indexed for retrieval with the Inquery search engine (Unprocessed DB). The second database contained the content files with tables extracted and prose sectioned, also indexed for retrieval (QuASM). Five documents were retrieved for each question, and we recorded the mean reciprocal rank[1] (MRR) [10] and the number of questions without an answer found in the top five documents. The results are in Table 5. We found these results encouraging.

Table 5 Experimental Results, Text Table QA

Onion Questions	Unprocessed DB	QuASM
MRR	0	.781
# Not Found	16	0

4.2 HTML Tables

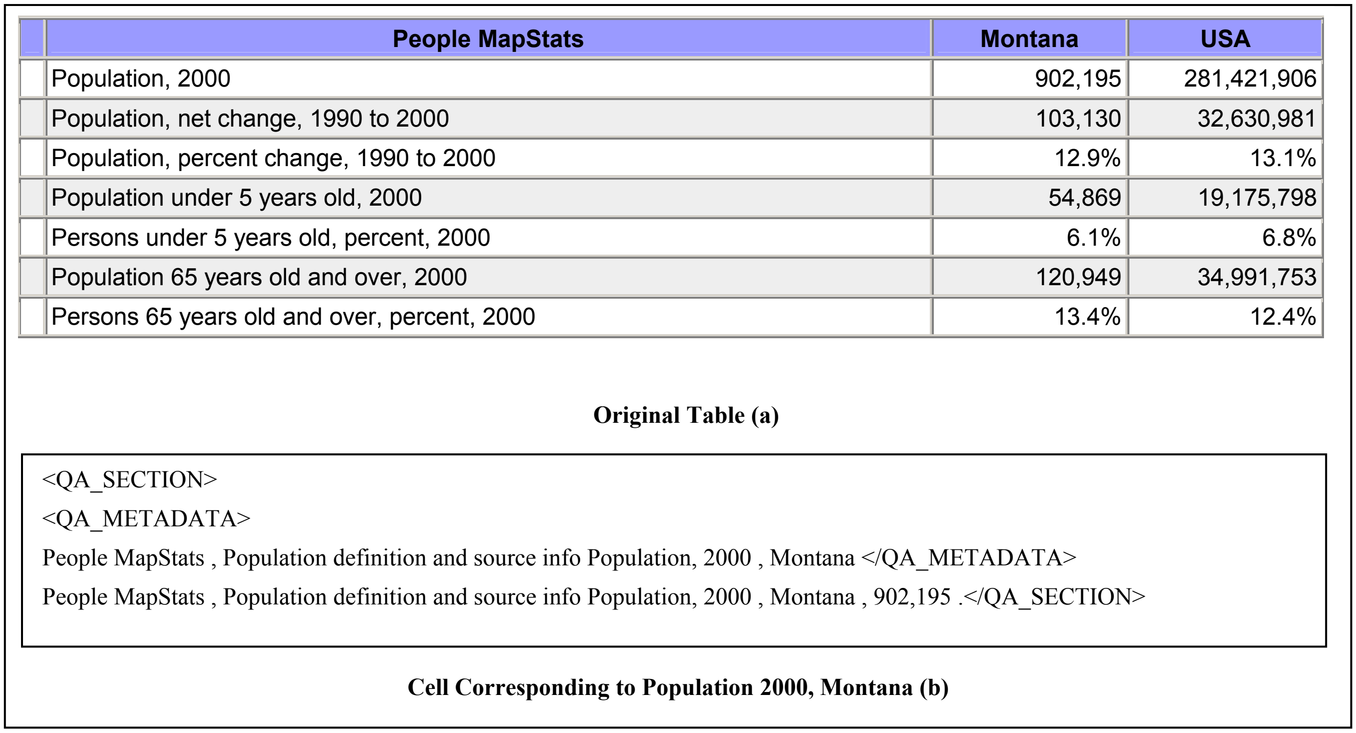
HTML Tables present a somewhat easier problem than text tables. The <table></table> tags clearly delineate the start and end of an author's construct. An example of an HTML table is shown in Figure 5a.

The heuristics for extracting table data are simple and effective.

- When tables are nested, the innermost table is considered to be the one holding the data.
- The title of the table is extracted and used as metadata for each cell.

[1] Score for each question is the reciprocal of the rank of the first correct response, or 0 if there is no correct response. The mean of these scores is reported.

- The first cell of the table (upper left hand corner) is also applied to each cell and is appended to the title.
- The remaining cells in the first row are considered headers for each column. If the last cells in a row are blank, take this as a sign of a multi-row header, and continue using the next row to add to the header text. The header for each column is associated with each cell below it.
- The first cell in each data row is considered the row header and the text is associated with each cell in the row.



	People MapStats	Montana	USA
	Population, 2000	902,195	281,421,906
	Population, net change, 1990 to 2000	103,130	32,630,981
	Population, percent change, 1990 to 2000	12.9%	13.1%
	Population under 5 years old, 2000	54,869	19,175,798
	Persons under 5 years old, percent, 2000	6.1%	6.8%
	Population 65 years old and over, 2000	120,949	34,991,753
	Persons 65 years old and over, percent, 2000	13.4%	12.4%

Original Table (a)

```
<QA_SECTION>
<QA_METADATA>
People MapStats , Population definition and source info Population, 2000 , Montana </QA_METADATA>
People MapStats , Population definition and source info Population, 2000 , Montana , 902,195 .</QA_SECTION>
```

Cell Corresponding to Population 2000, Montana (b)

Figure 5. HTML Table Examples

Each cell is written out as a sentence: title, column header, row header, data, since the scoring algorithm expects its data in the form of a sentence. A transformed cell can be seen in Figure 5b.

A QA experiment similar to the one used for text tables was performed. A page from MapStats was chosen, http://www.fedstats.gov/qf/states/30000.html, and 20 questions were generated from the data in the "Montana" column. The twenty questions were run against the same two databases as before. Five documents were retrieved for each question, and we recorded the MRR and the number of questions with an answer not found in the top five documents. The results are in Table 6. Again, we found the results encouraging.

Table 6. Experimental Results, HTML QA

Montana Questions	Unprocessed DB	QuASM
MRR	0	.367
# Not Found	20	12

5. QUESTION CLASSIFICATION AND ENTITY IDENTIFICATION

In order to find a likely answer to a question, an idea needs to be developed as to what the answer will look like. Is it the name of a person? A location? A number? A specific type of number? Figure 6 shows the categories classes used by QuASM. In addition to classifying questions, the answer entities must also be identified in the text being searched.

Identinfinder Classes

PERSON
ORGANIZATION
PERCENT
TIME
LOCATION
DATE
MONEY

QuASM Classes

URL
TEMPERATURE
HEIGHT
PERIOD
SPACE (VOLUME)
DENSITY
POWER
NUMBER
EMAIL
LENGTH
MASS
AREA
SPEED
ENERGY
ORDEREDNUMBER
BIOGRAPHY

Figure 6. Question and Entity Classes

5.1 Classification Experiments

The initial approach was to develop regular expressions to classify questions. While this approach works well for standard questions (Who is the CEO of GE? How much does a barrel of oil cost?), it works less well when the question strays from the expected form. Rather than try to develop more and more regular expressions, a language modeling approach was adopted.

The idea behind the language model is to discover the probability of the question (Q) given a question class (C). A unigram language model looks at the probability of individual words (Equation 1). The unigram model assumes that each word occurs independently 7]. A bigram model looks at the probabilities of pairs of words (Equation 2). The probability of a new word depends on the context, in this case the previous word [9].

$$P(Q|C)=P(w_1|C)*P(w_2|C)*...*P(w_n|C)$$

Equation 1 - Unigram Model

$$P(Q|C)=P(w_1|C)*P(w_2|C,w_1)*...*P(w_n|C,w_{n-1})$$

Equation 2 - Bigram Model

Language models for each question class were developed using three sets of classified questions:

- Questions extracted from GovBot logs and hand classified by our lab. GovBot was a database of US government web sites developed by the Center for Intelligent Information Retrieval (CIIR) at the University of Massachusetts and available for public search.
- Questions classified by Thomas Morton of the University of Pennsylvania.
- TREC Questions, also classified by Thomas Morton.

For each question, certain words were replaced with named entities. The idea behind this was that the entity **LOCATION**, for example, conveys more information than the name of a specific place. So a question like, “Who is the president of the US?” becomes “Who is the president of the **LOCATION**?” Unigram and bigram models were built for each question class using these transformed questions.

The questions containing named entities were transformed again, using part of speech tagging to label nouns. “Who is the president of the **LOCATION**?” then becomes, “Who is the **NOUN** of the **LOCATION**?” Unigram and bigram models were again built for each class.

The reason for building models with and without part of speech tagging is that information is both gained and lost by doing so. Certain nouns, such as “percentage,” give more clues to the question class left as is. By building models two ways, we hope to capture as much information as possible.

A 5-fold cross validation experiment was performed using the TREC questions. Precision for each of the four language models and the regular expression classifier are reported in Table 7 (tagging refers to entity and part of speech tagging).

Table 7. Experimental Results, Question Classification

Model	Precision
Regular Expressions	.59
Unigrams, no tagging	.74
Unigrams, with tagging	.56
Bigrams, no tagging	.73
Bigrams, with tagging	.60

Results with the language model can be improved by heuristically combining the three best scoring models above (unigrams with no tagging and the two bigram models). Aslam [1] uses the rank of documents returned by different search engines to combine the results of those search engines. Similarly, a weighted voting scheme was used to combine two models, unigrams with no tagging (M1) and bigrams with no tagging (M2). This voting scheme is a variation of a *Borda Count*, which Saari [8] shows to be a correct voting method. A score was assigned to the top five classes ranked by each model, with the top ranked class getting a score of 0.5 then down to 0.25 for 2nd, 0.1 for 3rd, 0.05 for 4th and 0.01 for 5th. M1 is given a weight of .6, M2 .4. The score for a class is then determined by a linear combination of the weight for the model times the score for the rank of that class by the model, M1*R1+M2*R2.

The model bigrams with tagging (M3) can improve performance in one special case. If M1 and M2 differ on the top ranked class, but agree on the 2nd ranked class, and M3 ranks the agreed 2nd ranked class first, then that class wins. This results in increasing precision to .75.

Neither method (regular expressions or language model) for classifying questions was ideal. However, the language model would often make mistakes that the regular expression classifier would judge correctly. To use this information to improve the precision, the two methods are combined using the following algorithm:

- Rank the classes based on language model score.
- Find all classes that match the question with a regular expression
- Choose as the answer the regular expression match ranked highest by the language model.

Using this method we were able to obtain a precision of .81 testing on the TREC questions.

5.2 Entity Identification Experiment

For entities not marked by Identifinder™, regular expressions are used to locate entities in the retrieved text. We are also careful not to reclassify entities already marked by Identifinder™.

For entities representing numbers, a two-step approach is taken. A regular expression is used to find a string representing a number (either in words, digits or a combination of the two). Once a number has been located, the following token is checked to see if the number can be further classified into a unit of measure. Once the number has been identified, it is tagged with a NUMEX tag, and the type field of this tag is set with the appropriate name (Figure 6).

For the non-number entities, a regular expression is used for each class to search the text for entities. Each match is tagged with an OBJECT tag, and the type field of this tag is set with the appropriate name (Figure 6).

Table 8. Experimental Results, Entity Identification

Number of entities in documents	8043
Number of entities recognized	8315
Number of entities recognized	7926

correctly	
Precision (correct/recognized)	.9532
Recall (correct/total)	.9855

To test entity identification, entities were hand labeled and compared against the output of our program for precision and recall (Table 8).

6. QUESTION ANSWERING EXPERIMENTS

The web crawl and content selection resulted in a collection of 177,670 documents. From these, a random set of documents was selected. These documents were used to generate 73 questions used in testing. The questions covered data in text tables, html tables and prose.

The experiment was designed to compare a QA system based on unaltered web files with a QA system based on files processed into smaller information units as described above. The original documents were indexed into an Inquery database (Unprocessed DB). Separately, the same documents were processed for text and html tables, and the remaining text sectioned into smaller units. These processed documents were also indexed into an Inquery database (QuASM). (These are the same DB used in the text and HTML table experiments.)

The 73 questions were spilt into groups of ~15 for batch runs. Each batch run was timed to see if there was an improvement in efficiency.

Table 9. Experimental Results, QuASM vs. Unprocessed Data

Test Questions	Unprocessed DB	QuASM
MRR	.209	.253
# Not Found	54	50
% Not Found	74.0%	68.5%
Avg. CPU Time per Question (in seconds)	137.5	8.1

6.1 Discussion

The goal of our work is to improve performance and efficiency when answering questions from web data. Our results indicate a first step in that direction. MRR was increased, while the average time to process a question was lowered.

Two factors make this a task more difficult than a TREC experiment. The emphasis on answering questions of a statistical nature, where the answers require finding a named entity, increases the difficulty of the search, especially when the entity is not in the middle of a sentence (in a table, for example). The data is from a more heterogeneous environment and unlike news articles; there is little consistency in format. These factors lead to lower scores versus systems reported in TREC QA tracks.

Processing documents into smaller units had a dramatic effect on efficiency. The size of web documents varies greatly, from a few bytes to millions of bytes. The action of table extraction and document sectioning reduce this variability and ensure the answer identifier has a reasonable amount of information to search. Time to process the average question was reduced by better than a factor of sixteen.

While the improvement in MRR was modest, there are indications that this score can be raised. Of the questions that had a correct answer in the top five, thirteen were answered by both systems. It would appear processing tables and prose both enhances and diminishes the information available for finding answers.

As shown in the experiments on text and HTML tables, there is an information gain through table transformations. Sectioning can also lead to an information gain. The question, “What is the molecular weight of calcium cyanamide?” is answered correctly by QuASM but incorrectly by the previous system, despite the fact that the correct document is retrieved from the unprocessed DB. The original document has a number of blank lines between the weight (80.11) and the units (g/mol). This is not proximate enough for the entity identifier to correctly tag 80.11 as a **MASS**. When the document is sectioned, however, the blank lines are stripped away, and the entity identifier correctly labels the weight.

The question, “What percent of watersheds does the EPA want to restore by 2005?” sheds light on how information is diminished by the processing. In the original document containing the answer, the query term **percent** does not appear near the answer to the question (a % is used). The term **percent** does appear later in the document. When the document is sectioned, the term percent is lost entirely to the small unit containing the answer. Other sectioned documents enhance the term frequency (*tf*) of **EPA** by repeating it in the metadata, increasing the score for retrieval. These two factors lead to the failure to retrieve the sectioned document containing the answer.

We have postulated that a two stage retrieval system could solve this problem. Documents are sectioned, but the sections remain in a single document for retrieval by the search engine. A second function then determines the best sections for the query, and only those are searched for an answer. This will be part of our future work.

The question classifier correctly identified 51 of the 73 questions (70%). Some incorrectly classified questions have pointed to deficiencies in the regular expressions for these classes, which underscores the need to develop better models for these classifications. For example, the question "What is the surface area of Lake Ontario?" is classified as a **LOCATION**, due to the language model giving that class a higher score.

7. FUTURE WORK

This implementation of QuASM used query formation heuristics based on previous work. This makes initial IR sensitive to how the question is formed. For example, in the test on text tables, if we phrase the question as, “How many acres were planted with onions in the spring of 1997in AZ?” the query generated treats “spring of 1997” as a phrase, and fails to find the table documents since terms are spread throughout the cell metadata. One area of research will be to change query generation to create more table friendly queries.

HTML and text table algorithms currently assume the first column contains the row name. Often, the row name or label is spread over a few columns. Better row and column names should give a boost to performance.

The heuristics that locate an answer in a document is another area of future study. One of the changes we made to this part of the system was to look for the last name entity match when the document represents a table cell (since we know the data is at the end of the document). Improving these heuristics for this task will be a continuing area of research.

8. ACKNOWLEDGMENTS

This work was supported in part by the Center for Intelligent Information Retrieval and in part by NSF grant #EIA-9983215.

The authors would like to thank Alvaro Bolivar for designing the web interface, David Fisher and Steve Harding for advice and technical support, R. Manmatha for his ideas on smoothing, and Andre Gauthier for the disk space.

Any opinions, findings and conclusions or recommendations expressed in this material are the author(s) and do not necessarily reflect those of the sponsor.

9. REFERENCES

[1] Aslam, J. and Montague, M. Bayes optimal metasearch: a probabilistic model for combining the results. *SIGIR 2000*, pages 379-381

[2] Bikel D., Miller S., Schwartz R. and Weischedel R. Nymble: a high performance learning namefinder, *Proceeding of the fifth Conference on Applied Language Processing*, Washington, USA, 1997

[3] Callan, J.P., Croft, W.B., and Harding, S.M. The INQUERY retrieval system. In *Proceedings of the 3rd International Conference on Database and Expert Systems Applications* (1992), pages 78-83.

[4] Finn, A., Kushmerick, N. & Smyth, B. Fact or fiction: Content classification for digital libraries. *Joint DELOS-NSF Workshop on Personalisation and Recommender Systems in Digital Libraries* (Dublin), 2001.

[5] Li, X. and Croft, W.B. Evaluating Question Answering Techniques in Chinese. Presented as a poster at *HLT 2001, in Notebook Proceedings*, pages. 201-206.

[6] Ponte, J.M. and Croft, W.B. A Language Modeling Approach to Information Retrieval. In *Proceedings of SIGIR '98*, pages 275-281. Melbourne, Australia, 1998.

[7] Pyreddy, P. and Croft, W.B. TINTIN: A System for Retrieval in Text Tables. In *Proceedings of the Second International Conference on Digital Libraries*, pages 193-200, 1997.

[8] Saari, D. and Valgones, F. Geometry, Voting and Paradoxes. In *Mathematics Magazine*, pages 243-259, October, 1998.

[9] Song, F. and Croft, W.B. A General Language Model for Information Retrieval. In *Proceedings on the 22nd Annual International ACM SIGIR Conference*, pages 279-280, 1999.

[10] Voorhees, E. The Trec-8 Question Answering Track Report. In *Proceedings of TREC-8*, 1999.