A Scalable Comparison-Shopping Agent for the World-Wide Web

Robert B. Doorenbos, Oren Etzioni, and Daniel S. Weld

Technical Report UW-CSE-96-01-03

Department of Computer Science and Engineering
University of Washington

Department of Computer Science and Engineering, FR-35
University of Washington, Seattle, WA 98195 USA

Abstract

The Web is less agent-friendly than we might hope. Most information on the Web is presented in loosely structured natural language text with no agent-readable semantics. HTML annotations structure the *display* of Web pages, but provide virtually no insight into their *content*. Thus, the designers of intelligent Web agents need to address the following questions: (1) To what extent can an agent understand information published at Web sites? (2) Is the agent's understanding sufficient to provide genuinely useful assistance to users? (3) Is site-specific hand-coding necessary, or can the agent automatically extract information from unfamiliar Web sites? (4) What aspects of the Web facilitate this competence?

In this paper we investigate these issues with a case study using the ShopBot. ShopBot is a fully-implemented, domain-independent comparison-shopping agent. Given the home pages of several on-line stores, ShopBot autonomously learns how to shop at those vendors. After its learning is complete, ShopBot is able to speedily visit over a dozen software stores and CD vendors, extract product information such as availability and price, and summarize the results for the user. Preliminary studies show that ShopBot enables users to both find superior prices and substantially reduce Web shopping time.

Remarkably, ShopBot achieves this performance without sophisticated natural language processing, and requires only minimal knowledge about different product domains. Instead, ShopBot relies on a combination of heuristic search, pattern matching, and inductive learning techniques.

1 Introduction

In recent years, AI researchers have created several prototype *software agents* that help users with email and netnews filtering [15], Web browsing [4, 13], meeting scheduling [6, 16, 14], and internet-related tasks [7]. Increasingly, the information such agents need to access is available on the World-Wide Web. Unfortunately, the Web is less agent-friendly than we might hope. Although Web pages are written in HTML, this language only defines how information is to be displayed, not what it means. There has been some talk of semantic markup of Web pages, but it is difficult to imagine a semantic markup language that is expressive enough to cover the diversity of information on the Web, yet simple enough to be adopted universally.

Thus, the advent of the Web raises several fundamental questions for the designers of intelligent software agents:

- **Ability:** To what extent can intelligent agents understand information published at Web sites?
- **Utility:** Is an agent's ability great enough to provide substantial added value over a sophisticated Web browser coupled with directories and indices such as Yahoo and Lycos?
- **Scalability:** Existing agents rely on a hand-coded interface to Internet services and Web sites [11, 7, 3, 18, 12]. Is it possible for an agent to approach an unfamiliar Web site and automatically extract information from the site?
- **Environmental Constraint:** What properties of Web sites underlie the agent's competence? Is sophisticated natural language understanding necessary? How much domain-specific knowledge is needed?

While we cannot answer all of the above questions conclusively in a single conference paper, we investigate these issues by means of a case study in the domain of electronic commerce.

This paper introduces ShopBot, a fully implemented comparison-shopping agent.[1] We demonstrate the utility of ShopBot by comparing people's ability to find cheap prices for a suite of computer software products with and without the ShopBot. ShopBot is able to parse product descriptions and identify several product attributes, including price and operating system, for the products. It achieves this performance without sophisticated natural language processing, and requires only minimal knowledge about different product domains. Instead, it extracts information from online vendors via a combination of heuristic search, pattern matching, and inductive learning techniques — with surprising effectiveness. Our experiments demonstrate the generality of ShopBot's architecture both within a domain — we test it on a suite of online software shops — and across domains — we test it on another domain, online CD stores.

The rest of this paper is organized as follows. We begin with a brief description of the online shopping task in Section 2. Section 3 provides a detailed description of the ShopBot prototype

[1] It is publicly accessible at `http://www.cs.washington.edu/homes/bobd/shopbot.html`.

and the principles upon which it is built. In Section 4 we present experiments that demonstrate ShopBot's usefulness and generality. Finally, we discuss related work in Section 5, and conclude with a critique of ShopBot and directions for future work.

2 The Online-Shopping Task

Our long-term goal is to design, implement, and analyze shopping agents that can help users with all aspects of online shopping. The capabilities of a sophisticated shopping assistant would include the following:

1. Help the user decide which product to buy. For example, the user may ask "What new science fiction books are available in paperback?" or "What CD-ROM encyclopedias are available for the Macintosh?"
2. Find product specifications and reviews. For example, if the user indicates they are interested in a special lens for a camera, the agent might point them at the recent discussion of such lenses in the newsgroup `rec.camera` and direct them to specialized Web sites.
3. Make savvy recommendations, taking into consideration recorded user requirements and product dependencies. For example, the shopping assistant might suggest that "Since you are running on an Gateway2000 P5-75, you will need a 16 bit soundblaster card and quad-speed CD-ROM drive in order to install the Myst software you have selected. Shall I search for the appropriate hardware?"
4. Comparison shop. Identify a set of vendors that sell the desired product, and rank them based on appropriate criteria such as price, speed of delivery, *etc.*
5. Monitor "What's new" lists and other sources to discover and analyze new vendors and sources of specifications or reviews.
6. Notice relevant special offers and discounts.

In the remainder of this paper, we discuss our implemented ShopBot prototype. As a first step, we have focused on comparison shopping. While other shopping subtasks remain topics for future work, ShopBot is already demonstrably useful (see Section 4). ShopBot's capabilities (and limitations) form a baseline for future work in this area.

3 ShopBot: A Comparison-Shopping Agent

Our initial research focus has been the design, construction, and evaluation of a scalable comparison-shopping agent called ShopBot. Figure 1 summarizes the problem tackled by ShopBot. Our ShopBot prototype operates in two phases: an offline learning algorithm that creates a vendor description

for each merchant, and a shopping assistant that uses this description to help a person shop in real time.

In the sections below, we describe some important observations that underlie our system, and then summarize the ShopBot architecture and its representation of product domains. Section 3.3 explains the ShopBot's offline learning algorithm, and Section 3.4 details the procedure for comparison shopping.

Given:

- A domain description, including information about product *attributes* useful for discriminating between different products and between variants of the same product (*e.g.*, name, manufacturer, price, *etc.*),
- A set of URL's for the home pages of possible vendors,
- An attribute A (*e.g.*, price) by which the user wants to compare vendors, and
- A specification of the desired product in terms of the values of selected attributes.

Determine: The set of stores where the desired product is available, sorted by A.

For Example: "Find me the cheapest price for the Mac version of Adobe Photoshop."

Figure 1: The Comparison Shopping Problem

3.1 Environmental Regularities

It may seem that construction of a scalable shopping agent is beyond the state of the art in AI, because it requires full-fledged natural language understanding and extensive domain knowledge. While this may be true for the full range of capabilities described in Section 2, we have been able to construct the successful ShopBot prototype by exploiting several environmental regularities that are usually obeyed by online vendors. These regularities are reminiscent in spirit of those identified as crucial to the construction of real-time [1], dynamic [9], and mobile-robotic [2] agents.

1. **Online stores are designed so consumers can find things quickly.** For example, most stores include mechanisms to ensure easy navigation from the store's home page to a particular product description (*e.g.*, a searchable index).
2. **Vendors attempt to create a sense of identity by using a uniform look and feel.** For example, although stores differ widely from each other in their product description formats, any given vendor typically describes all stocked items in a simple consistent format.
3. **Merchants use whitespace to facilitate customer comprehension of their catalogs.** In particular, while different stores use different product description formats, the use of ver-

tical separation is universal. For example, each store starts new product descriptions on a fresh line.

Online vendors obey these regularities because they facilitate sales to human users. Of course, there is no guarantee that what makes a store easy for people to use will make it easy for software agents to master. In practice, though, we were able to design ShopBot to take advantage of these regularities. Our prototype ShopBot makes use of the first regularity by focusing on stores that feature a search form.[2] The second and third regularities allow ShopBot's learning algorithm to incorporate a strong bias, and thus require only a small number of training examples, as we explain below.

3.2 Product-Independent Architecture

ShopBot decomposes the comparison-shopping problem into two phases. The learning phase, described in Section 3.3, analyzes online vendor sites to learn a symbolic description of each site. This phase is moderately computationally expensive, but is performed offline, and needs to be done only once per store.[3] The comparison-shopping phase, described in Section 3.4, uses the learned vendor descriptions to shop at each site and find the best price for a specific product desired by the user. This phase executes very quickly, with network delays dominating ShopBot computation time.

The ShopBot architecture is product-independent — to shop in a new product domain, it simply needs a description of that domain. To date, we have tested ShopBot in the domains of software and CD products. Figure 2 shows the information ShopBot requires about a new product domain. The information falls into three categories: a description of the product attributes, heuristics for understanding vendor pages, and seed knowledge to bootstrap learning. Supplying this information is clearly beyond the capability of the average user. Furthermore, it is difficult if not impossible for an expert to provide the necessary information without some investigation of online vendors in the new product domain. Nevertheless, we were surprised by the relatively small amount of knowledge ShopBot must be given before it is ready to shop in a completely new product domain.

3.3 Creating Vendor Descriptions

The most novel aspect of ShopBot is its learner module, illustrated in Figure 3. The learner automatically generates a vendor description for an unfamiliar online merchant. Together with the domain description, a vendor description contains all the knowledge required by the comparison-shopping phase for finding products at that vendor. Figure 4 shows the information contained in a vendor description. The problem of learning such a vendor description has three components:

- Identifying an appropriate search form,

[2] In future work, we plan to generalize ShopBot to shop at other types of stores.

[3] If a vendor "remodels" the store, providing different searchable indices, or a different search result page format, then this phase must be repeated for that vendor.

- Attributes of products in this domain (*e.g.*, for computer software, we have product name, manufacturer, price, operating system requirements, *etc.*).
- Heuristics for understanding vendor pages: regular expressions for recognizing attributes (or synonyms thereof) on vendor pages.
- Seed knowledge to support learning: descriptions of several popular products in this domain (*e.g.*, "Microsoft Encarta," "Adobe Photoshop") to be used as "test queries" for inductive learning of vendor page formats.

Figure 2: Elements of a ShopBot domain description

- Determining how to fill in the form, and
- Discerning the format of product descriptions from the resulting page.

These components represent three decisions the learner must make. The three decisions are strongly interdependent, of course — *e.g.*, the learner cannot be sure that a certain search form is the appropriate one until it knows it can fill it in and understand the resulting pages. In essence, the ShopBot learner searches through a space of possible decisions, trying to pick the combination that will yield successful comparison shopping.

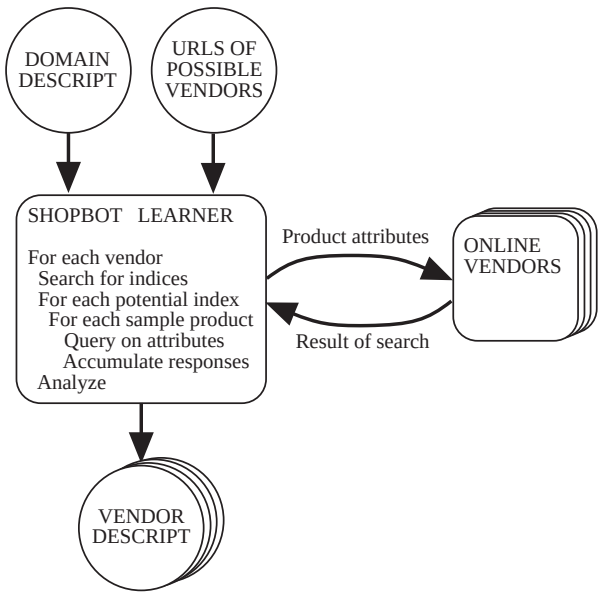

Figure 3: The ShopBot learner's algorithm for creating vendor descriptions.

The learner's basic method is to first find a set of candidate forms — possibilities for the first decision. For each form F_i, it computes an estimate E_i of how successful the comparison-shopping phase would be if form F_i were chosen by the learner. To estimate this, the learner determines how

- The URL of a page containing a form for a searchable index.
- A function mapping product attributes to fields of that form.
- Parsing functions for extracting product data from pages returned by the index:
  - A function that recognizes failure pages (*e.g.*, "Product not found").
  - A function that strips header and trailer information from successful pages.
  - A function that extracts a set of individual product descriptions from the remaining text on a successful page.

Figure 4: A vendor description.

to fill in the form (this is the second decision), and then makes several "test queries" using the form to search for several popular products. The results of these test queries are used for two things. They provide training examples from which the learner induces the format of product descriptions in the result pages from form F_i (this is the third decision). The results of the test queries are also used to compute E_i — the learner's success in finding these popular products provides an estimate of how well the comparison-shopping phase will do for users' desired products. Once estimates have been obtained for all the forms, the learner picks the form with the best estimate, and records a vendor description comprising this form's URL and the corresponding second and third decisions that were made for it.

In the rest of Section 3.3, we elaborate on this procedure. Our emphasis is on the basic techniques rather than the details. We do not claim to have developed an optimal set of heuristics; indeed, the optimal set will change as vendor sites evolve. We go into greatest detail on the third decision — learning result page formats — because we think it is the most novel and most likely to be useful in other Web-related tasks.

3.3.1 Finding and Classifying Forms

The learner begins by finding potential search forms. It starts at the vendor's home page and follows URL links, performing a heuristic search looking for any HTML forms at the vendor's site. (To avoid putting an excessive load on the site, we limit the number of pages the learner is allowed to fetch.) Since most vendors have more than one HTML form, this procedure usually results in multiple candidate forms.

The learner then attempts to identify and discard forms that are obviously *not* searchable indices yielding product information. For example, a form whose fields are labeled as requests for name, address, email, phone, and fax numbers, is probably used for gathering information about customers rather than providing quick access to products for sale. Because this process is heuristic, the learner only discards forms about which it is very confident.

1. Initialize LineDs to the empty set.
2. Induce what header and tail text the vendor uses on each page, by abstracting out references to product attributes and then pattern matching the beginning and end of each page.
3. For each page,
   (a) Remove the header and tail information from the page.
   (b) Partition the page text into *logical lines* such that each logical line starts with an HTML tag that forces a line break.
   (c) For each logical line, create a *line description*. If LineDs doesn't contain this description, add it with its associated counters set to zero. Increment the occurrence counter. If any product attribute appears in the logical line's text, increment the attribute counter. If a price appears in the logical line's text, increment the price counter.
4. Rank the descriptions in LineDs using a weighted sum of the counters, and return the best.

Figure 5: Procedure for unsupervised learning of the product description format, given a set of result pages obtained from the same vendor using the same form.

3.3.2 Analyzing Candidate Forms

At this point, the learner assumes that each remaining form could be a searchable index. It now seeks to identify the form that will yield the best results for the comparison-shopping phase, by making several "test queries" to each form and analyzing the responses. A key insight underlying this algorithm is the decomposition of the learning problem into three subproblems: learning a generalized failure template, learning to strip out irrelevant header and tail information, and learning product description formats. We describe each of these below.

The learner first queries each form with several "dummy" product names such as "`qrsabcdummynosuchprod`" to determine what a "Product Not Found" result page looks like for that form. The learner builds a generalized failure template based on these queries. All the vendors we examined had a simple regular failure response, making this learning subproblem straightforward.

Next, the learner queries the form with several popular products specified in the domain description (Figure 2). Since the domain model typically includes several attributes for each sample product, the learner must choose which attribute to enter in each of the form's fill-in fields. The domain description contains regular expressions encoding synonyms for each attribute; if the regular expression matches the text preceding a field, then the learner associates that attribute with the field. In case of multiple matching regular expressions, the first one listed in the domain description is used. Fields that fail to match any of the regular expressions are left blank. Thus, the learner makes its second decision — how to fill in the form — on the basis of the heuristics provided in the domain description.[4]

[4]We adopted this simple procedure for expedience; it is not an essential part of the ShopBot architecture. We plan

The learner now matches each result page for one of these popular products against the generalized failure template; any page that does not match the template is assumed to represent a successful search. If the majority of the test queries are failures rather than successes, the learner assumes that this is *not* the appropriate search form to use for the vendor.

3.3.3 Identifying Product Information

The learner now uses these pages from successful searches as training examples from which to induce the format of product descriptions in the result pages for this form — this is the third decision the learner must make. Each such page contains one or more product descriptions, each containing information about a particular product (or version of a product) that matched the query parameters. However, extracting these product descriptions turns out to be difficult. The problem is complicated by the ubiquitous presence of irrelevant information (*e.g.*, advertisements, headings, subheadings, and links to other pages). Initially, we thought that product descriptions would be easy to identify because they would always contain the product name, but this is not always the case. Furthermore, the product name often appears in other places on the result page, not only in product descriptions. We also suspected that the presence of a price would serve as a clue to identifying product descriptions, but this intuition also proved false — for some vendors the product description does *not* contain a price, and for others it is necessary to follow a URL link to get the price. In fact, the format of product descriptions varied widely and no simple rule worked robustly across different products and different vendors.

However, the regularities we observed (Section 3.1) suggested a learning approach to the problem. We considered using standard grammar inference algorithms (*e.g.*, [5, 20]) to learn regular expressions that capture product descriptions, but such algorithms require *large* sets of *labeled* example product descriptions — precisely what our ShopBot *lacks*, since it just has a set of pages, and it doesn't (yet) know what pieces of the pages constitute product descriptions. In short, standard grammar inference is inappropriate for our task because it is data intensive and relies on supervised learning. Instead, we adopted an unsupervised learning algorithm that induces what the product descriptions are, given the pages. Our algorithm requires only a handful of training examples, because it employs a very strong bias based on the second and third regularities described in Section 3.1. The algorithm is shown in Figure 5.

Based on the third regularity, the learner assumes that every product description starts on a fresh line, as specified by an HTML tag such as `<p>` or `<li>`. So after filtering out header and tail information, the algorithm breaks the remaining HTML code of each page into *logical lines* representing vertical-space-delimited text.

Based on the second regularity, the learner assumes that at a certain level of abstraction, every product is described in the same format.[5] Each logical line is abstracted into a *line description*

to investigate enabling ShopBot to override the heuristics in cases where they fail.

[5] In fact, the assumption of a uniform format is justified by more than the vendor's desire for a consistent look and feel. Most online merchants store product information in a relational database and use a simple program to create a custom page in answer to customer queries. Since these pages are created by a (deterministic) program, they have a uniform format.

Internet Shopping Network		NECX Direct	
Line Description	Rank	Line Description	Rank
 <a>*text**text*	324	<a>*text* *text* 	402
<a>*text**text*	4	<h4>*text*<a>*text*	21
<h2>*text*<a>*text**text*	3	<a>*text* *text*	12
</h2>*text*	0	<a>*text*	4
 text	0	 	0
		</h4>	0
		text	0

Figure 6: Line descriptions produced when learning the format of Internet Shopping Network (`http://www.internet.net`) and NECX Direct (`http://necxdirect.necx.com`). The highest ranked line descriptions were correctly recognized as encoding product information.

by removing the arguments from HTML tags and replacing all occurrences of intervening freeform text with the variable *text*. For example, a logical line consisting of the source:

```
<li>Click<a href="http://store.com/Encarta">here</a>for the price
of Encarta.
```

would be abstracted into the line description "<li>*text*<a>*text**text*." See Figure 6 for some other examples of line descriptions.

Finally, the learner uses a heuristic ranking process to choose which line description is most likely to be the one the store uses for product descriptions. Our current ranking function is the sum of the number of lines in which some text (not just whitespace) was found, plus the number in which a price was found, plus the number in which one or more of the required attributes were found. This heuristic exploits the fact that since the test queries are for *popular* products, vendors tend to stock multiple versions of each product, leading to an abundance of product descriptions on a successful page. Figure 6 shows the line descriptions produced during the process of learning the product description formats for two software vendors. In both cases, ShopBot correctly picked the line description corresponding to product descriptions. Other vendors have very different product formats, but this algorithm is broadly successful, as we will see in Section 4.

3.3.4 Generating the Vendor Description

The ShopBot learner repeats the procedure just described for each candidate form. The final step is to decide which form is the best one to use for comparison shopping. As mentioned above, this choice is based on making an estimate E_i for each form F_i of how successful the comparison-shopping phase would be if form F_i were chosen by the learner. The E_i used is simply the value of the heuristic ranking function for the winning line description. This function reflects both the number of the popular products that were found and the amount of information present about each

one. The exact details of the heuristic ranking function do not appear to be crucial, since there is typically a large disparity between the rankings of the "right" form and alternative "wrong" forms.

Once the learner has chosen a form, it records a vendor description (Figure 4) for future use by the ShopBot shopper described in the next section. If the learner can't find any form that yields a successful search on a majority of the popular products, then ShopBot abandons this vendor.

The ShopBot learner runs offline, once per merchant. Note that the learner's running time is linear in the number of vendors, the number of forms at a vendor's site, the number of "test queries," and the number of lines on the result pages. The learner typically takes 5–15 minutes per vendor.

3.4 Real-Time Comparison Shopping

While the ShopBot learner is run infrequently, the ShopBot shopper is designed for frequent operation by a wide range of users. Since all learning has been performed offline, the shopper can execute very quickly. Figure 7 summarizes the shopper architecture. As input, the shopper requires both the vendor description (Figure 4) and the domain description (Figure 2). The shopper interacts with the user through a graphical user interface (GUI) that is created from the domain description.

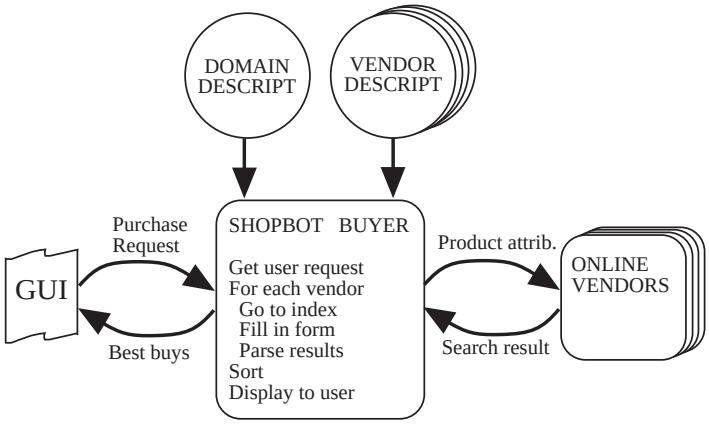

Figure 7: The ShopBot shopper's comparison-shopping algorithm

The operation of the shopper is fairly simple. Once it has received a request from the human via the GUI, it goes in parallel to each online vendor's searchable index, and fills out and submits the forms. For each resulting page not matching the vendor's failure template, it strips off the header and trailer, and looks in the remaining HTML code for any results — any logical lines matching the learned product description format. It then sorts the results (*e.g.*, by ascending order of price), and generates a summary for the user.

Several important principles underlie the ShopBot shopper's user interface:

- **Users are impatient.** Because most of the work has been done by the learner, the ShopBot shopper is very fast; in fact, fetching pages over the network is the bottleneck. The shopper accesses all vendors in parallel, so it is *much* faster than even an expert human.

- **Users desire continual feedback and control.** Norman [17] argues that users should feel in control of their agents, and that it helps if they can build an accurate conceptual model of their agent's activity. The shopper promotes this by providing constant feedback telling the user which vendors are being contacted and what prices have been found so far (see Figure 8). The user can interrupt it at any time.

- **Provide no information without graceful degradation.** The shopper provides the user with enough context around any information it extracts so that the user can verify its conclusions or investigate manually. For the user's convenience, ShopBot indicates the store's home page, the search form it used, and each full product description found.

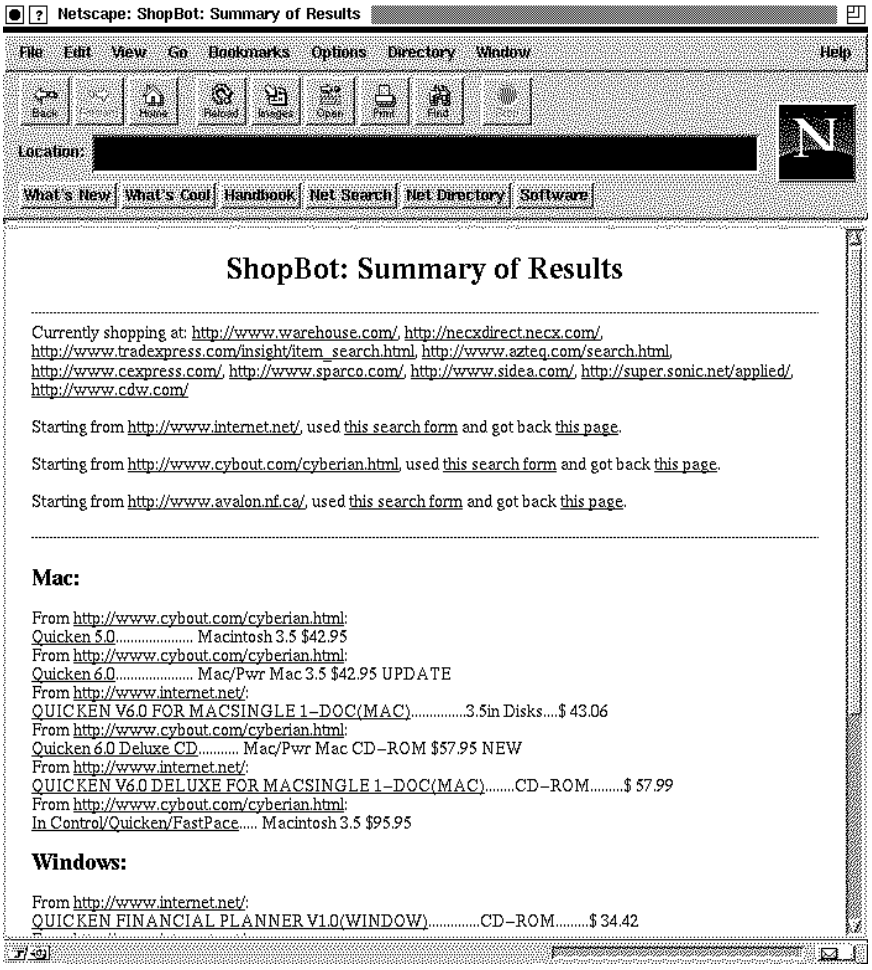

Figure 8: A snapshot of ShopBot shopping for Quicken.

Group	Time (min:sec)	Navigator	eXceed	Word	Quicken
1	13:20	$30.71	$373.06	$282.71	$ 42.95
2	112:30	38.21	(not found)	282.71	41.50
3	58:30	40.95	610.00	294.97	42.95

Table 1: Subjects using the ShopBot performed the task much faster and generally found lower prices.

4 Empirical Results

In this section we consider the overall usefulness of the current ShopBot prototype, the ease with which ShopBot can learn new vendors in the software domain, and its degree of domain independence.

4.1 Evaluating ShopBot Utility

We conjectured that there are two components responsible for the ShopBot's utility. First, the ShopBot shopper acts as a repository of knowledge about the Web: since the user interacts with the shopper after the learning phase has been completed, ShopBot is able to immediately access online vendors and search for the user's desired product. Second, the ShopBot shopper is both methodical and effective at actually finding products at a given vendor; most users are too impatient to perform a manual exhaustive search. For our first experiment, we attempted to measure the usefulness of the current prototype ShopBot and to determine which component was responsible for the utility. We enlisted seven subjects who were novices at electronic shopping, but who did have experience using Netscape. We divided the subjects into three groups:

1. Those who used ShopBot (3 subjects),
2. Those who used Netscape's search tools and were also given the URLs of twelve software stores used by ShopBot (2 subjects), and
3. Those who were limited to Netscape's search tools (2 subjects).

Two independent parties suggested popular software items, yielding descriptions of four products: Netscape Navigatior and Hummingbird eXceed for Windows, and Microsoft Word and Intuit Quicken for the Macintosh. We asked all subjects to try to find the best price for these products and to report how long it took them. Table 1 presents the mean time and prices for each group.[6]

It is perhaps unsurprising that ShopBot users completed their task much faster than the other subjects, but there are several interesting observations to draw from Table 1. First, subjects limited

[6]The trials using subjects in Group 1 were run separately and independently of each other, in order to avoid any positive effects from caching or negative effects from overloading of the ShopBot server.

to Netscape's search methods *never* found a lower price than ShopBot users. Second, although we thought the list of store URLs might make group 2 subjects more effective than ShopBot users, the URLs actually slowed the subjects down. We suspect the tedium of checking stores repeatedly caused group 2 subjects to make mistakes as well. For example, one group 2 subject failed to find a price for eXceed (the other found a low price on an inappropriate version). It seems clear that ShopBot's utility is due to *both* its knowledge and its painstaking search.

4.2 Acquistion of New Software Vendors

To assess the generality of the ShopBot architecture, we asked an independent person not familiar with ShopBot to find online vendors that sell popular software products and that have a search index at their Web site. The independent person found ten such vendors. Although it needs a bit of help at three, ShopBot is able to find products at all of them.[7] ShopBot currently shops at twelve software vendors: the aforementioned ten plus two more we found ourselves and used in the original design of the system. Table 2 shows the prices it found for each of the four test products at each store. (Some of the prices are slightly lower than the users in group 1 found above, because the data in Table 2 was obtained at a later date.) This demonstrates the generality of ShopBot's architecture and learning algorithm within the software domain. The table also shows the variability in both price and availability across vendors, which motivates comparison shopping in the first place.

4.3 Generality Across Product Domains

We have begun to define a new *domain description* that enables ShopBot to shop for pop/rock CD's. We chose the CD domain (first used by the hand-crafted agent BargainFinder [11]) to demonstrate the versatility and scope of ShopBot's architecture. With one day's work on describing the CD domain, we were able to get ShopBot to shop successfully at four CD stores. BargainFinder currently shops successfully at three. (It would shop at three more, but those vendors are blocking out its access.) So with a day's work, we were able to get ShopBot into the same ballpark as a hand-crafted agent in the same domain.

Of course, we do not claim our approach will work with *every* online vendor. In fact, we know of two vendors where its basic techniques would fail currently — one has product descriptions that comprise multiple logical lines, another has product descriptions in varying formats; ShopBot's learning algorithm uses such a strong bias that it cannot correctly learn the formats for these vendors. Nevertheless, the fact that it works on all ten sites found by an independent source strongly suggests that sites where it fails are not abundant.

[7] On three vendors, the system was unable to get to the search form by itself. For one vendor, the only way to get to the search form is to go through an image map; ShopBot cannot understand images. For two other vendors, the search form is at a different Web site — ShopBot searches for HTML forms only at the site where the vendor's home page resides, so if a vendor has pages at two different sites, ShopBot will find only the ones at the home page's site. To get ShopBot to handle these sites, we gave it the search page URL to start from, rather than the home page URL. We also needed to refine some heuristics in ShopBot to get it to handle all ten vendors — again, we do not claim to have identified the perfect set of heuristics.

Home Page URL	Navigator	eXceed	Word	Quicken
http://www.internet.net/	$ 28.57	–	$ 282.71	$ 43.06
http://www.cybout.com/cyberian.html	36.95	–	289.95	42.95
http://necxdirect.necx.com/	31.95	–	329.95	42.95
http://www.sparco.com/	35.00	–	312.00	49.00
http://www.warehouse.com/	39.95	–	fail	fail
http://www.cexpress.com/	?	?	fail	?
http://www.avalon.nf.ca/	44.95	–	–	–
http://www.azteq.com/	?	–	?	?
http://www.cdw.com/	–	–	289.52	fail
http://www.insight.com/web/zdad.html	–	–	315.00	–
http://www.applied-computer.com/twelcome.html	–	$ 349.56	–	43.47
http://www.sidea.com/	59.00	–	–	–

Table 2: Prices found by ShopBot for the four test products at twelve software stores. "–" indicates that ShopBot successfully recognized that the vendor was not selling this product; "?" indicates ShopBot found the product but did not determine the price; fail indicates that ShopBot failed to find the product even though the vendor was selling it.

5 Related Work

We can view related agent work as populating a three-dimensional space where the axes are the agent's task, the extent to which the agent tolerates unstructured information, and whether its interface to external resources is hand-coded. In this section, we contrast ShopBot with related agents along one or more of these dimensions. ShopBot is unique in its ability to *learn* to extract information from the semi-structured text published by Web vendors.

Much of the related agent work requires structured information of the sort found in a relational database (*e.g.*, [12, 3]). The Internet Softbot [7] is also able to extract information from the rigidly formatted output of UNIX commands such as `ls` and Internet services such as `netfind`. There are agents that analyze unstructured Web pages, but they do so only in the context of the assisted browsing task [4, 13], in which the agent attempts to identify promising links by inferring the user's interests from her past browsing behavior. Finally, there have been attempts to process semi-structured information, but again in a very different context than ShopBot. For example, FAQ-Finder [8] relies on the special format of FAQ files to map natural language queries to the appropriate answers.

In contrast with ShopBot, virtually all learning agents (*e.g.*, [15, 14, 6, 10]) learn about their user's interests, instead of learning about the external resources they access. The key exception is the Internet Learning Agent, ILA [18]. ILA learns to understand external information sources by explaining their output in terms of internal categories. ILA learns by querying an information source with familiar objects and analyzing the relationship of output tokens to the query. For example, it queries the University of Washington personnel directory with `Etzioni` and explains

the output token 685-3035 as his phone number.

ShopBot borrows from ILA the idea of learning by querying with familiar objects. However, ShopBot overcomes one of ILA's major limitations. ILA focused exclusively on the problem of category translation and explicitly finessed the problem of locating and extracting relevant information from a Web site — ILA relied on hand-coded "wrappers" to parse the response from each Web site into a small, ordered list of relevant tokens. Thus, ShopBot is solving a *different* learning problem than ILA: instead of trying to interpret each of a list of relevant tokens, ShopBot attempts to identify the relevant tokens and learn the format in which they are presented. ShopBot replaces ILA's hand-coded wrappers with an inductive learning algorithm biased to take advantage of the regularities in Web store fronts.

Along the task dimension, BargainFinder [11] is the closest agent to ShopBot. Indeed, ShopBot's task was inspired by BargainFinder's feasibility demonstration and popularity. However, there are major technical differences between BargainFinder and ShopBot. Whereas BargainFinder is hand-coded for one product domain, ShopBot is product-independent: it takes a description of a product domain as input. Whereas BargainFinder must be hand-tailored for each store it shops at, the only information ShopBot requires about a store is its URL — ShopBot *learns* how to extract information from the store. In short, BargainFinder is *not* an AI program, while ShopBot relies on AI techniques (heuristic search, pattern matching, and inductive learning). Consequently, ShopBot scales to different product domains and is robust to changes in online vendors and their product descriptions.

6 Summary, Critique, and Future Work

Although the Web is an appealing testbed for the designers of intelligent agents, its sheer size, lack of organization, and ubiquitous use of unstructured natural language make it a formidable challenge for these agents. In this paper we presented ShopBot, a fully-implemented comparison-shopping agent that operates on the Web with surprising effectiveness. ShopBot automatically learns how to shop at online vendors and how to extract product descriptions from their Web pages. It achieves this performance without sophisticated natural language processing, and requires only minimal knowledge about different product domains. Instead, it uses heuristic search, pattern matching, and inductive learning techniques which take advantage of regularities at vendor sites. The most important regularity we observed empirically is that vendors structure their store fronts for easy navigation and use a uniform format for product descriptions. Hence, with a modest amount of effort ShopBot can *learn* to shop at a Web store.

The experiments of Section 4 demonstrate that ShopBot is a useful agent which successfully navigates a variety of stores and extracts the relevant information. The first experiment showed that ShopBot provided significant benefit to its users, who were able to find better prices in dramatically less time than subjects without ShopBot. The second and third experiments showed that ShopBot scales to multiple stores and multiple product domains. It shops successfully at all ten software stores found by an independent person. And although it was originally designed for software, a new domain description enabled it to shop for CD's as well, with coverage comparable to that of

BargainFinder, an agent custom built for this domain.

While our experiments have shown that the ShopBot prototype is remarkably successful, they have also revealed a number of limitations. Some of these apply to ShopBot as it stands now, and can probably be fixed with fairly straightforward extensions:

- ShopBot needs to do a more detailed analysis of product descriptions. It does not distinguish between upgrades to a product and the product itself. Because the upgrades tend to be cheaper than the product, they appear higher in ShopBot's sorted list.
- ShopBot relies on a very strong bias, which ought to be weakened somewhat. In particular, ShopBot assumes that product descriptions reside on a single line, and that product description lines outnumber other line types. A more sophisticated learning algorithm would check whether these assumptions are violated, and if so, resort to a more subtle analysis of the vendor's product descriptions.

Other concerns may impact the ShopBot's basic architecture:

- ShopBot is limited to the comparison shopping task, and should be extended to the other tasks described in Section 2.
- ShopBot is limited to stores that provide a searchable index. Some online stores, especially ones with smaller inventories, provide no index, but use a hierarchical organization instead. ShopBot needs to be able to navigate such hierarchies.
- ShopBot is boldly consumer-oriented, which may aggravate vendors who do not want to be comparison-shopped. In fact, several vendors are currently blocking access by BargainFinder. We plan to reimplement the ShopBot shopper in Java, making it harder to distinguish from a person shopping.
- The ShopBot shopper's performance is linear in the number of vendors it accesses (except for the negligible cost of sorting the final results). Once an order of magnitude more merchants populate the Web, it will be important for ShopBot to restrict its search to vendors it considers likely to stock the product at a good price.
- ShopBot relies heavily on HTML. If a vendor provides information exclusively by embedding it in graphics or using Java, ShopBot will be unable to handle the vendor. However, future versions of ShopBot should be able to run Java applets and attempt to analyze their output, just as ShopBot currently does with HTML. We acknowledge that in some cases the output will be too complex or too graphical to permit analysis, but hope that the problem will be lessened by the fact that vendors tend to include "low-technology" formats for the benefit of users on slow network links and users whose browsers are not Java-compliant. Finally, in the more distant future, agents may have sufficient value to users that they will clamor for vendors to provide agent-friendly interfaces to their stores.

All these issues need to be addressed in future research. We also plan additional tests of the ShopBot learner to demonstrate scalability to more domains (*e.g.*, books, consumer electronics,

etc.). Each of these domains consists of products that can be concisely described with a small number of attributes, so it should be feasible to develop domain descriptions for them. We hope to endow ShopBot with more knowledge about the various product domains. For example, we plan to provide ShopBot with rough price expectations for different products. A $1.00 price for Encarta is probably an error of some sort, not a bargain.

We believe that the basic ideas behind the learning algorithm of Section 3.3 are not limited to creating descriptions of product catalogs. We are planning to extend the algorithm to generate "wrappers" (*i.e.*, interface functions) for accessing databases whose contents can be described with relational schemata and whose search forms can be interpreted as relational operations restricted with the use of binding templates [19]. For example, we are generalizing our approach to learn the contents of Web-based Yellow Pages services.

More generally, we conjecture that the vendor regularities that facilitate ShopBot's success are far from unique. ShopBot is a case study suggesting that many Web sites are *semi-structured* and thus amenable to automatic analysis via AI techniques. We anticipate that regularities will be discovered in other classes of Web sites, which will enable intelligent Web agents to thrive. Although the Web is less agent-friendly than we might hope, it is less random than we might fear.

7 Acknowledgements

Thanks to Nick Kushmerick and Marc Friedman for helpful comments on drafts of this paper. Thanks also to Brad Chamberlain, Marc Friedman, Keith Golden, Dylan McNamee, Xiaohan Qin, Jonathan Shakes, Alicen Smith, Geoff Voelker, and Jeanette Yee for their help with the experiments. This research was funded in part by ARPA / Rome Labs grant F30602-95-1-0024, by Office of Naval Research Grant N00014-94-1-0060, by Office of Naval Research grant 92-J-1946, by National Science Foundation Grant IRI-9303461, by National Science Foundation grant IRI-9357772, and by a gift from Rockwell International Palo Alto Research.

References

[1] P. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proc. 6th Nat. Conf. on A.I.*, 1987.

[2] P. Agre and I. Horswill. Cultural support for improvisation. In *Proc. 10th Nat. Conf. on A.I.*, 1992.

[3] Yigal Arens, Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.

[4] Robert Armstrong, Dayne Freitag, Thorsten Joachims, and Tom Mitchell. Webwatcher: A learning apprentice for the world wide web. In *Working Notes of the AAAI Spring Symposium:*

Information Gathering from Heterogeneous, Distributed Environments, pages 6–12, Stanford University, 1995. AAAI Press. To order a copy, contact sss@aaai.org.

[5] R. C. Berwick and S. Pilato. Learning syntax by automata induction. *Machine Learning*, 2:9–38, July 3 1987.

[6] Lisa Dent, Jesus Boticario, John McDermott, Tom Mitchell, and David Zabowski. A personal learning apprentice. In *Proc. 10th Nat. Conf. on A.I.*, pages 96–103, July 1992.

[7] O. Etzioni and D. Weld. A softbot-based interface to the internet. *CACM*, 37(7):72–76, July 1994.

[8] Kristen Hammond, Robin Burke, Charles Martin, and Steven Lytinen. Faq finder: A case-based approach to knowledge navigation. In *Working Notes of the AAAI Spring Symposium: Information Gathering from Heterogeneous, Distributed Environments*, pages 69–73, Stanford University, 1995. AAAI Press. To order a copy, contact sss@aaai.org.

[9] I. Horswill. Analysis of adaptation and environment. *Artificial Intelligence*, 72(1–2), 1995.

[10] Craig Knoblock and Alon Levy, editors. *Working Notes of the AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, Stanford University, 1995. AAAI Press. To order a copy, contact sss@aaai.org.

[11] B. Krulwich. Bargain finder agent prototype. Technical report, Anderson Consulting, 1995. `http://bf.cstar.ac.com/bf/`.

[12] Alon Y. Levy, Divesh Srivastava, and Thomas Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems, Special Issue on Networked Information Discovery and Retrieval*, 5 (2), September 1995.

[13] H. Lieberman. Letizia: An agent that assists web browsing. In *Proc. 15th Int. Joint Conf. on A.I.*, pages 924–929, 1995.

[14] Pattie Maes. Agents that reduce work and information overload. *Comm. of the ACM*, 37(7):31–40, 146, 1994.

[15] Pattie Maes and Robyn Kozierok. Learning interface agents. In *Proceedings of AAAI-93*, 1993.

[16] Tom Mitchell, Rich Caruana, Dayne Freitag, John McDermott, and David Zabowski. Experience with a learning personal assistant. *Comm. of the ACM*, 37(7):81–91, 1994.

[17] D. Norman. How might people interact with agents. *CACM*, 37(7):68–71, July 1994.

[18] Mike Perkowitz and Oren Etzioni. Category translation: Learning to understand information on the internet. In *Proc. 15th Int. Joint Conf. on A.I.*, 1995.

[19] A. Rajaraman, Y. Sagiv, and J. Ullman. Answering queries using templates with binding patterns. In *Principles of Database Systems*, 1995.

[20] J.C. Schlimmer and L.A.Hermens. Software agents: Completing patterns and constructing user interfaces. *Journal of Artificial Intelligence Research*, pages 61–89, 1993.