Combining Substructures to Uncover The Relational Web

B. Cenk Gazen
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
bcg@cs.cmu.edu

June 25th, 2004

Thesis Committee:
Jaime Carbonell, Carnegie Mellon University (chair)
William Cohen, Carnegie Mellon University
John Lafferty, Carnegie Mellon University
Steven Minton, Fetch Technologies

Abstract

I describe an approach to automatically convert web-sites into relational form. The approach relies on the existence of multiple types of substructure within a collection of pages from a web-site. Corresponding to each substructure is an expert that generates a set of simple hints for the particular collection. Each hint describes the alignment of some tokens within relations. An optimization algorithm then finds the relational representation of the given web site such that the likelihood of observing the hints from the relational representation is maximized.

The contributions of the thesis will be a new approach for combining heterogeneous substructures in document collections, an implemented system that will make massive amounts of web data available to applications that use only structured data, and new search techniques in probabilistic constraint satisfaction.

1 Introduction

1.1 Motivation

Even though the amount of information on the web has been growing at an incredible rate, software applications have only been able to make use of it in limited ways, such as spidering and indexing of words on a page, activities that do not require a deep understanding. This is mainly because as data is transformed into web format, its inherent structure is replaced with formatting structure that makes the data easier to absorb for humans but harder for computers.

There have been numerous attempts at getting applications to make more use out of the information on the web. These attempts can be broadly grouped into extraction and labeling attempts. The extraction work focuses on learning rules for extracting small segments of text. Because this approach requires extra human effort to prepare training sets, it is limited to converting only selected pieces of information to machine-processable form. In the labeling approach, the data is labeled with agreed-upon tags. Labeling also requires extra human effort for maintaining a common set of tags, assigning tags to data and keeping the tags attached to data. Either approach is limited in its applicability because of the extra effort needed.

One of the early projects that fall into the extraction category is the WebKB project at CMU[5]. The WebKB system starts with an ontology and a training set whose elements are instances of the classes in the ontology and learns general procedures to extract new instances from unseen pages. In contrast to most other extraction systems, WebKB applies multiple learning algorithms, uses both content and link structure, and learns procedures that are general enough to work on pages from sites different from those in the training set. However, it still suffers from the limitations of the extraction approach, because information is extracted for only the classes in the ontology for which a training set is maintained manually.

As a complement to the extraction and labeling techniques, I propose a new approach that will enable automatic conversion of web-sites into relational form. In the relational form, the data is organized into tables each row of which contains a single entity. Entities may be linked to related entities through references.

As an example, let's look at part of the relational representation of the data on a page showing a list of products (fig. 5 in appendix A) and two pages that give detailed information about individual products (fig. 6, 7). The main table of interest is the list of products. The tabular form groups together similar entities, the products, and also separates one entity from the other with rows.

t_0:

Milwaukee 1 / 2 Hole - Hawg Drill	299.00	118169	1675 - 6	$ref(t_1, 0)$
Milwaukee 3 / 8 Close Quarter Drill	159.00	118817	0375 - 6	$ref(t_1, 1)$
Milwaukee 1 / 2 Magnum Drill	139.00	74453	0234 - 6	$ref(t_1, 2)$
DeWalt 7.8 Amp ...	129.00	124552	DW236K	$ref(t_1, 3)$
Milwaukee 8 Amp , 1 / 2 Drill	129.00	137724	0300 - 20	$ref(t_1, 4)$
Hitachi 9 Amp 1 / 2 Drill	119.00	166939	D13VF	$ref(t_1, 5)$
Makita 1 / 2 Variable Speed Drill	99.00	128474	6303H	$ref(t_1, 6)$
Makita 3 / 8 Drill	59.97	167414	6408K	$ref(t_1, 7)$
DeWalt Heavy - Duty ...	59.00	19808	D21008K	$ref(t_1, 8)$
Hitachi 3 / 8 6 Amp Drill	49.98	118433	D10VH	$ref(t_1, 9)$

Through the $ref(\ldots)$ notation, the last column links the entities to another table whose data comes from the detail pages:

t_1:

0	Milwaukee 1 / 2 Hole - Hawg Drill		1/2	300/1200 rpm	
1	Milwaukee 3 / 8 Close Quarter Drill	3.5 Amp	3/8	0-1300 rpm	
2	Milwaukee 1 / 2 Magnum Drill	5.5 Amps	1/2	0-850 RPM	Keyed
...					

The relational representation can also capture information that is not necessarily in relational form to start with. For example, the following table represents the navigation links on the product list page. Being a static set of links, they would normally not be maintained as a table.

t_2:

http://www.lowes.com/lkn?action=topicSelect&topic=featuredBrands	Brands
javascript:LILWindow('http://www...index.html')	Design Tools
http://www.lowes.com/lkn?action=topicSelect&topic=buyGuide	How To
...	
http://www.lowes.com/lkn?action=topicSelect&topic=weeklyAd	Weekly Ads

Some benefits of converting the data into relational form are:

- Having been around since the 1970s[1], relational representation of large volumes of data is one of the best understood and most widely applied types of representation. (In fact, the data for a large number of web sites is likely to be already in relational databases as evidenced by the recent development of tools to convert relational data into semi-structured form[3].) With the proposed approach, all the techniques for analyzing relational data become applicable to web data. For example, one can query for the number of books written by an author or for the stores that sell a book below the average price of that book.
- Data becomes readily available for further processing by intelligent agents.
- Indices can be built for searching specific fields. For example, it becomes possible to find pages about movies whose director is a particular person. Existing indices on the web cannot distinguish between pages where the person is listed as a director and as an actor.
- A common task is to track incremental change on the web, such as the status of a shipment or the current headlines. Having this information in relational form rather than as a sequence of tokens improves the performance of such tracking applications because the changes can be identified at a finer detail.
- As with other unsupervised learning approaches, at the very least the structure that is discovered facilitates further analysis, by grouping data in chunks that can be analyzed as a whole.

By converting web sites into relational form automatically, the new approach enables a deeper machine-understanding of vast amounts of information that is available on the web.

1.2 Approach

The extraction and labeling work within the web domain have had varying amounts of success. Most of these approaches focus on a particular structure of web pages, typically the syntactic structure or the link structure. I propose to build a unifying framework where the multiple types of substructure that are identified by multiple approaches can be combined. The new approach finds a relational representation of a web site by integrating the output of a number of experts each of which look for a particular type of structure.

The proposed approach relies on the observation that different types of substructure are representative of the underlying structure over different pages of a web site and even over different parts of a single page. If the different substructures can be combined effectively, the resulting representation will be better than any other that uses only one substructure.

The relational form serves as the unifying representation because first, within it, it is possible to represent explicitly the types of substructure that are typical of data on the web and second, it is simple enough that it can be used easily by other agents.

Let's look at some examples of substructure. The following is a list of some of the substructures that can be observed on the single page (fig. 5).

- **Vertical Lists.** There are several locations on the page where the boundaries of items are aligned vertically. This is usually a good indication of an underlying list structure. For example, on the left side of the page, the navigation bar contains a list of links to the departments of the store. In the middle of the page is the list of products, which is the main content of this page. However, not all vertically aligned elements are lists. The entry for a product has several lines (name, item number, link to details, checkbox) which are not items in a list.

- **Horizontal Lists.** Similar to but less common than vertical lists are horizontal lists. Some examples on the page are the navigation bar on the top (Brands, Design Tools, ...) and at the bottom (Corporate, Help Desk, ...). The elements of the latter are vertical lists.

- **Common Formatting.** Visual cues are commonly used to show the structure of the data. For example, the item and model numbers are set in small gray letters whereas the prices are set in large, black, and bold letters indicating that the values in these fields are related to others in the same field but different from other pieces of text on the page.

- **Common Field Features.** The values in a single field usually follow a specific pattern. On this page, the item numbers are five or six digit integers. Model numbers among products of the same brand also have common patterns. Like common formatting, this hints at a relation between the values of the same field.

- **Repeated Values.** The URL attached to the name of a product and that attached to the "View Details..." link are the same for a given product. This structure is valuable in finding the boundaries of an item correctly. The two links and anything in between them are likely to be information about the same item. In addition, the URL for the "Buy Now!" button shares the product id (a superstring of the displayed item number) with the "View Details..." URL.

- **Numbered Lists.** At the bottom of the page, the links to all the pages that contain parts of the product list are indexed by the page numbers (1 and 2). Such sequences of integers are good indicators of list structure. But also note the counter-example of "Page 1 of 2".

Let us now look at some examples of substructure within a collection of pages (fig. 5, 6, 7)

- **Common Page Structure.** The navigation bars at the top, left and bottom of the pages are identical. Also, among the product pages, there is even more common structure. For example, below the product image there is a zoom button, and below the button is a list of product attributes (some of which are common among multiple products). To the right is a list of features for each product, and at the bottom is an optional image showing the customer ratings.

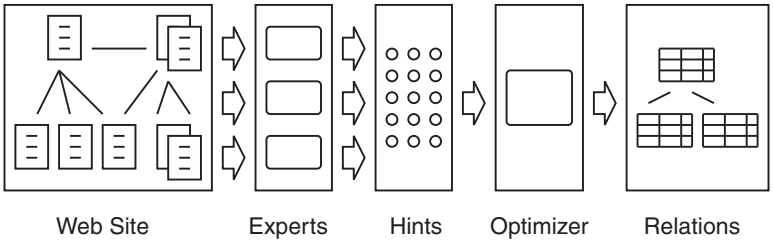


Figure 1: Architecture

- **Common Content.** Some of the content that is on the navigation page is also on the product pages. For example, product name, item number, model, price and "Buy Now!" URL. This is useful in associating the link on the navigation page with the text around it correctly.
- **Back Link.** On product pages, the URL attached to the "View All Products in this Category" link is the same as the URL of the product list page. This usually indicates a hierarchical structure of the pages rather than just a navigational shortcut.
- **Next Link.** Links that include "next" as a substring of their labels almost always connect to continuation of pages that contain lists.

The proposed approach (fig. 1) exploits the heterogeneous set of substructures by defining a *hint language* in which the the expert responsible for each type of substructure expresses its findings as hints to the optimizer. The best relational representation with respect to the hints is then found by an optimization algorithm.

One piece of information that is missing from the relational representation is column labels (zip code, price, last name, etc.). This is because the proposed approach focuses on finding the inherent structure of data whereas column labels relate the data to an external, larger knowledge base. However, the relational representation will make it significantly easier for other intelligent agents to label the data.

2 Previous Work

The work leading to my proposal has been on developing a set of algorithms that discover some of the substructures from collections of web pages and also building a system that combines these algorithms in an ad hoc way. In sections 2.1 and 2.2, I describe the template substructures and related algorithms. Section 2.3 is a description of of the ad hoc approach. In section 2.4, I describe the implementations of the algorithms of sections 2.1, 2.2, and 2.3. The proposed approach tackles the same problem as the ad hoc approach, namely converting whole web sites into relational form automatically, but does this in a more elegant and flexible way.

2.1 Templates

One of the most important types of substructure arises from the fact that an efficient way to exchange information after similar information has been exchanged is to take the structure of the first exchange and update it with the new parts. Having learned where the information about the weather conditions are in the following segment:

```
<TD VALIGN=MIDDLE ALIGN=CENTER CLASS=obsInfo2 WIDTH=50%>
  <B CLASS=obsTempTextA>54&deg;F</B>
```

```
</TD></TR>
<TR><TD VALIGN=TOP ALIGN=CENTER CLASS=obsInfo2>
  <B CLASS=obsTextA>Partly Cloudy</B>
</TD>
<TD VALIGN=TOP ALIGN=CENTER CLASS=obsInfo2>
  <B CLASS=obsTextA>Feels Like<BR>54&deg;F</B>
</TD></TR>
```

it is easy to write or read this segment:

```
<TD VALIGN=MIDDLE ALIGN=CENTER CLASS=obsInfo2 WIDTH=50%>
  <B CLASS=obsTempTextA>77&deg;F</B>
</TD></TR>
<TR><TD VALIGN=TOP ALIGN=CENTER CLASS=obsInfo2>
  <B CLASS=obsTextA>Fair</B>
</TD>
<TD VALIGN=TOP ALIGN=CENTER CLASS=obsInfo2>
  <B CLASS=obsTextA>Feels Like<BR>75&deg;F</B>
</TD></TR>
```

This idea of preserving structure as much as possible leads to templates. A *template* is a sequence of alternating *slots* and *stripes* where the stripes are the common strings among all the pages and slots are the placeholders for pieces of data that go in between the stripes. A template for the two segments above is shown below. The slots of the template are the labeled boxes.

```
<TD VALIGN=MIDDLE ALIGN=CENTER CLASS=obsInfo2 WIDTH=50%>
  <B CLASS=obsTempTextA>[ actualTemperature ]&deg;F</B>
</TD></TR>
<TR><TD VALIGN=TOP ALIGN=CENTER CLASS=obsInfo2>
  <B CLASS=obsTextA>[ currentCondition ]</B>
</TD>
<TD VALIGN=TOP ALIGN=CENTER CLASS=obsInfo2>
  <B CLASS=obsTextA>Feels Like<BR>[ feelsLikeTemperature ]&deg;F</B>
</TD></TR>
```

Pages generated using a single template are usually pages describing the details of a single item. Some examples of such pages are pages returned in response to an ISBN query on a bookstore site, a reverse phone-number lookup on white-pages, or a course number query on a college registrar site. When pages are generated from a single template, we will say that they are of the same *page-type*.

One way to find the template of a set of pages is to find the longest common subsequence (LCS) of the token sequences of all the pages. A subsequence of a sequence is the same sequence with some of the elements left out. A sequence is a common subsequence of a set of sequences if it is a subsequence of all the sequences in the set. The LCS, which is not necessarily unique, is a common subsequence that has the maximum length. The LCS immediately gives the stripes of the template and with a little bookkeeping, the slots can also be found.

The LCS algorithm is simple and can be made to run on long documents with acceptable memory usage[9], but it is not very fast. To improve the performance, we assume that the stripes are unique strings within documents. (In practice, this assumption holds for most stripes on detail pages.) With this simplification, the template is found by first finding all substrings common to all the pages but that are also unique within each page, as suggested in [13]. Since the substrings might occur in a different order on each page, the algorithm then finds the longest sequence of substrings that is common to all the pages. This sequence of common substrings is the sequence of stripes of the template. Once the template is known, it is easy to find the data on a page by finding the substrings in between the stripes of the page.

Let's look at an example run of the algorithm on the following three strings:

```
<a href=yahoo.html id=0 class=bookmark>yahoo.html</a>
<a class=bookmark href=google.html id=1>google.html</a>
<a class=bookmark href=cmu.html id=2>cmu.html</a>
```

Among these three strings, the set of common unique token sequences is {`.html</a>`, `.html id=`, `</a>`, `<a`, `=bookmark`, `>`, `bookmark`, `class=bookmark`, `href=`, `html id=`, `html</a>`, `id=`}. Next for each input string, a non-overlapping set of occurrences of these sequences is found. This determines an ordering of the token sequences for each string:

```
<a, href=, .html id=, class=bookmark, >, .html</a>
<a, class=bookmark, href=, .html id=, >, .html</a>
<a, class=bookmark, href=, .html id=, >, .html</a>
```

The LCS of these three sequences is (`<a, href=, .html id=, >, .html</a>`). So the template is:

```
<a [0] href= [1] .html id= [2] > [3] .html</a>
```

Using the template, the data can be extracted into a table:

	yahoo	0 class=bookmark	yahoo
class=bookmark	google	1	google
class=bookmark	cmu	2	cmu

2.2 Templates for Lists

Another common substructure of documents is tabular structure. In addition to reusing the same structure for rows, tables also group together similar items in columns.

Tables are typically used when the results of a query that retrieves more than one record are to be displayed. Multiple records are displayed in a list whose rows are generated by iterating over the records. The list is usually plugged into an outer template.

Just as a page shares a common structure with pages of the same page-type, each row shares a common structure with the other rows in the same list, and the template idea applies equally well to the rows of a list as it does to pages. Let us call the template representing the common structure of rows in a list a *row template*. We extend the basic template model so that slots are either *content slots* as before, or *list slots*, which are containers for row templates.

As an example, let's create a template to generate html segments like the following segment:

```
<h2>Cities in Pennsylvania</h2>
<ul>
  <li>Aaronsburg</li>
  <li>Aaronsburg</li>
  <li>Abbottstown</li>
  <li>Abington</li>
  <li>Ackermanville</li>
</ul>
```

The outer template generates the first, second and last lines of the segment and "calls" the inner template (represented by a double box) to generate the elements of the list:

```
<h2>Cities in [state]</h2>
<ul>
  [[cities]]
</ul>
```

The inner template, `cities`, simply generates one row of the list.[1]

[1] In practice, more information needs to be specified before pages can be generated. In particular, in addition to the slots being linked to the underlying data source, a method for determining the set of cities for a particular instantiation of `state` needs to be specified.

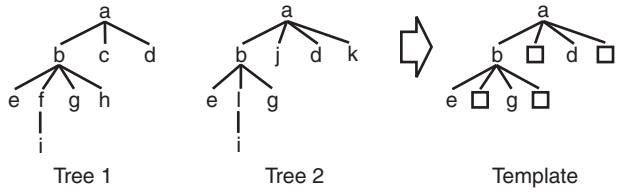


Figure 2: Templates for Trees

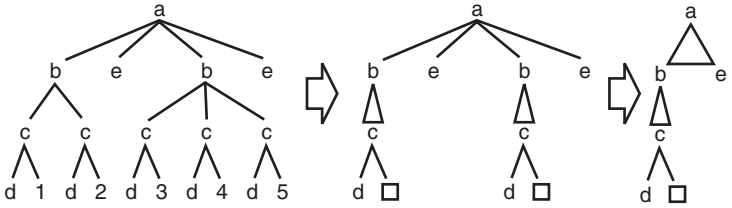


Figure 3: Row Templates

 city

The problem of inducing a template from a set of html pages becomes much harder with the introduction of list slots. To simplify the problem, we work on the document object model (DOM) tree of pages and make the following assumptions: The nodes representing the rows of a list are the children of a single parent node, and there can be at most one list within the children of a single parent node.[2]

Before going into the details of the template induction algorithm, we describe templates in the domain of trees. Given a set S of sequences of DOM elements, the TreeTemplate algorithm finds the LCS of S and then for each node in the LCS, builds a set of sequences of elements by gathering the child nodes of the node from every tree, and calls itself with that set. For example, given the root nodes of Tree 1 and 2 in figure 2, the algorithm first finds the LCS of the one element sequences [**a**] and [**a**] and then proceeds down to the child sequences [**b c d**] and [**b j d k**]. The LCS of these two is [**b d**], so it first recurses down to the child sequences of the two **b** nodes and then to those of the two **d** nodes to get [**e g**] and []. The template is the tree of LCSs found at each node. Once the template is computed, it can be used to extract the data in the slots. This is done by finding the nodes in the original DOM structures that are not in the stripes of the template. The nodes that fall into the slots are **c**, **f**, **h**, and **i** for Tree 1, and **j**, **k**, **l**, and **i** for Tree 2. By keeping track of the slots that the nodes go into, the data can be aligned in columns and represented in a table:

f i	**h**	**c**	
l i		**j**	**k**

We now return to the template induction algorithm. HierarchicalTemplate has two steps. In the first step, it processes the DOM tree of each page to find all the lists and the corresponding row templates. In the second step, it finds the page template.

The first step processes the tree bottom-up. At each node, the algorithm looks for a repeating pattern among the child nodes. This is done by using templates as a similarity measure between two DOM subtrees. (Similar subtrees will have templates with more stripes than those that

[2]This assumption does not necessarily hold. One of the goals of the proposed approach is to be able to use this same kind of substructure while relaxing these assumptions.

are not similar.) In particular, the algorithm looks for consecutive sequences of nodes whose similarity measure is above a threshold. Those sequences that pass the similarity test are assumed to be the rows of a list and are used as inputs to TreeTemplate to induce a row template. As the algorithm traverses up towards the root, it replaces the repeating sequences with row templates. For example, given the tree on the left in figure 3, the algorithm first finds row templates for the inner lists within the children of **b** nodes. The resulting tree is shown in the middle of the figure, where the row template is denoted by a triangle (instead of a line) connecting it to its parent. Next, the algorithm examines the children of node **a**. With repeating subtrees represented as row templates, the algorithm can detect the similarity between the first two and the last two children of node **a** and induce the row template as shown on the right. We will refer to this intermediate tree as a row-template tree.

The second step of the induction algorithm merges the row-template trees using a variation of the basic template induction algorithm, in which row template nodes are matched only with row template nodes and regular nodes only with regular nodes.

Another way to look at templates is to consider the web-publishing system as a function that takes as input relational data and outputs a set of hyper-linked pages. Recovering the relational data from the set of hyper-linked pages is equivalent to estimating this function with a template, and inverting it to infer the underlying data.

2.3 Utilizing Multiple Types of Substructure

The template substructure of pages is useful in extracting data from a set of pages that are all of the same page-type. But let us look at the case when the pages are not of the same page-type. If we knew the page-type of each page, then we could easily find the template for each page-type. Conversely, if we knew the template for each page-type, then we could determine the page-type of each page. Lacking both pieces of information, clustering is a logical direction to follow, so we look at it next.

The obvious approach to clustering web pages is to use templates as a similarity measure and apply a standard clustering algorithm. The problem with this is that among pages from a single web site, there is enough commonality that the differences are usually outweighed. In fact, some clusters of pages which are easily identified by other types of substructure, such as links in a list, can be found with the template substructure only if the the pages happen to share a template among them but with not other pages.

An alternative, but ad hoc, approach is to use the template substructure in evaluating the clusters which are found by other means. The advantage of this approach is that it allows a second substructure to be used as the means to search through the cluster space. In the prototype implementation, the pages are clustered into three groups: navigation pages, detail pages, and "other" pages. The clustering algorithm generates hypotheses based on the link structure of the site (for example, navigation pages are visited before detail pages, possibly point to one or two other navigation pages, etc.) and evaluates the resulting clusters with a minimum description length (MDL) metric based on templates.

The template-based MDL metric is derived from the observation that a template can be used to compress a set of pages. After a template is induced, it is possible to factor out the stripes from each page and encode the set as the template and a set of slots. If the template has enough content in its stripes, the resulting encoding reduces the number of bits required to represent the pages without loss of information. This idea can easily be turned into an evaluation metric for clusters where the goal is to minimize the number of bits to represent all the pages. The resulting MDL metric balances between the two extremes of having a single cluster of all the documents but with a template that has no stripes, and having one cluster for every document but with each template fully covering its document.

2.4 Implemented Systems

2.4.1 AutoWrap Version 1

The first version of AutoWrap is an implementation of the HierarchicalTemplate algorithm. It takes as input a set of pages, builds a hierarchical template on the DOM trees of the pages, uses the template to extract data, and outputs the extracted data in relational form. Appendix B contains a set of example pages and the corresponding output of AutoWrap V1.

Because only one template is induced for the given set of pages, it is crucial that the input consists of pages that are of the same page-type. A single page that does not fit the template causes the system to induce an empty template and an empty template in turn causes the tokens of each page to be extracted as one large slot. This implies that the pages need to be classified very accurately, which is a difficult task even when done manually.

In the normal case where the input set consists of pages of the same page-type, AutoWrap V1 is still limited by the assumptions of HierarchicalTemplate. In particular, there is a significant number of cases where a single parent node contains more than one list among its children. This happens when a small number of lists with different row templates are put together under one parent node. AutoWrap V1 picks the list whose rows are most similar to each other and finds a row template for it, but leaves the nodes for the other lists untouched. As a result, some of the lists are never discovered. For example, in the sequence [**a 1 a 2 a 3 x b b b b**], AutoWrap would discover the list of **b**'s but not the list that contains the **a**'s.

AutoWrap V1 is not very good at finding the correct row breaks. This is because it relies only on the template-based similarity between rows. This measure is useful for identifying repeating sequences of nodes, but not for determining the boundaries of the unit of repetition. For example, just by looking at the similarity between consecutive sequences in the sequence [**a, b, c, a, b, c, a, b, c, a**], it is impossible to tell if each rows is [**a, b, c**] or [**b, c, a**].

2.4.2 AutoWrap Version 2

The second version of AutoWrap is a prototype implementation that utilizes the link and template substructures of a web site. Instead of taking a set of pages, it takes a single entry URL[3] as input. Its built-in web spider creates a map of the pages connected to the entry page. After the map is created, the system starts looking for a clustering of the pages into the three clusters "navigation", "detail" and "other". The clustering that gives the best template-based compression is chosen and data is extracted from the detail pages using the induced flat template. Appendix C contains an example site, and the data extracted from the detail pages.

AutoWrap V2 suffers from the fact that it clusters pages into three predefined groups. This means that the spidering step needs to be controlled so that most of the retrieved pages belong to either the "navigation" or "detail" group. Otherwise, picking the navigation and detail pages among a large set of "noise" pages becomes difficult and both performance and quality degrade quickly.

One of the motivating factors for my thesis is that even though it is easy to put together systems that handle particular types of web sites by focusing on particular types of structures, these systems do not generalize well to other types of web sites. Extending them in ad hoc ways lead to inelegant systems that are still limited to a few different types of web sites. My goal is to make the next version of AutoWrap, which I will be developing based on the approach I am proposing, an elegant and flexible solution for handling a wide variety of web sites.

[3] In case the URL is not sufficient to retrieve the page, the contents of the entry page can also be input.

3 Proposed Work

As with most large collections of information, the data on the web inherently exhibits multiple types of substructure. Some of these substructures are obvious from the organization of the pages. For example, a hierarchical data set, such as information about the schools, departments, programs, and courses of a university, is usually presented so that the pages correspond to the nodes of the hierarchy. Some others are apparent from page layout. The items of a list are almost always aligned vertically or horizontally. And others are hinted at by visual or syntactic similarities between data items. Fields such as dates, addresses, or phone numbers follow a certain set of syntactic patterns. Other fields are typically typeset in bold, italic, small or large fonts or in particular colors so that they are distinguishable from each other.

Current approaches to getting machines to understand the data on the web have been limited to using a few substructures at a time. As a result, machines have been able to work either in simplistic ways (e.g., indexing) or in limited domains (e.g., extracting job postings).

Thesis Statement: By combining the information available from multiple heterogeneous substructures of a document collection, it is possible to recover the hidden relational structure of the underlying data with high accuracy.

3.1 Types of Substructure

Below is a list of some of the types of substructure that can mirror the underlying structure of the data. They also share the characteristic that each can be detected easily with simple algorithms. We'll refer to the algorithms that can discover substructures as *substructure experts*.

- **Template Coverage.** The stripes of a template should generate a large portion of the tokens of a page.
- **URL Patterns.** Pages with similar URLs are of the same page-type.
- **Next Links.** Pages connected to one another by next-links are of the same page-type.
- **Links in a List.** Pages pointed by the links in a list column are of the same page-type.
- **Indexed Items.** Items indexed with consecutive integers are rows in a list.
- **Redundant Data.** The data around a link on a list page is also found on the detail page pointed by the link.
- **Page Layout.** The layout of data on the displayed page matches that of the data.
- **Domain Specific Knowledge.** For example, mailing addresses should be aligned on a column.

Before we look at how to turn these abstract substructures into actual experts in section 3.3, I will describe the relational representation in which the experts will express their output.

3.2 Relational Representation

3.2.1 Definitions

A page is a sequence of tokens $(t_0, t_1, ..., t_{n-1})$. A token is a string with the property that breaking it down further does not yield shorter tokens that can possibly go into different cells and result in a better relational representation. For instance, in most cases natural text can be tokenized into sentences, and even paragraphs, with no negative effects on the relational form.

A set of pages together with the links between the pages form a site $(P, l : token \mapsto page)$ where P is a set of pages and l is a function mapping tokens to pages. l is defined for only a small subset of tokens, but for every token t that it is defined for, $l(t) \in P$.

We represent relations as tables whose rows are ordered. Each cell of the table is either a content-cell that contains a sequence of tokens or a reference-cell that contains a reference to a subset of rows from another table. Each row of a table has an associated key, not necessarily unique. Formally, a table is a 5-tuple $(rows, cols, k : i \mapsto key, c : i \times j \mapsto (t_0, t_1, ..., t_{n-1}), r : i \times j \mapsto (table, key))$ where $rows$ is the number of rows, $cols$ is the number of columns, k is a function that maps row indices to keys, c is a function that maps cells to their contents, and r is a function that maps cells to references to other tables. For any cell, only one of c and r is defined. The functions k and r are analogous to primary keys and foreign keys in relational databases. The pair $r(i, j) = (t, k)$ is a reference to rows $i_0, i_1, ..., i_{n-1}$ of table t for which $k(i_j) = k$. In other words, (t, k) selects the rows of t that have a key of k. We will refer to the sequence $(i_0, i_1, ..., i_{n-1})$ as $r_i(t, k)$. Circular references are not allowed.

Next, we define three functions ts, ts', and ts'' to map cells and tables into token sequences. The token sequence of a set of adjacent cells is obtained by concatenating the tokens of each cell in row-major order. When a cell is a reference-cell, the tokens are obtained from the set of rows referred by the cell. Formally, we start by defining the token sequence function ts for a single cell. The tokens of a content-cell are its contents: $ts(t, i, j) = c(i, j)$ if c is defined for (i, j). The tokens of a reference-cell are the tokens of the rows referred by it: $ts(t, i, j) = ts'(r(i, j))$ if c is not defined for (i, j). Next, we define the token sequence of a set of rows, which is the sequence obtained by concatenating the sequences of rows in order: $ts'(t, k) = concat(ts''(t, i)$ for i in $r_i(t, k))$. Finally, the token sequence for a row is the concatenation of the token sequences of the cells in that row. $ts''(t, i) = concat(ts(t, i, j)$ for j in $(0, 1, ..., cols - 1))$.

The relational representation of a site (P, l) is $(T, p_t : page \mapsto (table, key))$ where T is a set of tables and p_t is a mapping from pages to references. For every page $p \in P$, the token sequence of the reference is equal to the page: $ts'(p_t(p)) = p$. In other words, any page can be reconstructed by looking at some set of rows in one of the tables.

3.2.2 Example

The example site (fig. 4) is a dictionary of small counting numbers. The home page contains the beginning of a list of numbers and a link to the next section of the list. Each number is linked to a page where its definition, and decimal and binary representations can be found.

First, we will look at tokenization. Below is the html code for one of the pages.

```
<html><body>
  <p><a href='first.html'>Home</a>
  <h1>One</h1>
  <table border=1>
    <tr>
      <td>Definition</td>
      <td>being a single unit or thing</td>
    </tr>
    <tr><td>Decimal</td><td>1</td></tr>
    <tr><td>Binary</td><td>1</td></tr>
  </table>
  <p>Previous <a href='two.html'>Next</a>
</body></html>
```

We use a simple html-aware lexer. URLs, e.g., `first.html` and text blocks, e.g., `being a single unit or thing`, are represented with single tokens based on the assumption that shorter

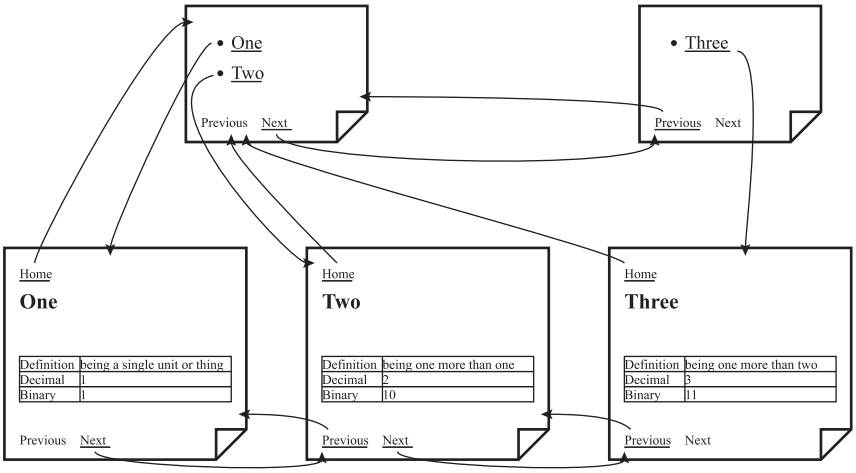


Figure 4: Example Site

tokens would not lead to a better relational representation. So, the above page is represented with the following sequence of tokens:

(`<html, > <body, >, <p, >, <a, href, =, ', first.html, ', >, Home, </a, >, <, h1, >, One, </h1, >, <table, border, =, 1, >, >, <tr, >, <td, >, Definition, </td, >, <td, >, being a single unit or thing, </td, >, ...`)

For notational convenience, we will let p_0 be the token sequence for the home page, p_1 the token sequence for the next page of the list of numbers, and p_2, p_3 and p_4 the token sequences for the item pages of one, two and three respectively.

Next, we define the function l that maps tokens into pages. l describes the connectivity of the site:

t	$l(t)$
first.html	$p0$
second.html	$p1$
one.html	$p2$
two.html	$p3$
three.html	$p4$

The tokens and the link function define the site: $S = (\{p_0, p_1, p_2, p_3, p_4\}, l)$.

Next, we will define a relational model for this site. Others are certainly possible. This one is intuitively one of the better ones. The particular model contains three tables. The first table, t_0 represents the pages that contain the list of numbers. The second table, t_1, represents the lists within those pages. And the third table, t_2, represents the item pages.

In the diagrams below, the contents of the cells in all but the leftmost column are the values of the functions c and r. In some cells, where $c(i, j)$ is a long string, it is abbreviated with an ellipsis. Values of $r(i, j)$ are in italics. The leftmost column represents the function k. Since the tables can get rather wide, we break them apart into multiple lines and indicate the continuation by double borders on the right side of columns to the left and on the left side of columns to the right.

t_0:

0	<html><body>	$(t_1, 0)$	<p>		Previous	
1	<html><body>	$(t_1, 1)$	<p>		Previous	

		Next		</body></html>
		Next		</body></html>

t_1:

0	<a href='	one.html	'>	One	
0	<a href='	two.html	'>	Two	
1	<a href='	three.html	'>	Three	

t_2:

0	<html>...href='first.html'>Home<h1>	One	</h1><table...Definition</td><td>
1	<html>...href='first.html'>Home<h1>	Two	</h1><table...Definition</td><td>
2	<html>...href='first.html'>Home<h1>	Three	</h1><table...Definition</td><td>

being a single unit or thing	</td></tr><tr><td>Decimal</td><td>	1	</td>...ary</td><td>
being...one in number	</td></tr><tr><td>Decimal</td><td>	2	</td>...ary</td><td>
being...two in number	</td></tr><tr><td>Decimal</td><td>	3	</td>...ary</td><td>

1	</td></tr></table><p>				Previous			<a href='	two.html
10	</td></tr></table><p>	<a href='	one.html	'>	Previous			<a href='	three.html
11	</td></tr></table><p>	<a href='	two.html	'>	Previous				

'>	Next		</body></html>
'>	Next		</body></html>
	Next		</body></html>

To complete the relational representation we map the pages into the rows whose token sequences are the same as the pages. The relation representation of S is (t_0, t_1, t_2, p_t) where p_t is:

p	$p_t(p)$
$p0$	$(t_0, 0)$
$p1$	$(t_0, 1)$
$p2$	$(t_2, 0)$
$p3$	$(t_2, 1)$
$p4$	$(t_2, 2)$

As mentioned above, alternative representations are possible. For example, we can represent the tabular data on item pages as a separate table and refer to those rows from the main table. Tables t_2' and t_4 below replace table t_2:

t_2':

0	<html>...href='first.html'>Home<h1>	One	</h1><table border=1>
1	<html>...href='first.html'>Home<h1>	Two	</h1><table border=1>
2	<html>...href='first.html'>Home<h1>	Three	</h1><table border=1>

$(t_4, 0)$	</table><p>				Previous			<a href='	two.html
$(t_4, 1)$	</table><p>	<a href='	one.html	'>	Previous			<a href='	three.html
$(t_4, 2)$	</table><p>	<a href='	two.html	'>	Previous				

'>	Next		</body></html>
'>	Next		</body></html>
	Next		</body></html>

t_4:

0	<tr><td>	Definition	</td><td>	being a single unit or thing	</td></tr>
0	<tr><td>	Decimal	</td><td>	1	</td></tr>
0	<tr><td>	Binary	</td><td>	1	</td></tr>
1	<tr><td>	Definition	</td><td>	being one more than one in number	</td></tr>
1	<tr><td>	Decimal	</td><td>	2	</td></tr>
1	<tr><td>	Binary	</td><td>	10	</td></tr>
2	<tr><td>	Definition	</td><td>	being one more than two in number	</td></tr>
2	<tr><td>	Decimal	</td><td>	3	</td></tr>
2	<tr><td>	Binary	</td><td>	11	</td></tr>

Within the the proposed relational formalism, the ordering of tokens within a page, the grouping of similar structures, and the grouping of related structures are explicitly represented. The first of these is imposed by the rules of the relational representation: the order of tokens is preserved within the relations. The second, grouping of similar structures, is the grouping represented by columns. Book titles go in their own column, author names go in a separate column, etc. Similarly, the third, grouping of related structures, is the grouping represented by rows. The title, author and price of a book all go into the same row.

3.3 Hints

The interface between the relational representation and the substructure experts is the hint language. Each expert expresses the substructure it discovers in the hint language. A key feature of the hint language is that it is simple enough that a relational instantiation can be easily evaluated in how well it satisfies a set of hints.

I will develop the exact details of the hint language as part of my thesis work, but the idea is to represent the various types of substructure as column and row hints. A column hint represents the preference of an expert to have two or more token sequences to be aligned vertically, and a row hint does the same but for horizontal alignment.

3.3.1 Substructures As Hints

- **Template Coverage.** Find the template between each pair of pages. For each slot and stripe in the template, generate a column hint that contains the tokens in the slot or stripe of the first page and the tokens in the slot or stripe of the second page.
- **URL Patterns.** Compare the URLs of each pair of pages. If the two URLs are similar enough, generate a column hint that contains all the tokens of the first page and all the tokens of the second page.
- **Next Links.** For each next (or previous) link, generate column hint that contains all the tokens of the source page (the page containing the link) and all tokens of the target page (the page pointed by the link).
- **Indexed Items.** Find sequences of consecutive integers on pages. For each sequence, generate a column hint that contains the tokens of the integers.
- **Redundant Data.** For each link, find common sequences of tokens between the context of the link and the target page. For each common sequence, generate a row hint that contains the sequence on the source page and the sequence on the target page.
- **Page Layout.** Examine the layout of the page. For tokens that are required to be aligned vertically and in close proximity, generate a column hint. Similarly, for horizontally aligned tokens, generate a row hint.
- **Domain Specific Knowledge.** For example, for each token identified as a state abbreviation, if it is followed by another token identified as a zip abbreviation, generate a row hint. Also, for each set of zip codes on a page, generate a column hint.
- **Links in a List.** When a column contains a set of links, generate a column hint containing all the target pages.

3.4 High-Level Algorithm

The top-level algorithm of the system simply mirrors the proposed architecture (fig. 1).

Input – *site*: set of pages from a web site
Output – relational representation of the pages as defined in section 3.2

hints = ∅
for each *expert*
    hints = *hints* ∪ getHintsFromExpert(*expert*, *site*)
find best relational representation r with respect to *hints*
return r

Finding the best relational representation is the most difficult task. It involves efficiently evaluating a relational representation with respect to a large number of hints and searching through the large space of relational representations. In the next two sections, I discuss these issues in more detail.

3.5 Evaluating Relational Instances

After the substructure experts generate a set of hints, the next step is to find the optimal relational instance. The optimality measure can be defined in several ways.

The simplest measure is the number of hints that are in agreement with the relational model. The details of the hint language will determine the exact definition of agreement, but at a high-level, a hint agrees with the relational instance if the token sequences in the hint are placed in alignment within the instance.

A more powerful approach is to evaluate the relational instances using a probabilistic model. In this approach, we view the relational model as a generator for row and column hints and define a probability distribution of row and column hints given a relational instance. Then, by the maximum likelihood principle, the optimal relation is the one that maximizes the probability of the set of hints.

In contrast to typical applications of the maximum likelihood principle where the observed data is directly modeled by the hypothesis, in the proposed approach the hypothesis models the information output by experts. This indirection allows the output of multiple experts to be unified in a probabilistic framework.

3.6 Search Space

Having defined the range of valid relational instances of a web site and a way to evaluate each relational representation, the optimization task reduces to generating each relation instance, evaluating it with respect to the heuristics and then picking the best candidate among the set. In practice, the huge number of instances for a given site prohibits this generate-and-test approach, and part of my thesis work will be investigating techniques to turn it the optimization problem into a tractable problem.

3.7 Evaluation

The two criteria for evaluating the implemented system are the percentage of web sites covered and the goodness of the relational instantiation for the sites that are covered.

Evaluating the system with respect to coverage involves selecting a random set of web sites and counting the number of sites among that set for which the system outputs a relational instantiation whose goodness is above a predefined threshold. Unfortunately, choosing a random

set of web sites is problematic and does not necessarily give a meaningful coverage ratio, because unlike the concept of a web page, the concept of a web site is only loosely defined (Is Google a web site? Are Amazon Books and Amazon Movie Showtimes one site or two?). So instead of random samples of web-sites, I plan to use existing web site directories, which list a number of other web sites, to generate sample sets. For example, the bookstore directory of Yahoo lists more than 200 sites. Evaluating the system on this set of sites would give a coverage ratio that is specific to bookstore sites, but also one that is easier to interpret (since the concept of a bookstore site is somewhat better defined than that of a general web site). Running this kind of evaluation on a number of different directories (retail, geographical, weather, automotive, health, etc.) from multiple sources will give an overall coverage figure for the system.

To evaluate the output of the system, the output will be compared to a target relational instantiation. The target relational instantiation can be generated automatically from independent sources using the proposed system or other extraction tools, or even from the same web-site with supervised extraction techniques. It can also be manually created. Given the lack of tools that can create a full relational instantiation of a web site or the tedious effort needed to do this manually, I will allow the target relational instantiation to be partial. A partial relational instantiation places only a subset of the tokens into relations. This allows the target to focus on the "interesting" pieces of the data. For example, given a set of pages from a bookstore, the target might only have one relation whose rows represent the books and whose columns represent the attributes (title, author, ISBN, price, etc.), but no other relations to represent other data that might be available on the site (related books, reviews, etc.).

To evaluate the output with respect to the target, I will initially use the following simple approach: Consider the neighboring (column-wise and row-wise) pairs of cells in the target relations. For each such pair, check to see if the pair displays the same column or row arrangement in the output as it does in the target relation. For example, if A is the left-neighbor of B in the target relation, then we expect to find A to the left of B (but not necessarily as a neighbor) in some row of one of the output relations. The recall of the system is given by the ratio of the number of pairs that agree between the target and output to the number of total pairs in the target. [4] Note that even though precision can be defined similarly, it is much less meaningful unless the target relations contain all of the tokens. When the target is only a subset, then the output contains pairs that are not in the target and the precision is inherently low.

The remaining issue is to find target relational instantiations for web-sites. Following is a list of possible techniques:

- **Synthetic.** Create a set of web-sites using the relational model, but that are also representative of typical real-world web sites. Use the system to find the relational model and compare the discovered model to the original model, which becomes the target instantiation.
- **User Evaluation.** Have a human user manually create the target for sample sites.
- **Against Supervised Wrappers.** As an alternative to user evaluation, compare the relational data against the data extracted by wrappers generated by hand or by supervised learning algorithms. Typically only a subset of the data available on a page will be extracted by such wrappers, but the data will be accurate both in content and relational structure.
- **Against Aggregation Sites.** To reduce the number of supervised wrappers, use the data available on aggregation sites. These sites gather data from a number of sites and present it in a uniform format. Some examples are PriceGrabber, a comparison shopping

[4] Because A and B are token sequences, they can possibly be broken into multiple cells or even split across relations in the output instantiation. Initially, I will use the cell that contains all of a given token-sequence as the match, and if no such cell exists, I will assume that the neighboring relation does not hold.

Site	URL
agirlsworlduk	http://www.agirlsworld.co.uk
ajelectronicsuk	http://www.ajelectronics.co.uk/
appliancebargains	http://www.appliancebargains.co.uk/index.php
appliances4u	http://www.appliance4u.co.uk/uk2shop.htm
chocolatestoreuk	http://www.chocolatestore.com/
family123fr	http://www.123famille.com
firstvitality	http://www.1stvitality.co.uk/acatalog/index.htm
flowers800	http://www.0800flowers.com
furniture123fr	http://www.123meuble.com/boutique/liste_rayons.cfm?code_lg=lg_fr
uksurf	http://www.surf.uk.com/index.html
unemilleordifr	http://www.1000ordi.fr/
americangolferuk	http://shop.american-golf-discount-online-golf-shop.co.uk/online_shop/americangolf.html

Table 1: Sites for AutoWrap V1 evaluation

site, CiteSeer, and Google News. These kinds of sites are an easy source of labeled data for other sites. They present extracted data from many different sites in pages that are usually easy to extract from with supervised wrappers. This means that one supervised wrapper for an aggregate site provides data for many different sites.

3.7.1 Evaluation of Existing Tools

To validate the recall metric I described in the previous section, I set up a small test suite to evaluate AutoWrapV1. The test suite contains 44 page-sets (each page-set corresponding to a wrapper) containing a total of more than 700 pages. The page-sets were collected from the sites shown in Table 1.

For each page-set, a manually trained wrapper was available. These wrappers extract only a a few items from each page-set, but they work with 100% recall and precision on the pages in the test-suite. The output of each wrapper is one or more tables. For example, for a page that lists the names of cities in a given state, a typical wrapper will output two tables: One for the names of the states, and another for the names of the cities. The first table will have exactly as many rows as there are pages, and the second one as many rows as the number of all cities listed. The tables output by the wrappers define the target relational instantiation for the evaluation.

Next, I generated the relational instantiations using AutoWrap V1. AutoWrap V1 finds a single template for a given set of pages and then uses the template to extract data, so I ran it on each page-set separately. Note that this is a substantially easier task than what the proposed system will be evaluated on. In that evaluation, the system will be run on all pages from a site, without dividing the pages into page-sets.

To evaluate the output of AutoWrap V1 with respect to the target as defined by the manual wrappers, I used the recall metric I defined earlier. Table 2 summarizes the results. The 'tables' and 'cells' columns contain the number of tables and cells in the target instantiation for each wrapper. The counts for the 'retrieved' and 'relevant' columns are the sum of the counts for the individual tables in the target-set.

The important observation to make is that the distribution of the recall score corresponds to the expected performance of AutoWrap V1, namely that it performs poorly on most page-sets but does very well on a few of them. This is evidence that the proposed recall score, or a variation of it, is suitable for evaluating the relational output of the system that I will develop.

3.7.2 Thesis Statement Validation

To validate my thesis statement, I will run a sequence of experiments where I will evaluate the output of the system over a test suite of web-sites as I vary the number of experts being used. If my statement holds, then I expect to get better results, both in coverage and recall, as more experts are used by the system.

Site	Wrapper	Pages	Tables	Cells	Retrieved	Relevant	Recall
agirlsworlduk	Wrapper1	2	3	126	10	111	0.09
agirlsworlduk	DetailsWrapper	10	1	50	28	85	0.33
agirlsworlduk	ListWrapper	29	2	1369	293	1900	0.15
ajelectronicsuk	DetailsWrapper	28	1	140	135	247	0.55
ajelectronicsuk	CategoryPageWrapper	8	2	90	14	88	0.16
ajelectronicsuk	Wrapper1	1	2	45	0	43	0.00
ajelectronicsuk	NoResultsWrapper	14	1	28	18	24	0.75
ajelectronicsuk	ListWrapper	18	3	657	56	721	0.08
appliancebargains	Wrapper1	1	3	92	41	52	0.79
appliancebargains	DetailsWrapper	44	2	716	404	954	0.42
appliancebargains	ListWrapper	11	2	186	2	92	0.02
appliancebargains	SearchWrapper	6	2	129	126	126	1.00
appliances4u	Wrapper1	2	2	120	0	118	0.00
appliances4u	ListWrapper	15	2	3741	98	3859	0.03
chocolatestoreuk	Wrapper1	1	2	14	6	6	1.00
chocolatestoreuk	CategoriesListWrapper	7	2	135	10	133	0.08
chocolatestoreuk	ProductListWrapper	11	2	198	16	196	0.08
chocolatestoreuk	DetailsWrapper	19	2	169	83	229	0.36
chocolatestoreuk	CategoryPageWrapper	7	1	14	5	19	0.26
family123fr	NextListWrapper	38	3	3152	137	1737	0.08
family123fr	DetailsWrapper	15	1	45	42	72	0.58
family123fr	Wrapper1	1	2	32	15	15	1.00
family123fr	ListWrapper	33	4	3197	139	1706	0.08
firstvitality	EntryWrapper	1	2	15	5	13	0.38
firstvitality	SectionWrapper	5	2	120	56	118	0.47
firstvitality	DetailsWrapper	26	4	1677	116	1909	0.06
flowers800	Wrapper1	1	2	60	3	58	0.05
flowers800	ListWrapper	11	2	379	39	466	0.08
flowers800	DetailsWrapper	10	1	70	19	123	0.15
furniture123fr	DetailsWrapper	15	2	116	55	84	0.65
furniture123fr	Wrapper1	2	2	28	0	13	0.00
furniture123fr	ListWrapper	24	2	526	35	273	0.13
furniture123fr	CategoryWrapper	14	2	96	23	53	0.43
uksurf	DetailsWrapper	15	1	105	30	100	0.30
uksurf	Wrapper1	1	2	27	25	25	1.00
uksurf	ListWrapper	9	2	414	16	412	0.04
unemilleordifr	DetailsWrapper	64	1	192	305	305	1.00
unemilleordifr	Wrapper1	1	2	141	0	139	0.00
unemilleordifr	SubcategoryListingWrapper	20	2	2798	1717	4181	0.41
unemilleordifr	ListWrapper	15	3	2394	1441	3532	0.41
americangolferuk	ListWrapper	90	2	506	14	169	0.08
americangolferuk	EntryPage	1	3	186	6	180	0.03
americangolferuk	DetailsWrapper	20	2	142	4	7	0.57
americangolferuk	NoResultsWrapper	67	1	134	0	193	0.00
Total		733	89	24571	5587	24886	0.22

Table 2: Results of AutoWrap V1 evaluation

4 Contributions

- **Structuring the Web.** The main contribution of my thesis will be a new approach for combining heterogeneous substructures in semi-structured web sites to transform them into relational form. In particular, the approach introduces a relational framework to represent web-sites, a language for defining substructure experts, and an optimization algorithm that allows the output of multiple experts to be combined.

- **Implemented System.** I will also have a fully implemented system capable of handling many sites without any modification and many more with the addition of new experts. With the system, massive amounts of data will be available to applications and research that use only structured data.

- **Probabilistic Constraint Satisfaction.** Another contribution will be generalizing the probabilistic-evidence approach I am taking. Littman's crossword solver, Moore's link analysis tool, and the system I propose to develop use the same technique of combining multiple "experts" by taking the output of the experts as inputs to a probabilistic constraint satisfaction framework. One of my goals is to investigate the properties of this approach in general. In particular, I hope to develop search techniques that are applicable in all instances of the approach.

5 Schedule

Define hint language Refine probabilistic model Develop search algorithms Convert existing experts and build additional ones Begin AutoWrap V3 implementation	Summer 2004
Evaluate system Refine search algorithms Complete AutoWrap V3 implementation	Fall 2004
Write dissertation	Fall 2004 & Spring 2005

6 Related Work

- **Grammar Induction.** Starting from a set of positive and negative examples, grammar induction aims to find a formal grammar or automata for a language that contains the positive examples but not the negative ones. In general, grammar induction is a hard problem.

  The relational web problem can be cast as a limited form of grammar induction where the relational structure is imposed on the space of grammars. For example, the grammar rule for a page is in the form of $page_i \rightarrow cell_1 cell_2 \ldots cell_n$ and each *cell* rule is either $cell_i \rightarrow$ **literal** or $cell_i \rightarrow relation_j$. In this formulation, the problem is to choose the correct rules and instantiate the literals to correct token sequences.

  AutoWrap is similar to Sequitur[14] in that both produce hierarchical representations of their inputs. (In the case of AutoWrap, the representation is also relational.) The representations are not generalized grammars that can generate other inputs, but they still capture the structure of the input. Sequitur relies on two hard constraints to limit its

search space: Pairs of symbols are unique within the right side of grammar rules and every rule is used more than once. AutoWrap generalizes the constraint idea in two ways: First, the constraints, which are representations of different types of substructure, are external to the search algorithm. Second, AutoWrap evaluates the constraints probabilistically and avoids getting stuck with conflicting external constraints.

Another research project that is based on grammar induction and that aims to build wrappers for web pages automatically is RoadRunner[6]. RoadRunner induces grammars from a set of positive examples and restricts the form of the candidate grammars to turn its search space into a tractable one.

The disadvantage of grammar induction approaches to wrapper generation is that the types of grammars that can be induced within reasonable time bounds are not expressive enough to capture the different types of substructure. The expressiveness issue is somewhat similar to representing comment blocks within the grammar of a programming language. Even though the structure of comment blocks is simple, representing them within the grammar is not. AutoWrap works around this by simplifying the form of the "grammar" and using external constraints to take advantage of multiple types of substructure.

- **Relational Model Learning.** The relational learning problem is to find a model that can predict the values in a relation. The model, which can be decision trees, first order logic formulas, Markov models, etc., is built based on a given set of tuples from the relation. Once the model is learned, missing attributes can be predicted based on the values of known attributes. We will look at Rapier and probabilistic relational models (PRMs), two approaches that have been applied to the web domain, in detail.

  Rapier's[2] learning algorithm is based on ideas from inductive logic programming (ILP) research. The extraction pattern language it uses is analogous to the first order logic formulas of ILP in that the patterns are generalizations of the training examples. Rapier uses a specific-to-general search to find patterns and guides its search using a compression-based metric.

  PRMs[7] extend Bayesian networks from modeling flat data sets to modeling richer relational data sets. Like Bayesian networks, PRMs are probabilistic networks that represent the statistical dependencies of attributes within a single table, but in addition to Bayesian networks, the dependencies in PRMs also include attributes from related tables. Once the parameters of a PRM are determined, it assigns a probability to any instantiation of the relational structure.

  In general, relational model learning approaches are difficult to apply to the web wrapping problem, because these approaches assume that their input is from a relational source. In the web wrapping problem, the bulk of observable data is in the form of token sequences. To apply relational model learning approaches, the web sites need to be converted into relational form first.

  One way to do the conversion is to use a meta-model where the relations do not model the relations between the data but between the objects, such as pages and tokens, that represent the data[8]. To capture the sequential relation between tokens, the meta-model has to introduce either some "precedes" relation or indices to label the tokens. In the first case, long range structures, such as between the header and footer of a page, are hard to discover. In the second case, the indices need to be treated specially, because withing the relational model, indices do not carry their usual ordering and closeness meaning.

  A second way is to start with a known relational model for the data[16]. This approach is useful in classifying and clustering pages when the underlying relational model is known, but applying it to new sites requires additional manual modeling work.

In contrast to relational model learning approaches, where the algorithms look for a model that best fits the relational data, AutoWrap searches for a relational representation of the (sequential and linked) data that best represents the multiple types of substructure of the data.

- **Data Mining in Graphs.** Subdue[4] is a system that discovers substructures in structural data represented as graphs. Objects in the system are represented by nodes or small subgraphs, and relations between them by edges. Substructures are subgraphs that occur multiple times in the graph. Subdue builds a dictionary of substructures and replaces the occurrences, which may match only approximately to the substructures, by references to the entries of the dictionary. The process is repeated on the resulting graph and substructures are allowed to contain references to existing substructures. In this way, nested substructures can be discovered.

  The HierachicalTemplate algorithm and the Subdue approach are similar in that starting from instances of a concept, both induce the concept and replace the instances with references to the concept and so are able to discover complex structures. In the case of HierachicalTemplate, the instances are the rows of a list and the concept is the row template. In Subdue, the instances are the subgraphs and the concept is a pattern that matches the subgraphs.

  In some ways, the HierachicalTemplate algorithm is a specialized version of Subdue, because it works on a particular graph, the DOM tree, and looks for a particular kind of substructure. As in the relation learning case, applying graph mining techniques to the web wrapping problem suffers from the fact that the token sequences are not easily put into structured form.

- **Wrapper Induction.** In its simplest form, a wrapper is a function that maps web pages to labels[12]. A label is usually just a substring of the page it is mapped from. The wrapper induction algorithms take as training examples a set of labeled pages and typically find regular-expression like patterns to locate the labels within the pages.

  Extensions to the basic wrapper, such as in Stalker[11], are possible so that the induced wrapper extracts structured data instead of labels. The input to Stalker is the *embedded catalog*, which defines the hierarchical structure of the data in terms of lists, groups and items, and a set of training examples, which are labeled pages. A page is labeled by marking up the data to be extracted on the page and linking it to its role in the embedded catalog. Stalker then learns a set of extraction rules using a covering algorithm and attaches them to the embedded catalog. The embedded catalog together with the attached rules is a wrapper that can extract structured data.

  Wrapper induction systems like Stalker are very useful in reducing the time to generate a wrapper by hand, but they have several shortcomings: First, an induced wrapper typically works on a single type of page. Second, a wrapper makes available only the type of data that it is trained on (e.g., if a wrapper is built to extract zip codes, it will not extract state abbreviations). And most importantly, wrapper induction requires human effort to create training sets.

  In contrast, AutoWrap can wrap a web site without any training sets and will turn most, if not all, data that is on the site into machine processable form. In fact AutoWrap's output can be viewed as the result of building and applying all possible wrappers for a given site.

- **Table Extraction.** Table extraction research focuses on detection and understanding of tables in text and web documents. Detection involves locating the tables on a page, or more generally on a set of pages. Understanding involves segmenting the data into

cells and aligning them correctly. The current techniques either rely on the layout of the document or on the syntax and semantics of its content. Either technique works well on some documents and not others, but combining the two approaches is also possible[10].

AutoWrap is attacking a bigger problem, which includes table extraction as a subproblem (in fact, it could use a table extraction algorithm as one of its experts). Even though the problem is bigger, AutoWrap is potentially at an advantage, because it starts with a richer input that can include multiple samples of a single table structure and other types of substructure linked to the tabular data.

- **Probabilistic Constraint Satisfaction.** AutoWrap takes a similar approach to Proverb[15], a crossword puzzle solver. Proverb has a number of "experts" that given a clue, output their list of best candidate answers together with probabilistic preferences assigned to each candidate. The solution to the puzzle is found by globally optimizing the probability assignment for the particular choice of answers. AutoWrap's multiple substructure experts are analogous to Proverb's experts, and the hints play the role of probabilistic preferences.

References

[1] M. M. Astrahan and others. System r: Relational approach to database management. In J. Mylopoulos and M. L. Brodie, editors, *Readings in Artificial Intelligence and Databases*, pages 560–580. Kaufmann, San Mateo, CA, 1989.

[2] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, pages 6–11, Menlo Park, CA, 1998. AAAI Press.

[3] Michael J. Carey, Jerry Kiernan, Jayavel Shanmugasundaram, Eugene J. Shekita, and Subbu N. Subramanian. XPERANTO: Middleware for publishing object-relational data as XML documents. In *The VLDB Journal*, pages 646–648, 2000.

[4] Diane J. Cook and Lawrence B. Holder. Graph-based data mining. *IEEE Intelligent Systems*, 15(2):32–41, 2000.

[5] Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew K. McCallum, Tom M. Mitchell, Kamal Nigam, and Seán Slattery. Learning to extract symbolic knowledge from the World Wide Web. In *Proceedings of AAAI-98, 15th Conference of the American Association for Artificial Intelligence*, pages 509–516, Madison, US, 1998. AAAI Press, Menlo Park, US.

[6] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 109–118, 2001.

[7] Lise Getoor, Nir Friedman, Daphne Koller, and Benjamin Taskar. Learning probabilistic models of relational structure. In *Proc. 18th International Conf. on Machine Learning*, pages 170–177. Morgan Kaufmann, San Francisco, CA, 2001.

[8] Lise Getoor, Eran Segal, Ben Taskar, and Daphne Koller. Probabilistic models of text and link structure for hypertext classification, 2001. IJCAI Workshop on "Text Learning: Beyond Supervision".

[9] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343, 1975.

[10] M. Hurst. Layout and language: An efficient algorithm for text block detection based on spatial and linguistic evidence. In *Proc. Document Recognition and Retrieval VIII*, pages 56–67, 2001.

[11] Craig A. Knoblock, Kristina Lerman, Steven Minton, and Ion Muslea. *Accurately and reliably extracting data from the web: A machine learning approach*, pages 275–287. Intelligent Exploration of the Web. Springer-Verlag, Berkeley, CA, 2003.

[12] Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1-2):15–68, 2000.

[13] Kristina Lerman, Craig Knoblock, and Steven Minton. Automatic data extraction from lists and tables in web sources, 2001. Automatic Text Extraction and Mining workshop (ATEM-01).

[14] C. Nevill-Manning and I. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, 7:67–82, 1997.

[15] Noam M. Shazeer, Michael L. Littman, and Greg A. Keim. Solving crossword puzzles as probabilistic constraint satisfaction. In *AAAI/IAAI*, pages 156–162, 1999.

[16] Benjamin Taskar, Eran Segal, and Daphne Koller. Probabilistic classification and clustering in relational data. In Bernhard Nebel, editor, *Proceeding of IJCAI-01, 17th International Joint Conference on Artificial Intelligence*, pages 870–878, Seattle, US, 2001.

A Sample Pages

The pages in this appendix are for the discussion in the introduction.

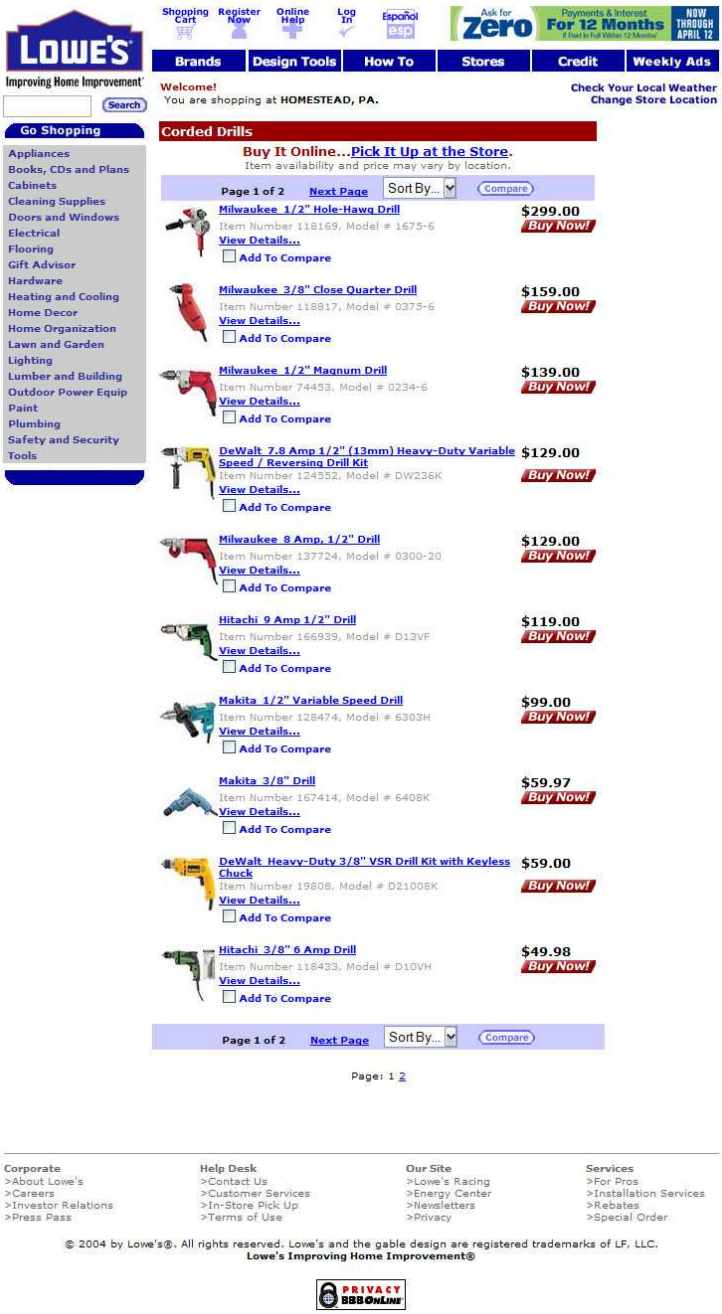


Figure 5: Product List

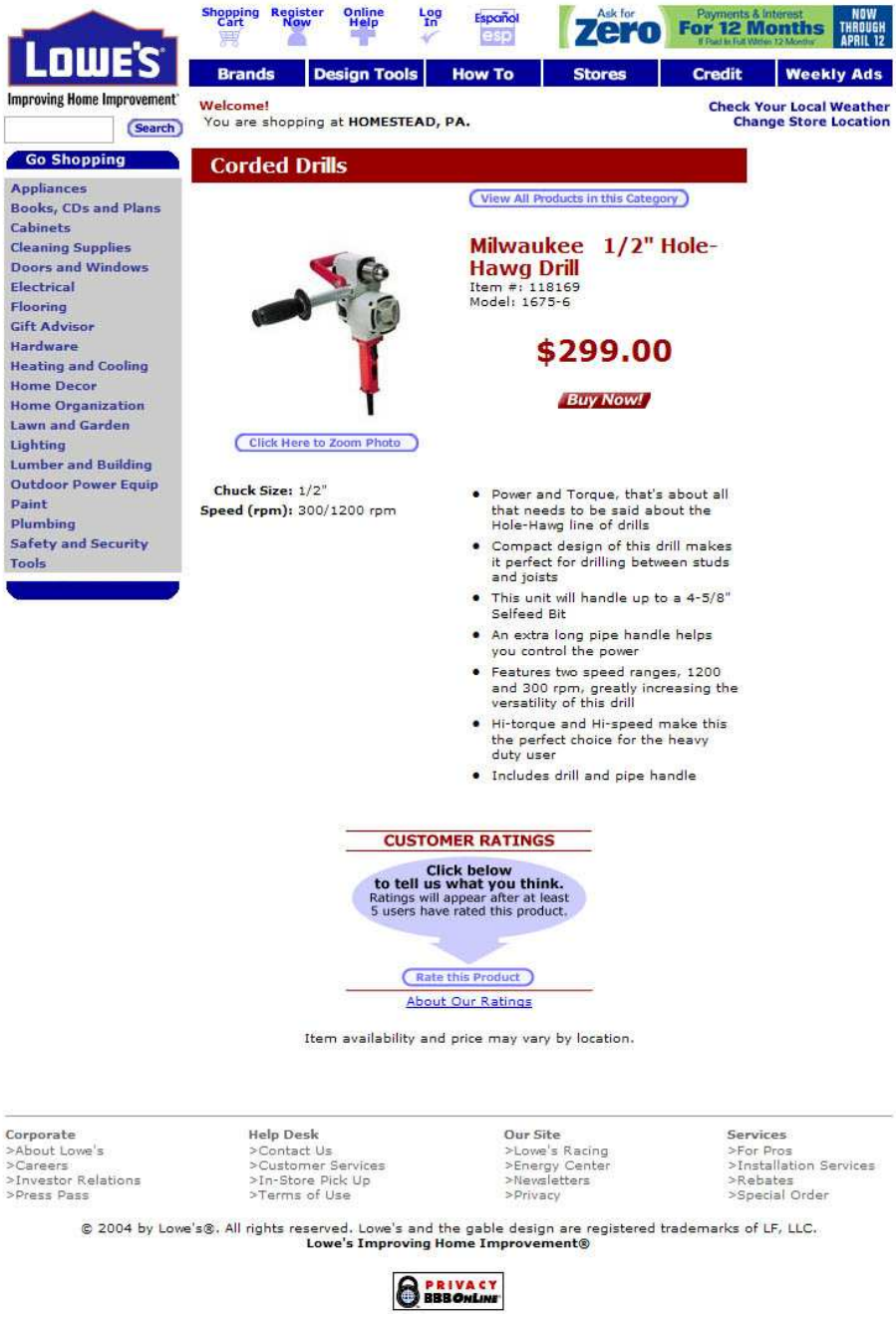


Figure 6: Product Details 1

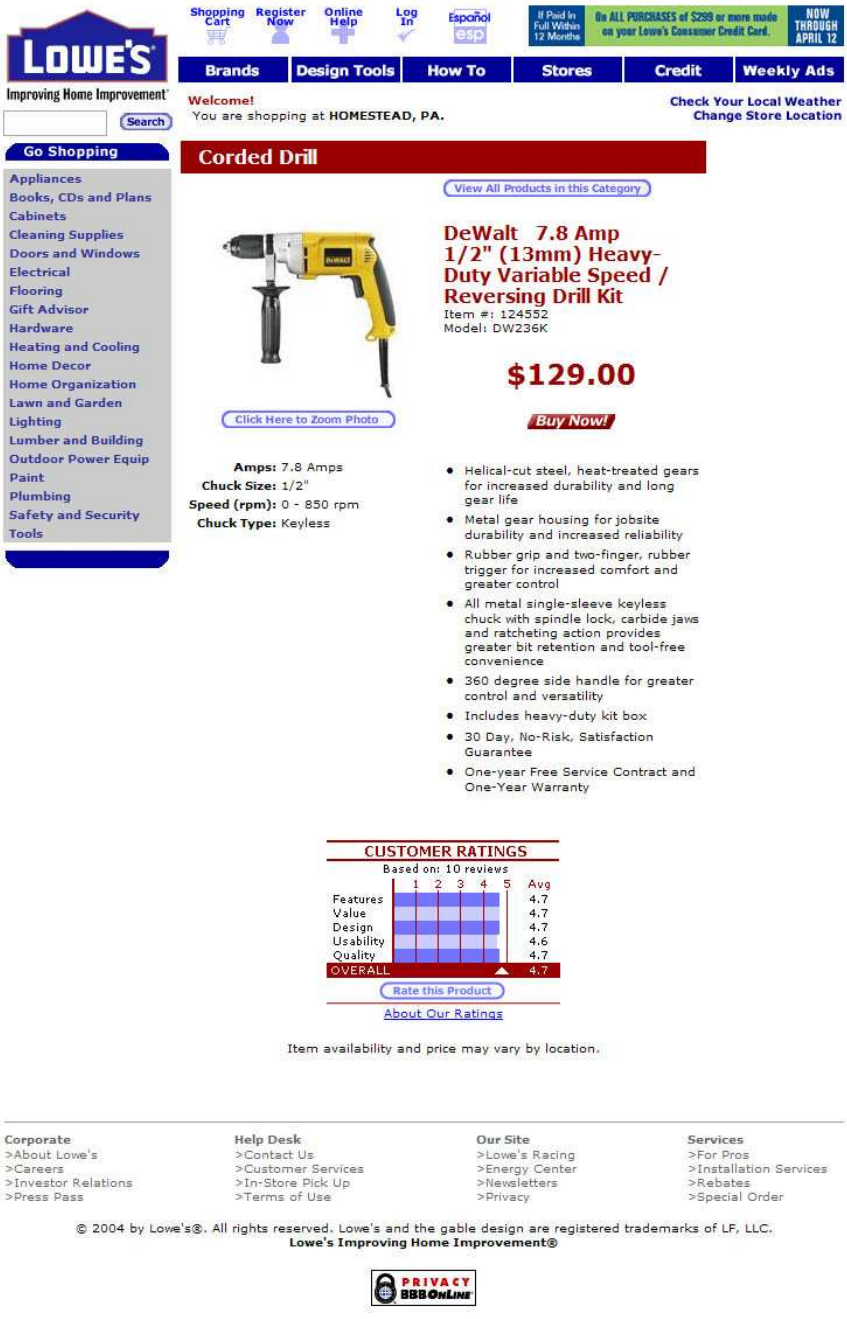


Figure 7: Product Details 2

B AutoWrap V1 Sample

The first two figures are samples of pages collected from the Allegheny County Real Estate Web site. (The figures are the displayed versions of these pages. AutoWrappruns on the html source.) Each page gives detailed information about a particular property. AutoWrap V1 was run on the collection of forty such pages. The next two figures show the extracted data. Some columns were removed from the raw output to make the tables fit on the printed pages, and column labels were added manually to improve readability. In the first table, each row corresponds to a sample page. The attributes column (M) contains cells that refer to the second table. Each reference includes an identifier that selects the related rows of the second table. Note that the second table would have been better represented with only the three columns id, key, and value.

Go to Allegheny County Web site

ALLEGHENY COUNTY
REAL ESTATE WEB SITE

4/9/2004 12:51:23 PM

Search Results | Search Page | Help | Home

OWNER GENERAL INFORMATION
Municipal Code: 828 FOREST HILLS

Block Lot:	0234-M-00076-0000-00	School District:	Woodland Hills
Previous Block Lot:	0234-M-00076-0000-00	Neighborhood Code:	82810
Owner Name:	DURANTI THEODORA R MARIA D BERTA		
Property Location:	238 CASCADE DR PITTSBURGH, PA 15221		
Tax Code:	Taxable	Sale Date:	10/8/2003
Owner code:	Regular	Sale Price:	$1
State Code:	Residential	Deed Book:	11812
Use Code:	Single Family	Deed Page:	588
Homestead:	Yes	Abatement:	No
Farmstead:	No	Lot Area (SQFT):	7,998
		2003 Market Value:	$112,900

County Assessed Value		Full Market Value	
Total Land Value	$18,900	Total Land Value	$18,900
Total Building Value	$79,000	Total Building Value	$94,000
Total Market Value	$97,900	Total Market Value	$112,900

Address Information

Tax Bill Mailing:
DURANTI THEODORA R
1231 BARTOW ST
PITTSBURGH, PA 15205-

Change Notice Mailing:
1231 BARTOW ST
PITTSBURGH, PA 15205-

Legal Disclaimer

Figure 8: Details 1

Go to Allegheny County Web site

ALLEGHENY COUNTY
REAL ESTATE WEB SITE

4/9/2004 12:54:31 PM

Search Results | Search Page | Help | Home

OWNER GENERAL INFORMATION

Municipal Code: 108 PITTSBURGH - 8TH WARD

Block Lot:	0026-M-00170-0000-00	School District:	City Of Pittsburgh
Previous Block Lot:	0026-M-00170-0000-00	Neighborhood Code:	10802
Owner Name:	MILLARD JACQUES A & ROEY VANASKY		
Property Location:	100 MOREWOOD AVE PITTSBURGH, PA 15213		
Tax Code:	Taxable	Sale Date:	7/7/1993
Owner code:	Regular	Sale Price:	$66,800
State Code:	Residential	Deed Book:	9003
Use Code:	Single Family	Deed Page:	14
Homestead:	Yes	Abatement:	No
Farmstead:	No	Lot Area (SQFT):	3,228
		2003 Market Value:	$59,400

County Assessed Value		Full Market Value	
Total Land Value	$12,300	Total Land Value	$12,300
Total Building Value	$32,100	Total Building Value	$47,100
Total Market Value	$44,400	Total Market Value	$59,400

Address Information

Tax Bill Mailing:
VALLEY NATIONAL BANK
1460 VALLEY RD
WAYNE, NJ 07470-

Change Notice Mailing:
100 MOREWOOD AVE
PITTSBURGH, PA 15213-1121

Legal Disclaimer

Figure 9: Details 2

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1		Municipal Code		Block Lot No			School District	Neighorhood	Owner Name			Property Location	Attributes	2003 Market Value	
2	828	FOREST HILLS	234	M	76		Woodland Hills	82810	DURANTI THEODORA R MARIA D BERTA		238	CASCADE DR	<tbl=3 id=0>	112	900
3	828	FOREST HILLS	234	H	158		Woodland Hills	82810	MCCARTY MARY JANE		100	CASCADE RD	<tbl=3 id=1>	96	500
4	828	FOREST HILLS	234	H	156		Woodland Hills	82810	GOOD CHARLES S	DONNA T	102	CASCADE RD	<tbl=3 id=2>	87	500
5	828	FOREST HILLS	234	H	54		Woodland Hills	82810	PERRY JOHN	GLORIA J (W)	103	CASCADE RD	<tbl=3 id=3>	105	500
6	828	FOREST HILLS	234	H	154		Woodland Hills	82810	DATESH JOHN N JR	EDWINA S (W)	104	CASCADE RD	<tbl=3 id=4>	96	000
7	828	FOREST HILLS	234	H	152		Woodland Hills	82810	REARICK JUDITH A		106	CASCADE RD	<tbl=3 id=5>	108	600
8	828	FOREST HILLS	234	H	150		Woodland Hills	82810	GAMBINO MICHAEL J	SALLY A (W)	108	CASCADE RD	<tbl=3 id=6>	138	900
9	828	FOREST HILLS	300	B	60		Woodland Hills	82811	STEINER MIRIAM L		127	CASCADE RD	<tbl=3 id=7>	150	300
10	828	FOREST HILLS	234	H	148		Woodland Hills	82810	RUFFALO JOHN P	DENISE M (W)	220	CASCADE RD	<tbl=3 id=8>	121	500
11	828	FOREST HILLS	234	H	146		Woodland Hills	82810	KING JAMES W	JUDITH A (W)	222	CASCADE RD	<tbl=3 id=9>	115	100
12	828	FOREST HILLS	234	H	144		Woodland Hills	82810	RUSSO ANTHONY M	PATRICIA A	224	CASCADE RD	<tbl=3 id=10>	88	000
13	828	FOREST HILLS	234	M	56		Woodland Hills	82810	MATICH MARK	MARY A (W)	225	CASCADE RD	<tbl=3 id=11>	113	200
14	828	FOREST HILLS	234	H	142		Woodland Hills	82810	STAUFF JAMES M JR	KATHERINE E (W)	226	CASCADE RD	<tbl=3 id=12>	135	000
15	828	FOREST HILLS	234	M	54		Woodland Hills	82810	CHORLE BERNADETTE		227	CASCADE RD	<tbl=3 id=13>	86	600
16	828	FOREST HILLS	234	M	66		Woodland Hills	82810	MCEWEN ROGER G	SUSAN E MC EWEN	228	CASCADE RD	<tbl=3 id=14>	95	100
17	828	FOREST HILLS	234	M	52		Woodland Hills	82810	GUY RICHARD J JR		229	CASCADE RD	<tbl=3 id=15>	90	000
18	828	FOREST HILLS	234	M	68		Woodland Hills	82810	MANSFELD WALTER V	JACQUELINE E (W)	230	CASCADE RD	<tbl=3 id=16>	115	300
19	828	FOREST HILLS	234	M	50		Woodland Hills	82810	RISHEL DOUGLAS M	LINDA L (W)	231	CASCADE RD	<tbl=3 id=17>	145	400
20	828	FOREST HILLS	234	M	70		Woodland Hills	82810	PATALSKI EDWARD J	ELAINE (W)	232	CASCADE RD	<tbl=3 id=18>	105	500
21	828	FOREST HILLS	234	M	72		Woodland Hills	82810	NAGEL GEORGE W	MARY D	234	CASCADE RD	<tbl=3 id=19>	118	400
22	108	PITTSBURGH - 8TH WARD	51	J	108		City Of Pittsburgh	108C3	OLANDER H WARD OLANDER H WARD 50 % SHIRLEY OLANDER		MOREWOOD AVE		<tbl=3 id=20>	1	925 , 600
23	108	PITTSBURGH - 8TH WARD	51	J	177	0001 - 00	City Of Pittsburgh	108C9	PORT AUTHORITY OF ALLEG CO		MOREWOOD AVE		<tbl=3 id=21>	1	400
24	108	PITTSBURGH - 8TH WARD	51	J	178	0001 - 00	City Of Pittsburgh	108C9	PORT AUTHORITY OF ALLEG CO		MOREWOOD AVE		<tbl=3 id=22>	1	500
25	108	PITTSBURGH - 8TH WARD	51	J	181	0001 - 00	City Of Pittsburgh	108C9	PORT AUTHORITY OF ALLEGHENY COUNTY		MOREWOOD AVE		<tbl=3 id=23>	300	
26	108	PITTSBURGH - 8TH WARD	51	J	181	0002 - 00	City Of Pittsburgh	108C9	PORT AUTH OF ALLEG CO		MOREWOOD AVE		<tbl=3 id=24>	5	200
27	108	PITTSBURGH - 8TH WARD	51	J	183	0001 - 00	City Of Pittsburgh	108C9	PORT AUTHORITY OF ALLEGHENY COUNTY		MOREWOOD AVE		<tbl=3 id=25>	4	100
28	108	PITTSBURGH - 8TH WARD	51	J	183	0002 - 00	City Of Pittsburgh	108C9	PORT AUTH OF ALLEG CO		MOREWOOD AVE		<tbl=3 id=26>	2	600
29	108	PITTSBURGH - 8TH WARD	51	J	187	0001 - 00	City Of Pittsburgh	108C9	PORT AUTHORITY OF ALLEGHENY COUNTY		MOREWOOD AVE		<tbl=3 id=27>	16	300
30	108	PITTSBURGH - 8TH WARD	51	J	187	0002 - 00	City Of Pittsburgh	108C9	PORT AUTH OF ALLEG CO		MOREWOOD AVE		<tbl=3 id=28>	1	300
31	108	PITTSBURGH - 8TH WARD	51	J	190	0001 - 00	City Of Pittsburgh	10802	PORT AUTHORITY OF ALLEGHENY COUNTY		MOREWOOD AVE		<tbl=3 id=29>	13	500
32	108	PITTSBURGH - 8TH WARD	51	J	190	0001 - 01	City Of Pittsburgh	108C9	PORT AUTH OF ALLEGHENY CO		MOREWOOD AVE		<tbl=3 id=30>	14	700
33	108	PITTSBURGH - 8TH WARD	51	J	193	0001 - 00	City Of Pittsburgh	108C9	PORT AUTHORITY OF ALLEGHENY COUNTY		MOREWOOD AVE		<tbl=3 id=31>	11	100
34	108	PITTSBURGH - 8TH WARD	26	M	170		City Of Pittsburgh	10802	MILLARD JACQUES A	ROEY VANASKY	100	MOREWOOD AVE	<tbl=3 id=32>	59	400
35	108	PITTSBURGH - 8TH WARD	26	M	204		City Of Pittsburgh	10802	SHEPHERD ' S HEART FELLOWSHIP		103	MOREWOOD AVE	<tbl=3 id=33>	75	400
36	108	PITTSBURGH - 8TH WARD	26	M	202		City Of Pittsburgh	10802	JIANG YUAN		105	MOREWOOD AVE	<tbl=3 id=34>	56	700
37	108	PITTSBURGH - 8TH WARD	26	M	201		City Of Pittsburgh	10802	SHARP KENNETH G		107	MOREWOOD AVE	<tbl=3 id=35>	83	700
38	108	PITTSBURGH - 8TH WARD	26	M	198		City Of Pittsburgh	10802	CHATHA PREVEZ IQBAL	DR IFIKHAR AHMED CHATH	113	MOREWOOD AVE	<tbl=3 id=36>	55	100
39	108	PITTSBURGH - 8TH WARD	26	M	183		City Of Pittsburgh	10802	SHADYOAK PROPERTIES		114	MOREWOOD AVE	<tbl=3 id=37>	53	300
40	108	PITTSBURGH - 8TH WARD	26	M	197		City Of Pittsburgh	10802	LINDQUIST OLGA M		115	MOREWOOD AVE	<tbl=3 id=38>	54	600
41	108	PITTSBURGH - 8TH WARD	26	M	196		City Of Pittsburgh	10802	BARTOLINI NAZZARENO F	DONNA M (W)	117	MOREWOOD AVE	<tbl=3 id=39>	63	900

Figure 10: Extracted Table (0)

	A	B	C	D	E
1	**Id**	**Key**	**Value**	**Key**	**Value**
2	0	Tax Code	Taxable	Sale Date	10/8/2003
3	0	Owner code	Regular	Sale Price	$1
4	0	State Code	Residential	Deed Book	11812
5	0	Use Code	Single Family	Deed Page	588
6	0	Homestead	Yes	Abatement	No
7	0	Farmstead	No	Lot Area (SQFT)	7 , 998
8	1	Tax Code	Taxable	Sale Date	3/3/2004
9	1	Owner code	Regular	Sale Price	$ 121 , 000
10	1	State Code	Residential	Deed Book	11964
11	1	Use Code	Single Family	Deed Page	48
12	1	Homestead	No	Abatement	No
13	1	Farmstead	No	Lot Area (SQFT)	7 , 098
14	2	Tax Code	Taxable	Sale Date	
15	2	Owner code	Regular	Sale Price	$0
16	2	State Code	Residential	Deed Book	0
17	2	Use Code	Single Family	Deed Page	0
18	2	Homestead	Yes	Abatement	No
19	2	Farmstead	No	Lot Area (SQFT)	7 , 500
20	3	Tax Code	Taxable	Sale Date	8/13/1993
21	3	Owner code	Regular	Sale Price	$ 90 , 000
22	3	State Code	Residential	Deed Book	9031
23	3	Use Code	Single Family	Deed Page	543
24	3	Homestead	Yes	Abatement	No
25	3	Farmstead	No	Lot Area (SQFT)	7 , 500
26	4	Tax Code	Taxable	Sale Date	7/18/1983
27	4	Owner code	Regular	Sale Price	$ 81 , 000
28	4	State Code	Residential	Deed Book	6694
29	4	Use Code	Single Family	Deed Page	225
30	4	Homestead	Yes	Abatement	No
31	4	Farmstead	No	Lot Area (SQFT)	7 , 500
32	5	Tax Code	Taxable	Sale Date	10/4/1993
33	5	Owner code	Regular	Sale Price	$ 92 , 000
34	5	State Code	Residential	Deed Book	9068
35	5	Use Code	Single Family	Deed Page	292
36	5	Homestead	Yes	Abatement	No
37	5	Farmstead	No	Lot Area (SQFT)	7 , 500
38	6	Tax Code	Taxable	Sale Date	3/21/1996
39	6	Owner code	Regular	Sale Price	$ 127 , 000
40	6	State Code	Residential	Deed Book	9654
41	6	Use Code	Single Family	Deed Page	54
42	6	Homestead	Yes	Abatement	No
43	6	Farmstead	No	Lot Area (SQFT)	7 , 500

Figure 11: Extracted Table (3)

C AutoWrap V2 Sample

This sample was created by running the system on a small set of pages from Yahoo Sports. The entry page is shown in figure 12. The spidering rule was set to fetch another list page (by chasing one of the "Players by Last Name" links), all the player pages (e.g., fig. 13) linked to the list pages, and a few other pages (e.g., fig. 14). After finding the navigation and detail pages, AutoWrap V2 extracted the data shown in figure 15 from the detail pages. Only the first few columns of this table are shown and columns that contain tokens that are not visible in the displayed pages have been removed.

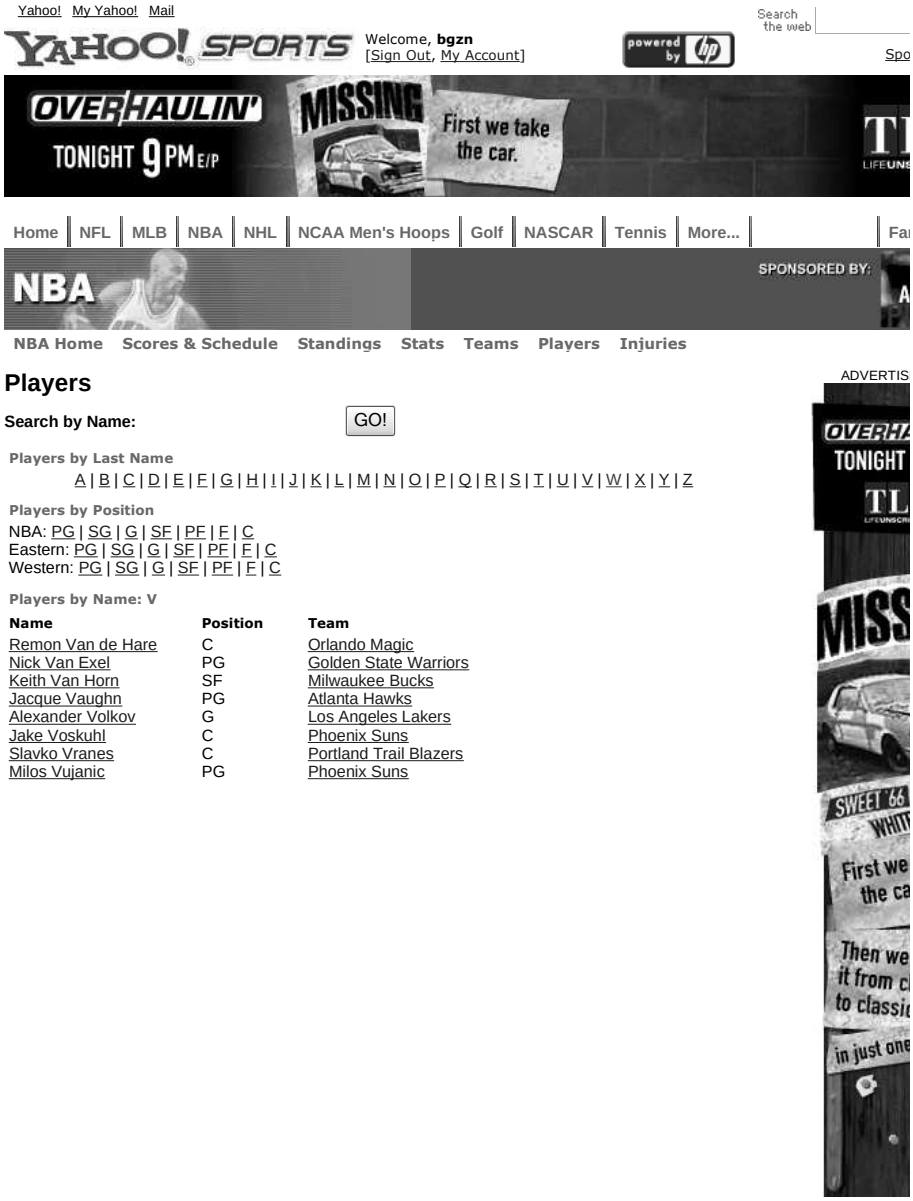
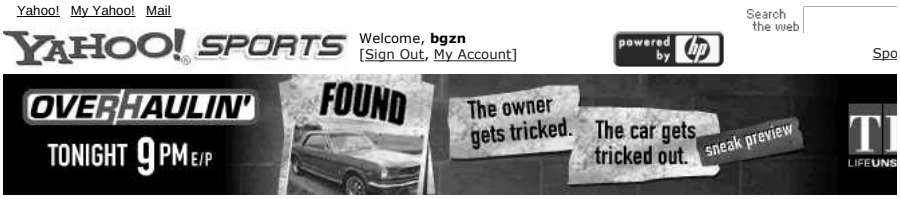


Figure 12: List of Players



Nick Van Exel #37 | **Point Guard** | **Golden State Warriors**

Height: 6-1 Weight: 195 Born: Nov 27, 1971 - Kenosha, Wisconsin College: Cincinnati
Draft: 1993 - 2nd round (37th overall) by the Los Angeles Lakers

Profile | Splits | Career Stats | Game Log | News & Notes | Photos

Injured List as of Mar 1, 2004 (Chronic left knee inflammation)

Recent News

- Musselman could be out despite Golden State's revival Apr 10 (AP)
- Warriors 97, Rockets 90 Apr 7 (AP)
- Warriors 103, Cavaliers 100 Apr 3 (AP)

More News

Guard Comparisons

Scoring | Assists | Steals | 3PT Made

Van Exel
League Average
League Leader

Next Game (4/13 vs. LAL)			FG			3PT			FT			Rebounds						Mi
	G	Min	M	A	Pct	M	A	Pct	M	A	Pct	Off	Def	Tot	Ast	TO	Stl	
Last game vs LAL	0	27	5	10	50.0	0	2	0.0	3	4	75.0	0	2	2	4	2	0	
Career vs. LAL	21	36.1	7.0	17.4	40.2	1.9	5.5	33.9	3.3	4.0	83.3	0.2	3.0	3.2	7.7	2.4	0.5	
Away (this year)	22	29.8	4.3	11.3	38.3	1.1	3.4	32.4	1.2	1.6	75.0	0.5	2.1	2.6	4.8	2.0	0.5	
3+ Days Rest (this year)	7	25.0	4.7	9.4	50.0	0.6	2.9	20.0	2.1	3.6	60.0	0.3	1.3	1.6	4.3	1.7	0.6	

Season Outlook			FG			3PT			FT			Rebounds						Mis
	G	Min	M	A	Pct	M	A	Pct	M	A	Pct	Off	Def	Tot	Ast	TO	Stl	
Last 5 Games	5	28.2	2.6	9.4	27.7	0.6	3.6	16.7	2.0	2.4	83.3	0.4	3.4	3.8	4.2	1.6	0.4	
Feb 21 (SEA)	1	33	2	10	20.0	0	5	0.0	4	6	66.7	0	6	6	5	3	0	
Feb 24 (@ IND)	1	30	4	11	36.4	2	5	40.0	2	2	100.0	0	2	2	3	3	1	
Feb 25 (@ MEM)	1	11	0	1	0.0	0	0	0.0	2	2	100.0	0	0	0	3	1	0	
Feb 27 (@ MIN)	1	32	4	10	40.0	0	3	0.0	2	2	100.0	1	5	6	7	1	1	
Feb 28 (@ CHI)	1	35	3	15	20.0	1	5	20.0	0	0	0.0	1	4	5	3	0	0	
2003 Year to Date	39	32.2	4.8	12.3	39.0	1.2	3.9	30.7	1.8	2.5	70.7	0.4	2.3	2.7	5.3	2.0	0.5	

Recent Career			FG				3PT			FT			Rebounds						Mi
Year	Team	G	Min	M	A	Pct	M	A	Pct	M	A	Pct	Off	Def	Tot	Ast	TO	Stl	
2001	DAL	27	28.0	4.8	11.6	41.1	1.3	3.6	34.7	2.4	2.9	84.4	0.3	2.8	3.1	4.2	1.5	0.5	
2002	DAL	73	27.8	4.7	11.4	41.2	1.6	4.3	37.8	1.5	2.0	76.4	0.5	2.4	2.9	4.3	1.7	0.6	
2003	GS	39	32.2	4.8	12.3	39.0	1.2	3.9	30.7	1.8	2.5	70.7	0.4	2.3	2.7	5.3	2.0	0.5	
Career		**762**	**34.6**	**5.6**	**13.8**	**40.6**	**1.8**	**5.1**	**35.5**	**2.3**	**2.9**	**79.6**	**0.5**	**2.5**	**3.0**	**7.1**	**2.2**	**0.9**	

Last updated through games completed on Apr 12, 2004

Figure 13: Player Page

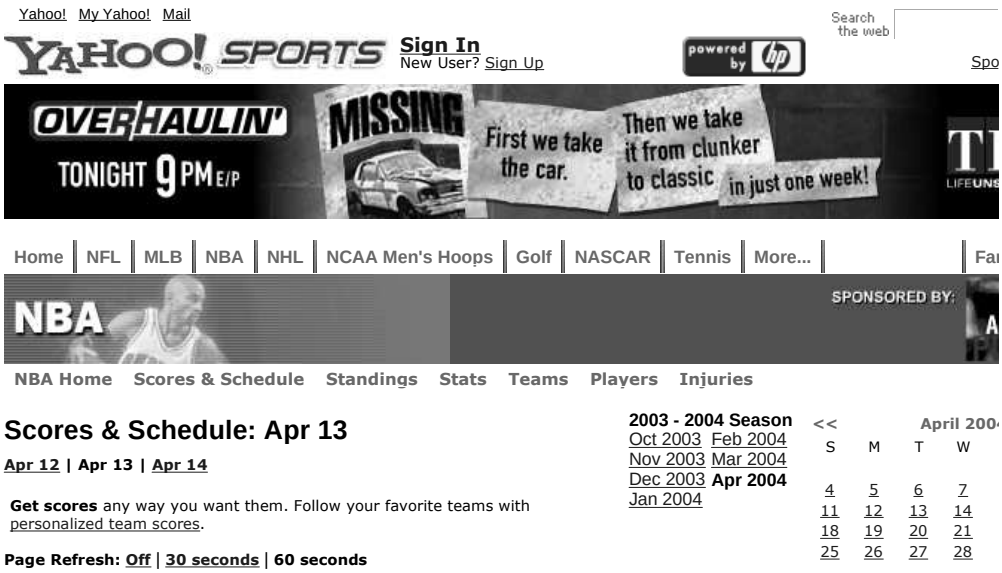


Figure 14: "Other" Page

	A	B	C	D	E	F	G
1	**Name**	**Number | Position**	**Team**	**Height**	**Weight**	**Born**	**College**
2	Milos Vujanic	99 | Point Guard	pho">Phoenix Suns	6-2	190	=- Partizan, Yugoslavia	None
3	Alexander Volkov	6 | Guard	lal">Los Angeles Lakers	6-10	220	Mar 28, 1964 - Omsk, USSR	Kiev Institute
4	Remon Van de Hare	| Center	orl">Orlando Magic	7-2	253	May 23, 1982 - Amsterdam, Holland	None
5	Galen Young	3 | Shooting Guard	sea">Seattle SuperSonics	6-6	230	Oct 16, 1975 - Memphis, Tennessee	Charlotte
6	Nick Van Exel	37 | Point Guard	gsw">Golden State Warriors	6-1	195	Nov 27, 1971 - Kenosha, Wisconsin	Cincinnati
7	Xue Yuyang	| Forward-Center	den">Denver Nuggets	7-1	210	Oct 4, 1982 - Henan, China	None
8	Slavko Vranes	29 | Center	por">Portland Trail Blazers	7-5	275	Jan 30, 1983 - Belgrade, Serbia-Montenegro	None
9	Jake Voskuhl	43 | Center	pho">Phoenix Suns	6-11	245	Nov 1, 1977 - Tulsa, Oklahoma	Connecticut
10	Keith Van Horn	44 | Small Forward	mil">Milwaukee Bucks	6-10	245	Oct 23, 1975 - Fullerton, California	Utah
11	Jacque Vaughn	11 | Point Guard	atl">Atlanta Hawks	6-1	190	Feb 11, 1975 - Los Angeles, California	Kansas

Figure 15: Extracted Data