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

Robert B. Doorenbos, Oren Etzioni, and Daniel S. Weld
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195
{bobd, etzioni, weld}@cs.washington.edu

Abstract

The World-Wide-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 ShopBot, a fully-implemented, domain-independent comparison-shopping agent. Given the home pages of several online stores, ShopBot autonomously learns how to shop at those vendors. After learning, it is able to speedily visit over a dozen software and CD vendors, extract product information, 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.

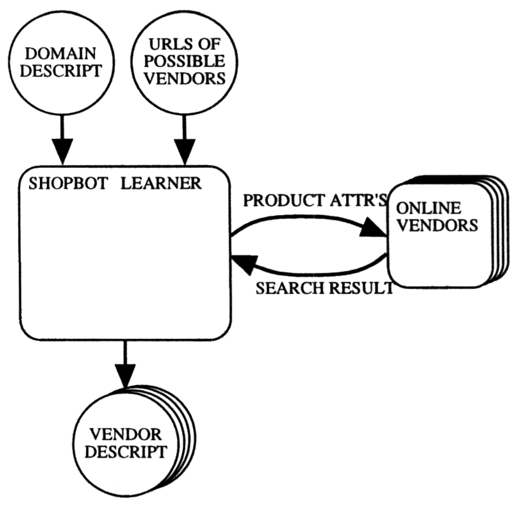
1 Introduction

In recent years, AI researchers have created several prototype *software agents* that help users with email and netnews filtering (Maes & Kozierok 1993), Web browsing (Armstrong *et al.* 1995; Lieberman 1995), meeting scheduling (Dent *et al.* 1992; Mitchell *et al.* 1994; Maes 1994), and internet-related tasks (Etzioni & Weld 1994). 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 universally adopted.

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 (Krulwich 1996; Etzioni & Weld 1994; Arens *et al.* 1993; Perkowitz *et al.* 1996; Levy, Srivastava, & Kirk 1995). 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



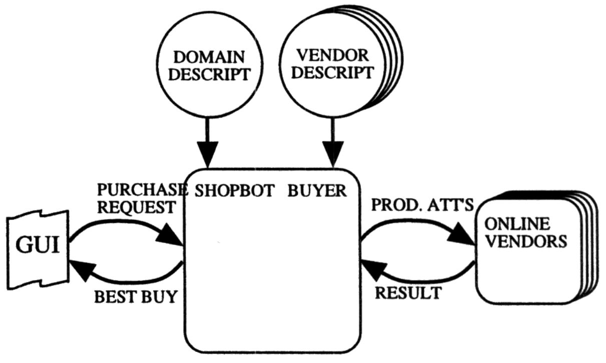


Figure 1: (a) The learner's algorithm for creating vendor descriptions; (b) the shopper's comparison-shopping algorithm.

gation 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.

In the rest of this section, we describe some important observations that underlie our system, then discuss ShopBot's offline learning algorithm and its procedure for comparison shopping. Finally, we give empirical results from our initial prototype ShopBot.

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. However, we have been able to construct a successful ShopBot prototype by exploiting several 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 (Agre & Chapman 1987), dynamic (Horswill 1995), and mobile-robotic (Agre & Horswill 1992) agents.

- **The navigation regularity.** 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.
- **The uniformity regularity.** 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.
- **The vertical separation regularity.** Merchants use whitespace to facilitate customer comprehension of their catalogs. In particular, while different stores use different product description formats, the use of vertical 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 navigation regularity by focusing on stores that feature a search form.[3] The uniformity and vertical separation 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 Creating Vendor Descriptions

The most novel aspect of ShopBot is its learner module, illustrated in Figure 1 (a). Starting with just an online store's home page URL, the learner must figure out how to extract product descriptions from the site. Leaving aside for now the problem of finding the particular web page containing the appropriate product descriptions, the problem of extracting the product descriptions from that page is difficult because such a page typically contains not only one or more product descriptions, but also information about the store itself, meta-information about the shopping process (*e.g.*, "Your search for Encarta matched 3 items" or "Your shopping basket is empty"), headings, subheadings, links to related sites, and advertisements. Initially, we thought that product descriptions would

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

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 music 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 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: 1) helping the user decide what product to buy, *e.g.*, by listing what products of a certain type are available, 2) finding specifications and reviews of them, 3) making recommendations, 4) comparison shopping to find the best price for the desired product, 5) monitoring "What's new" lists and other sources to discover new relevant online information sources, 6) and watching for special offers and discounts.

In the remainder of this paper, we discuss our fully-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. ShopBot operates in two phases: in the learning phase, an offline *learner* creates a vendor description for each merchant; in the comparison-shopping phase, a real-time *shopper* uses these descriptions to help a person decide which store offers the best price for a given product.

The learning phase, illustrated in Figure 1 (a), 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.[2] Table 1 summarizes the problem tackled by the learner for each vendor. The learner's job is essentially to find a procedure for extracting appropriate information from an online vendor.

Given:
1. Incomplete domain model:
   - Example products: P_1, P_2, ..., P_n.
   - Attributes of the products (*e.g.*, `manufacturer`(P_1)=Microsoft, `name`(P_1)=Encarta,...).
2. The URL for the home page of a vendor.

Determine: A procedure which accesses the vendor site to look for a given product and returns a set of strings, each corresponding to a product description returned by the vendor.
Example:
```
MS ENCARTA 1995 - MAC CD-ROM...<a
href=http://www.warehouse.com/> EDU1057...
</a>$89.95
```

Table 1: The Extraction Procedure Learning Problem

The comparison-shopping phase, illustrated in Figure 1 (b), uses the learned vendor descriptions to shop at each site and find the best price for a specific product desired by the user. It simply executes the extraction procedures found by the learner for a variety of vendors and presents the user with a summary of the results. 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 computer software and music CDs. The domain description consists of the information listed in Table 1, plus some domain-specific heuristics used for filling out HTML search forms, as we describe below. Supplying a domain description is beyond the capability of the average user; in fact, it is difficult if not impossible for an expert to provide the necessary information without some investi-

[1]The current version of ShopBot is publicly accessible at `http://www.cs.washington.edu/research/shopbot`.

[2]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.

be easy to identify because they would always contain the product name, but this is not always the case; moreover, the product name often appears in *other* places on the result page, not just 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 above suggested a learning approach to the problem. We considered using standard grammar inference algorithms (*e.g.*, (Berwick & Pilato 1987; Schlimmer & Hermens 1993)) to learn regular expressions that capture product descriptions, but such algorithms require *large* sets of *labeled* example product descriptions — precisely what our ShopBot *lacks* when it encounters a new vendor. We don't want to require a human to look at the vendor's web site and label a set of example product descriptions for the learner. 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 several example pages.

Based on the uniformity regularity, we assume all product descriptions (at a given site) have the same format at a certain level of abstraction. The basic idea of our algorithm is to *search through a space of possible abstract formats and pick the best one.* Our algorithm takes advantage of the vertical separation regularity to greatly reduce the size of the search space. We discuss this in greater detail below.

Overview. 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. Table 2 shows the information contained in a vendor description. The problem of learning such a vendor description has three components:

- Identifying an appropriate search form,
- Determining how to fill in the form, and
- Discerning the format of product descriptions in pages returned from the form.

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.

- The URL of a page containing a form for a searchable index.
- A function mapping product attributes to fields of that form.
- 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 tailer information from successful pages.
  - A function that extracts a set of individual product descriptions from the remaining text on a successful page.

Table 2: A vendor description.

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 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 3.2, we elaborate on this procedure. We do not claim to have developed an optimal procedure; indeed, the optimal one will change as vendor sites evolve. Consequently, our emphasis is on the architecture and basic techniques rather than low-level details.

Finding and Analyzing Candidate 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. Some simple heuristics are used to discard forms that are clearly *not* searchable indices, e.g., forms which prompt the user for "name," "address," and "phone number". Each remaining form is considered potentially to be a searchable index; the final decision of which form the shopper should use is postponed for now.

The learner now turns to its second decision — how to fill in each form. Since the domain model typically includes several attributes for each test product, the learner must choose which attribute to enter in each of the form's fill-in fields. Our current ShopBot does this using a set of domain-specific heuristic rules provided in the domain description.[4] 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.

Identifying Product Description Formats. The learner's third decision — determining the format of product descriptions in pages returned from the form — is the most complex. The algorithm relies on several common properties of the pages typically returned by query engines. (1) For each form, the result pages come in two types: one for "failure" pages, where nothing in the store's database matched the query parameters, and one for "success" pages, where one or more items matched the query parameters. (2) Success pages consist of a header, a body, and a tailer, where the header and tailer are consistent across different pages, and the body contains all the desired product information (and possibly irrelevant information as well). (3) When success pages are viewed at an appropriate level of abstraction, all product descriptions have the same format, and nothing else in the body of the page has that format.[5] Based on these properties, we decompose the learner's third decision into three subproblems: learning a generalized failure template, learning to strip out irrelevant header and tailer information, and learning product description formats.

The learner first queries each form with several "dummy" products 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 given in the domain description. It matches each result page for one of these products against the 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. Otherwise, the learner records generalized templates for the header and tailer of success pages, by abstracting out references to product attributes and then finding the longest matching prefixes and suffixes of the success pages obtained from the test queries.

The learner now uses the bodies of these pages from successful searches as training examples from which to induce the format of product descriptions in the result pages for this form. 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, as discussed above, extracting these product descriptions is difficult, because their format varies widely across vendors.

We use 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 uniformity and vertical separation regularities described in Section 3.1. Based on the uniformity regularity, we assume all product descriptions have the same format at a certain level of abstraction.[6] The algorithm searches through a space of possible abstract formats and picks the best one. Our abstraction language consists of strings of HTML tags and/or the keyword *text*. The abstract form of a fragment of HTML is obtained by removing the arguments from HTML tags and replacing all occurrences of intervening free-form text with *text*. For example, the HTML source:

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

would be abstracted into "<li>*text*<a>*text*</a>*text*."

There are infinitely many abstract formats in this language to consider in our search. Of course, we need only consider the finitely many which actually occur in one of the bodies of the success pages from the test products. While this still leaves us with a very large search space, we can prune the space further. Based on the vertical separation regularity, the learner assumes that every product description starts on a fresh line, as specified by an HTML tag (*e.g.*, <p>, <br>, <li>, *etc.*).

[4]We adopted this simple procedure for expedience; it is not an essential part of the ShopBot architecture. We plan to investigate enabling ShopBot to override the heuristics in cases where they fail.

[5]Property (2) can be made trivially true by taking the header and tailer to be empty and viewing the entire page as the body. However, an *appropriate* choice of header and tailer may be necessary to obtain property (3).

[6]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.

So the algorithm breaks the body of each result page into *logical lines* representing vertical-space-delimited text, and then only considers abstract formats that correspond to at least one of the logical lines in one of the result pages. Thus, instead of being linear in the size of the original hypothesis space, the learning algorithm is linear in the amount of training data, *i.e.*, the number and sizes of result pages.

The bodies of success pages typically contain logical lines with a wide variety of abstract formats, only one of which corresponds to product descriptions. (See (Doorenbos, Etzioni, & Weld 1996) for some examples.) The learner uses a heuristic ranking process to choose which format 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 of that format 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. Different vendors have very different product formats, but this algorithm is broadly successful, as we show in Section 4.

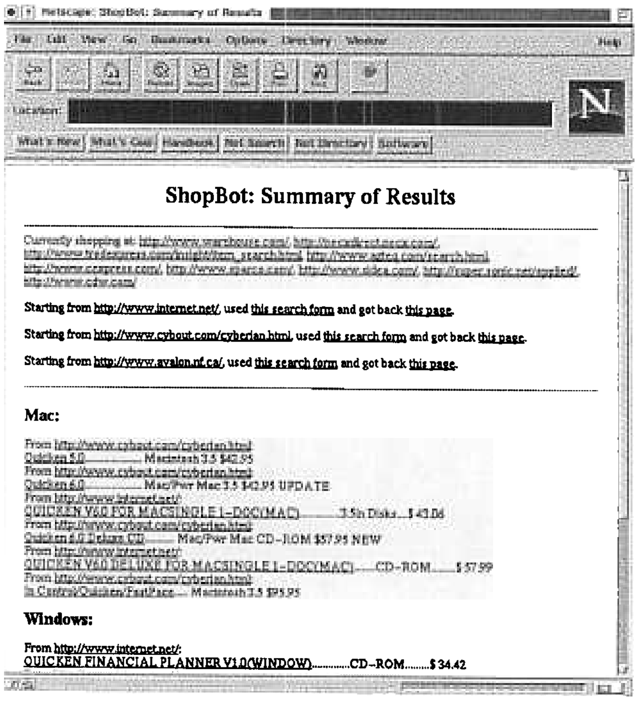
Generating the Vendor Description. The ShopBot learner repeats the procedure just described for each candidate form. The final step is to decide which form is the best one to use for comparison shopping. As mentioned above, this choice is based on making an estimate E_i for each form F_i of how successful 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 abstract format. 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 (Table 2) 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. Its 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.

3.3 Real-Time Comparison Shopping

Learned vendor descriptions are used by the ShopBot shopper to do comparison shopping, as illustrated in Figure 1 (b). The shopper interacts with the user through a graphical user interface (GUI) based on the domain description. The operation of the shopper is fairly simple. Once it has received a request from the user 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 tailer, and looks in the remaining HTML code for any results — any logical lines matching the learned product description format. It then sorts the results by ascending order of price,[7] and generates a summary for the user.


Netscape: ShopBot: Summary of Results
File Edit View Go Bookmarks Options Directory Window Help
Location:
What's New What's Cool Handbook Net Search Net Directory Software

ShopBot: Summary of Results

Currently shopping at: http://www.warehouse.com/, http://necxdirect.necx.com/, [illegible] http://www.[illegible].com/search.html
http://www.cexpress.com/, http://www.[illegible].com/, http://www.[illegible].com/, http://super.sonic.net/[illegible]/
http://www.cdw.com/

Starting from http://www.internet.net/, used this search form and got back this page.

Starting from http://www.cybout.com/cyberian.html, used this search form and got back this page.

Starting from http://www.avalon.nf.ca/, used this search form and got back this page.

Mac:

From http://www.cybout.com/cyberian.html
Quicken 5.0 .... Macintosh 3.5 $42.95
From http://www.cybout.com/cyberian.html
Quicken 6.0 .... Mac/Pwr Mac 3.5 $42.95 UPDATE
From http://www.internet.net/
QUICKEN V6.0 FOR MACSINGLE 1-DOC(MAC) .... 3.5in Disks ... $ 43.06
From http://www.cybout.com/cyberian.html
Quicken 6.0 Deluxe CD ...... Mac/Pwr Mac CD-ROM $57.95 NEW
From http://www.internet.net/
QUICKEN V6.0 DELUXE FOR MACSINGLE 1-DOC(MAC) ..... CD-ROM ..... $ 57.99
From http://www.cybout.com/cyberian.html
In Control/Quicken/FastPace ... Macintosh 3.5 $95.95

Windows:

From http://www.internet.net/:
QUICKEN FINANCIAL PLANNER V1.0(WINDOW)............CD-ROM.........$ 34.42

Figure 2: A snapshot of ShopBot shopping for Quicken.

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

[7]Prices are extracted using special hand-coded techniques.

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 3 presents the mean time and prices for each group.

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 3. First, subjects limited 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 Acquisition 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 subject found ten such vendors, and ShopBot is able to shop at all of them.[8] 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 4 shows the prices it found for each of four test products chosen by independent subjects at each store. (Some of the prices are slightly lower than the users in group 1 found above, because the data in Table 4 was obtained at a later date.[9]) 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 created 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 (Krulwich 1996), 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.[10] So with a day's work, we were able to get ShopBot into the same ballpark as a domain-specific hand-crafted agent.

Of course, we do not claim our approach will work with *every* online vendor. In fact, we know of several vendors where it currently fails, because its learning algorithm uses such a strong bias that it cannot correctly learn their formats. Nevertheless, the fact that it works on all ten software vendors found by an independent source strongly suggests that sites where it fails are not abundant.

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.*, (Levy, Srivastava, & Kirk 1995; Arens *et al.* 1993)). The Internet Softbot (Etzioni & Weld 1994) 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

[8]Of these ten, four were sites we had studied while designing ShopBot, while six were new. ShopBot requires special assistance on one site, where the only way to get to the search form is to go through an image map; since ShopBot cannot understand images, we gave it the URL of the search form page instead of the URL of the home page.

[9]One subject in group 2 managed to find a lower price for Quicken than ShopBot, by going to a web site ShopBot didn't know about.

[10]In our informal experiments, BargainFinder tried to shop at eight stores. Three of them blocked out its access. It was successful at three others. At the remaining two, it tried but failed to find products. We were able to find products at these two stores "by hand," so BargainFinder's failure may be the result of buggy or out-of-date hand-crafted code written just for these two stores (which motivates our use of a learning algorithm in the first place).

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

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		–	–
`http://www.cexpress.com/`	?	?	–	?
`http://www.avalon.nf.ca/`	44.95			
`http://www.azteq.com/`	?		?	?
`http://www.cdw.com/`			289.52	–
`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 4: Prices found by ShopBot for the four test products at twelve software stores. A space left blank indicates that ShopBot successfully recognized that the vendor was not selling this product; "?" indicates ShopBot found the product but did not determine the price; "–" indicates that ShopBot failed to find the product even though the vendor was selling it.

only in the context of the assisted browsing task (Armstrong *et al.* 1995; Lieberman 1995), 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 (Hammond *et al.* 1995) relies on the special format of FAQ files to map natural language queries to the appropriate answers.

In contrast with ShopBot, virtually all learning software agents (*e.g.*, (Maes & Kozierok 1993; Maes 1994; Dent *et al.* 1992; Knoblock & Levy 1995)) learn about their user's interests, instead of learning about the external resources they access. The key exception is the Internet Learning Agent, ILA (Perkowitz *et al.* 1996). 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 (Krulwich 1996) 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 must be hand-tailored for each store it shops at, the only information ShopBot requires about a store is its URL — ShopBot uses AI techniques to *learn* how to extract information from the store.

6 Summary, Critique, & 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 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.
- 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 3.2 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 (Rajaraman, Sagiv, & Ullman 1995; Kwok & Weld 1996). 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

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

Agre, P., and Horswill, I. 1992. Cultural support for improvisation. In *Proc. 10th Nat. Conf. on AI*, 363–368.

Arens, Y., Chee, C. Y., Hsu, C.-N., and Knoblock, C. A. 1993. Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems* 2(2):127–158.

Armstrong, R., Freitag, D., Joachims, T., and Mitchell, T. 1995. Webwatcher: A learning apprentice for the world wide web. In *Working Notes of the AAAI Spring Symposium: Information Gathering from Heterogeneous, Distributed Environments*, 6–12. Stanford University: AAAI Press. To order a copy, contact sss@aaai.org.

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

Dent, L., Boticario, J., McDermott, J., Mitchell, T., and Zabowski, D. 1992. A personal learning apprentice. In *Proc. 10th Nat. Conf. on AI*, 96–103.

Doorenbos, R. B., Etzioni, O., and Weld, D. S. 1996. A scalable comparison-shopping agent for the world-wide web. Technical Report 96-01-03, University of Washington, Department of Computer Science and Engineering. Available via FTP from `pub/ai/` at `ftp.cs.washington.edu`.

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

Hammond, K., Burke, R., Martin, C., and Lytinen, S. 1995. FAQ finder: A case-based approach to knowledge navigation. In *Working Notes of the AAAI Spring Symposium: Information Gathering from Heterogeneous, Distributed Environments*, 69–73. Stanford University: AAAI Press. To order a copy, contact sss@aaai.org.

Horswill, I. 1995. Analysis of adaptation and environment. *Artificial Intelligence* 73(1–2):1–30.

Knoblock, C., and Levy, A., eds. 1995. *Working Notes of the AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments.* Stanford University: AAAI Press. To order a copy, contact sss@aaai.org.

Krulwich, B. 1996. The bargainfinder agent: Comparison price shopping on the internet. In Williams, J., ed., *Bots and Other Internet Beasties.* SAMS.NET. `http://bf.cstar.ac.com/bf/`.

Kwok, C., and Weld, D. 1996. Planning to gather information. In *Proc. 14th Nat. Conf. on AI.*

Levy, A. Y., Srivastava, D., and Kirk, T. 1995. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems, Special Issue on Networked Information Discovery and Retrieval* 5 (2).

Lieberman, H. 1995. Letizia: An agent that assists web browsing. In *Proc. 15th Int. Joint Conf. on AI*, 924–929.

Maes, P., and Kozierok, R. 1993. Learning interface agents. In *Proceedings of AAAI-93.*

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

Mitchell, T., Caruana, R., Freitag, D., McDermott, J., and Zabowski, D. 1994. Experience with a learning personal assistant. *Comm. of the ACM* 37(7):81–91.

Perkowitz, M., Doorenbos, R., Etzioni, O., and Weld, D. 1996. Learning to understand information on the internet: An example-based approach. *Journal of Intelligent Information Systems.* (to appear).

Rajaraman, A., Sagiv, Y., and Ullman, J. 1995. Answering queries using templates with binding patterns. In *Proceedings of the ACM Symposium on Principles of Database Systems.*

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