

4. EXTRACTING DATA

Just as Exhibit has wrapped up much complexity in the conventional three-tier web application architecture in order to making data publishing easy enough for casual users, to bring web data extraction technologies to casual users we must also wrap them up in a ready-to-use package.

- First, some set of commonly needed features such as sorting and filtering can be provided out-of-the-box.
- Second, to keep the user’s visual context from before to after extraction, and to take advantage of already well designed visual elements from the original web page, the presentation elements in the original page are reused as much as possible. In particular, when a data record is extracted from a part of the original page, that fragment of the page is also extracted so that in order to show that data record later on, we can simply show the fragment again. The rest of the web page, which does not contain data to extract, is kept as-is.
- Finally, direct manipulation “handles” can be inserted right into the web page, next to each data value, so that the user can essentially grab a data value and interact with it, such as invoking a sorting or filtering command. This application of direct manipulation lets us do away with extracting field labels or with requiring the user to label the fields herself.

These ideas have been built into a browser extension called Sifter that can augment a sequence of web pages (typically search result pages) *in-place* with filtering and sorting functionality while requiring from the user as few as two clicks. The remainder of this chapter first describes the user interface of Sifter from the extraction phase to the augmentation phase. Next, the data extraction algorithm is explained. Finally, evaluations of the algorithm and the user interface are reported and discussed.

4.1 User Interface Design

User interaction with Sifter consists of two stages:

- extraction, when the system ascertains which parts of the web site to extract, and gives the user feedback about this process; and
- augmentation, when the system adds new controls to the web page and the browser that allow the user to filter and sort the extracted items in-place.

4.1.1 Extraction User Interface

Sifter's user interface resides within a pane docked to the right side of the web browser (Figure 4.1). When the user first visits a web site, the Sifter pane shows a single button that, when clicked, triggers the detection of items on the current web page as well as links to subsequent pages, if any. An item is, for example, a product as in Figure 4.1. Items are highlighted in-place, and the total number of items spanning the detected series of pages is displayed prominently in the pane. If the system has incorrectly detected the items or the subsequent-page links, the user can correct it by clicking on an example item or a subsequent-page link in the web page. Once the **Continue** button (Figure 4.1) is clicked, the extraction process starts and Sifter pops up a dialog box showing the subsequent web pages being downloaded. Over all, getting the data extracted usually takes two button clicks.

During the extraction process, Sifter locates all items on all the web pages, extracts field values from each item as well as its HTML code, and stores each item as a record in a local database. For example, the local database accumulated from extracting the 7 pages of 59 items in Figure 4.1 would contain 59 records, each having one text field (title), two numeric fields (price and percent saving), and an HTML code fragment representing the entire item. This code fragment is used as a rendering of the item when the result set is filtered or sorted.

4.1.1.1 Making Corrections

If the items are detected incorrectly, the user can click on the **Locate Items** button (Figure 4.1) and the Sifter pane will change to highlighting mode (Figure 4.2). In this mode, as the user moves the mouse cursor over the web page, the system inspects the smallest HTML element under the mouse cursor, generalizes it to other similar elements on the page, expands those elements to form whole items, and highlights these candidate items with translucent overlays. When the user is satisfied with the highlighting, she can click the mouse button and the Sifter pane switches out of its highlighting mode.

There was consideration to eliminate the extraction UI altogether and provide a correction UI after the extraction process has finished. However, as the extraction process may be lengthy and not completely reliable, providing a preview of what the system is going to do makes the wait more acceptable and gives the user a sense of greater control over the system.

1 A single button click triggers the detection of items and pages.

2 Items are detected automatically and then highlighted by translucent overlays. Their indices and count are shown.

3 Subsequent pages are automatically detected. The number of pages is used to estimate the total number of items.

4 A second button click starts the extraction process.

Figure 4.1. Sifter provides a one-click interface for triggering the detection of items and pages. One more button click starts the extraction process.

4. EXTRACTING DATA

Candidate items are highlighted and their indices and count are shown as watermarks.

Position of mouse pointer helps identify items to be extracted.

The whole Web page is shown in miniature for an overview of the highlighted items, bringing attention to misses evident as whitespace gaps, if any.

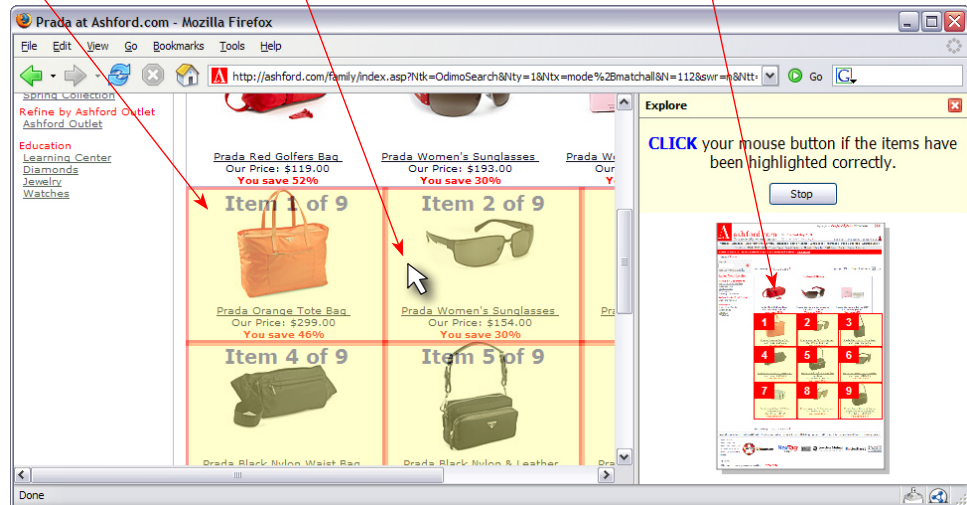


Figure 4.2. Sifter’s highlighting mode provides an interactive mechanism for the user to correct the automatic detection of items by hovering the mouse pointer over the web page and clicking once the items have been highlighted correctly.

4.1.2 Augmentation User Interface

Figure 4.4 shows Sifter in action as the user makes use of new filtering and sorting functionality. An asterisk is inserted after each field value in the web page. When an asterisk is clicked, a *browsing control box* is displayed in the Sifter pane, containing the filtering and sorting controls for that field. Hovering the mouse pointer over either a field’s asterisks or its browsing control box highlights both synchronously, so that the user can tell which browsing control box corresponds to which asterisk, and hence, which field. By keeping the original presentations of the items, Sifter can provide affordance for applying direct manipulation techniques on the field values without ever knowing the field labels.

As the user invokes filtering and sorting commands, Sifter dynamically rewires the web page to show the set of items satisfying the current filters in the current sorting order, as if the web site itself had performed the filtering and sorting operations. Sifter does so by removing the HTML fragment of each item on the page and then injecting into the same slots (where those removed fragments previously fit) the HTML fragments of the items satisfying the current dynamic query. These HTML fragments are retrieved from the local database, so there is no need to make a request to the original site when the dynamic query changes.

Items satisfying the current dynamic query are inserted into the original Web page as if the Web site itself has performed the query.

Extraneous content (e.g., sponsor links) is faded away to avoid confusion and distraction.

Sorting controls for a field

Paging controls for the whole collection.

An asterisk is inserted after each field value. Clicking on an asterisk adds a browsing control box to the Sifter pane. Corresponding asterisks and boxes are co-highlighted when hovered.

Figure 4.4. After extraction is complete, the Sifter pane hosts filtering and sorting controls, which when invoked, re-render the resulting items inside the same web page (without needing to contact the original web server again).

Since exploring the collection of items may involve clicking a link to view details about an item, Sifter stores the query state and automatically restores it when the user returns to the augmented page.

4.1.2.1 Filtering

The filtering controls for different field types (text, numbers, date/time) manage the field values differently. For numbers and date/time fields the values are classified into ranges and sub-ranges hierarchically, while for text fields the values are listed individually. Selecting a value or a range filters the current collection of items

down to only those having that value or having values in that range. Multi-selection in a single field adds disjunctive query terms. Filtering on more than one field forms conjunctive queries. Selecting and de-selecting field values or ranges in a browsing control box updates the available values and ranges in other boxes as in any dynamic query interface [66].

Note that filtering in Sifter is equivalent to faceted browsing in Exhibit (the different terminologies are an artifact of the evolution of my research). Whereas in Exhibit each facet has a label (taken from the corresponding property's label or assigned explicitly by the publisher), in Sifter none of the browsing control boxes has label. Furthermore, in Exhibit, a facet's class (list or numeric range) and placement are determined by the publisher who knows the data and is interested in making her exhibit look and behave however she feels appropriate. In contrast, a Sifter user does not know a web page's data as well and is only interested in sorting or filtering the items therein. For that reason, Sifter takes a more utility-focused than creativity-focused approach: the user is not given control over how the browsing control boxes look and their classes (list, numeric range, or date range) are chosen automatically as much as possible.

4.1.2.2 *Visual Context*

While the user is using Sifter's filtering and sorting functionality, the rest of the original web page is kept for three reasons:

- To maintain visual context so that the user does not have to orient herself in a completely new view of the same information she was seeing before invoking Sifter.
- To avoid throwing away vital information that helps the user understand the items. For instance, the fact that the sale prices are only applicable to students might not be embedded in every item's presentation but is isolated in another part of the page. It would be hard to extract that piece of information.
- To make features offered by the original page still accessible to the user.

However, while preserved, the rest of the page is faded out to visually indicate a disconnection between the features in the original page and the items under Sifter's control. The original status indicators (e.g., number of items, number of pages) are faded to imply that they no longer apply to the items inside the web page. The original pagination, sorting, and browsing controls are faded to imply that invoking them would switch out of Sifter's augmentation mode and let the user interact with the original web site.

4.2 Data Extraction

Given the user interface design described above that requires minimal user intervention during the extraction phase, this section explains how the data extraction algorithm is designed to support that interface. The extraction algorithm consists of three steps: locating items to extract; identifying subsequent web pages; and parsing useful field values from each item.

4.2.1 Item Detection

Let us posit that for many sequences of web pages containing lists of items (e.g., search results, product listings), there exists an XPath [38] that can precisely address the set of items on each page. Thus, our item detection algorithm involves deriving such an XPath. (The cases where each item consists of sibling or cousin nodes [77] will be addressed in future work.) Let us also posit that this XPath can be computed just from the sample items on the first page in a sequence of pages. These assumptions are expected to generally hold on database-backed web sites that generate HTML from templates as opposed to hand-writing their HTML.

The algorithm is based on two observations. First, in most item collections, each item contains a link, often to a detail page about the item. So, links are likely to be useful as starting points for generating hypotheses for the XPath. Second, the item collection is typically the main purpose of the web page, so the items themselves consume a large fraction of the page's visual real-estate. This gives us a way to choose the most likely hypothesis, namely, the one that uses the largest area of the page.

The item detection algorithm starts by collecting all unique XPaths to `<A>` elements on the current web page. For each element, its XPath is calculated by stringing together the tag names of all elements from the document root down to that element. CSS class names are also included. The resulting XPath looks something like this: `/HTML/BODY/TABLE/TBODY/TD/DIV[@class='product']/SPAN/A`. Each such XPath is general enough to cover more than just the original `<A>` element, but restrictive enough to address only those elements similar to it. This is called the generalization phase, as illustrated by the straight arrows in Figure 4.3.

Each of these `<A>` XPaths addresses a collection of `<A>` elements that could correspond one-to-one with the collection of items to be detected. We wish to find which of these `<A>` XPaths corresponds to a collection of items that take the largest amount of screen space.

Next, each `<A>` XPath is expanded to fully encompass the hypothetical items that the `<A>` elements reside within. To expand an XPath, we repeatedly append `/..` to it (see the curved arrows in Figure 4.3). As `/..` is appended, the set of HTML elements that the XPath addresses gets closer and closer to the document root. As long as the cardinality of that set remains unchanged, each HTML element in that set still resides spatially inside a hypothetical item. When the cardinality of the set drops, the XPath has been expanded too much such that it now describes the par-

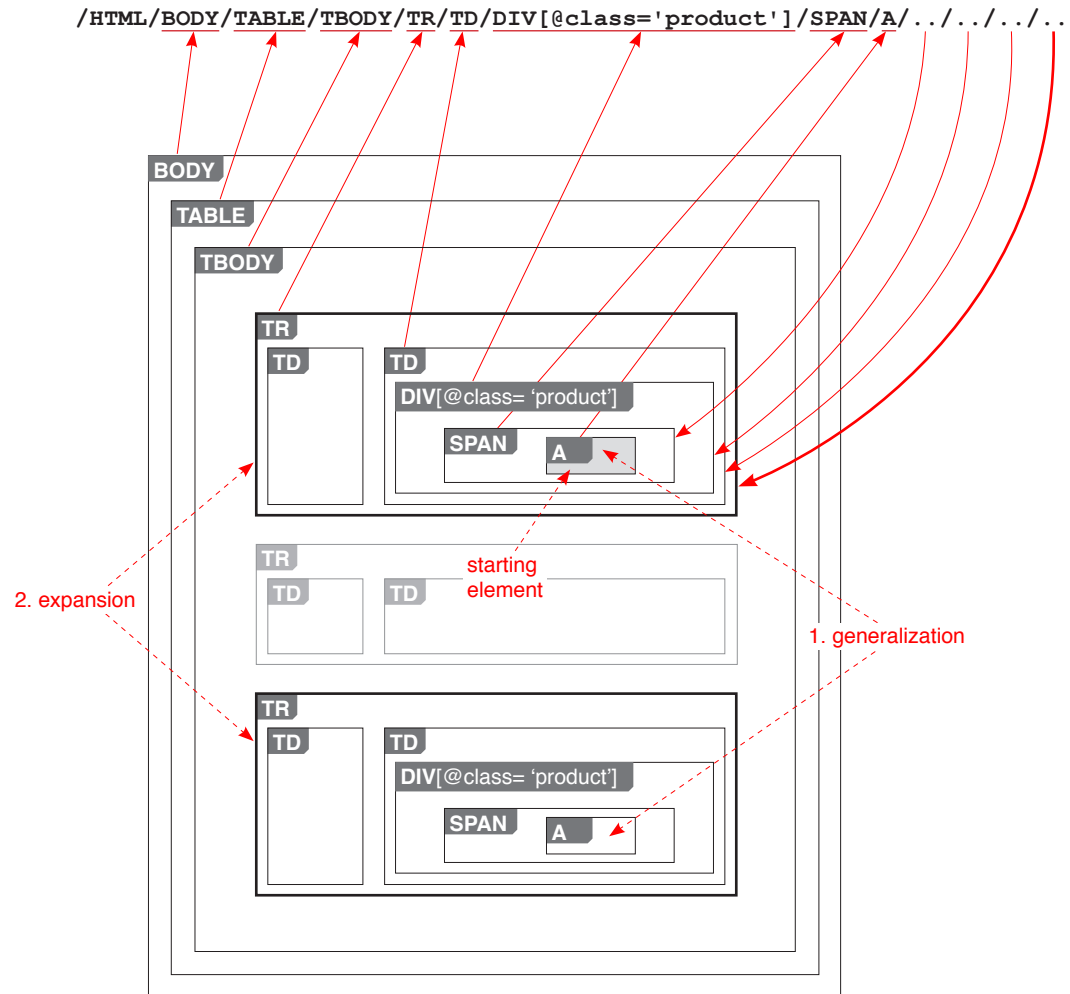


Figure 4.3. Given a starting HTML element, an item XPath is constructed by generalization to similar elements (straight arrows) and then expansion (curved arrows).

ent node(s) of the hypothetical items. For example, in Figure 4.3, if we append another `/. .`, the resulting XPath would address a single **TBODY** element rather than two **TR** elements. We stop appending `/. .` just before that happens. The result is a candidate item XPath.

Note that we append `/. .` rather than truncate ending segments because truncation loses information. If the XPath in Figure 4.3 were instead truncated 4 times, the resulting XPath, `/HTML/BODY/TABLE/TBODY/TR`, would have included the middle **TR**, which does not have a link inside and could be extraneous, intervening content. Using `/. .` guarantees matching only nodes that have the link inside them.

For each candidate item XPath, we calculate the total screen space covered by the HTML elements it addresses. The candidate item XPath with the largest screen space wins and is then used to add highlight overlays to the web page.

4.2.2 Subsequent-Page Detection

Two heuristics are used to automatically detect subsequent pages. The heuristics are run in the order presented below, and when one succeeds, its results are taken as final.

4.2.2.1 *Link Label Heuristic*

Often, a web page that belongs in a sequence of pages contains links to the other pages presented as a sequence of page numbers, e.g.,

Pages: 1 [2] [3] [4] [5] [Next] [Last]

Occasionally, such a sequence shows not the page numbers but the indices of the items starting on those pages, e.g.,

Items: 1–10 [11–20] [21–30]

This heuristic attempts to pick out URLs from such a sequence of linearly increasing numbers. First, the text labels of all **<a>** links on the current web page are parsed. Only labels that contain numbers are kept. They are then grouped by the XPaths generated from the **<a>** elements. For each XPath, the numbers parsed from the labels are sorted in ascending order. Only those XPaths with linearly increasing sequences of numbers are kept. These final candidates are then sorted by the lengths of their sequences. The XPath with the longest sequence is then used to pick out URLs to subsequent pages. If there is a tie, the XPath whose sequence increases at the highest rate wins. This heuristic fails if no XPath has a linearly increasing sequence of numbers.

4.2.2.2 *URL Parameter Heuristic*

URLs of pages in a sequence often encode the page numbers or the starting item indices as numeric URL parameters. For instance, Amazon.com encodes page numbers in the **page** parameter and Yahoo.com encodes starting item indices in the **b** parameter (Table 4.1). This heuristic attempts to detect such parameters so that URLs to subsequent pages can be generated. The URLs pointed to by all the links on the current web page are parsed to extract out URL parameters. For each parameter that has numeric values, its numeric values are collected in an array and sorted in ascending order. Then, only parameters whose values form linearly increasing sequences are kept. These final candidates are sorted by the lengths of their value sequences. The parameter with the longest sequence is then used to generate URLs to subsequent pages. If there is a tie, the parameter whose sequence increases at the highest rate wins. This heuristic fails if no parameter has a linearly increasing sequence of values.

If these heuristics fail then the user can intervene and point at the link to one of the subsequent pages (not necessarily the immediately following page). The XPath of that link is then computed, which describes a collection of **<a>** elements. Given such a collection of **<a>** elements, the following heuristic is used to pick out the one that points to the next page.

4.2.3 Field Detection

Field values within the DOM fragments of items can be picked out if what vary among the fragments can be isolated from what are common among many of them. To detect what are common and what vary, any tree alignment algorithm [53, 62, 70] can be applied to these DOM fragments. The result of the tree alignment is a *DOM tree template* with some nodes marked as variable and some nodes marked as common, or non-variable. The non-variable nodes are decorative elements, and the variable nodes correspond to fields. Generic field names (e.g., “field0”, “field1”) are then assigned to the variable nodes in the DOM tree template.

After field assignment is complete, the template is ready to be used. The DOM fragment of each item to be extracted is aligned to the DOM tree template. If a node in the item is aligned to a variable node in the template, its text content is extracted and stored in the corresponding field.

4.2.3.1 Embedded, Typed Fields

Often, a field value does not take up a whole HTML text node. That is, you might not find a price field value “14.99” to be the only text inside a DOM text node. Instead, you might find that the text node contains “You save \$14.99!” To handle such cases, the tree alignment must work not at the granularity of HTML nodes, but at the granularity of *tokens* within text nodes.

Care must be taken when parsing “You save \$14.99!” into tokens lest “14” and “99” be split into different tokens. Splitting “14” and “99” apart will cause two different fields to be generated; sorting or filtering by a single price field is no longer possible. Similarly, splitting “Published on March 29, 1998” into tokens must keep “March”, “29”, and “1998” together as a single date token. In other words, tokenization must generate typed tokens for numbers and dates so that sorting and filtering on the corresponding fields will be possible.

A side effect of tokenization is that paragraphs of text (e.g., product review comments, publication abstracts) can cause too many fields to be generated as their tokens don’t align. To solve this problem, in the field assignment phase, if there are too many variable tokens that are siblings of one another, then their immediate parent is assigned a field name instead.

4.3 Evaluation

Sifter has been tested for robustness in its extraction algorithm as well as for usability in its user interface. The two evaluations are presented separately below.

4.3.1 Evaluation of Data Extraction

Sifter’s extraction algorithm has been tested on 30 collections of items from 30 common web sites, including Amazon, Best Buy, CNET Reviews, Froogle, Target, Walmart, and Yahoo Shopping (Table 4.3). These collections contain from about 25 to 450 items each, spanning from two to 25 pages, with each page containing from nine to 50 items. The results are shown in Table 4.4.

The item detection algorithm worked perfectly on the first pages of 23 collections (out of 30, 77% accuracy). For the failure cases, Sifter’s item highlighting tool was used and four cases were corrected. Overall, item XPaths could be found for 27 of the 30 collections (90%). In the remaining three cases, items consisted of sibling nodes, which we do not currently handle.

Among the 30 sites, 24 (80%) displayed sequences of page numbers, also called page tables of content (rather than just “Next Page” links). The Link Label Heuristic detected 13 sequences, and the URL Parameter Heuristic detected two ($(13 + 2)/24 = 63\%$). The Next Page Heuristic (requiring users’ intervention) worked on 9 of the remaining 15 cases ($9/15 = 60\%$). Overall, subsequent pages could be identified for 23 out of 30 collections (77%).

There were 21 collections for which item XPaths could be found and subsequent pages could be identified accurately. Out of these 21 collections, 19 were perfectly extracted, yielding precisely the original numbers of items. The overall accuracy of extraction is $19/30 = 63\%$.

Note that accuracy was measured per collection rather than per item as in other data extraction work. To put this in perspective, the latest work on data extraction [77] processed 72 *manually provided* pages from 49 sites and achieved $40/49 = 82\%$ collection accuracy. Over the 23 collections for which subsequent pages could be identified, our algorithm processed 176 pages automatically and achieved $19/23 = 82.6\%$ collection accuracy.

The fields extracted that could be useful for filtering and sorting included: current price, original price, percent saving, author, artist, medium, date, shipping option, brand, number of store reviews, number of bids, container size, city, etc.

4.3.2 Evaluation of User Interface

Since augmentation of web sites is a novel concept even to experienced web users, a formative evaluation of Sifter’s user interface has been conducted to determine whether it was basically usable and useful, assuming the automatic extraction algorithm performed its best.

4.3.2.1 *Design and Procedure*

This study consisted of a structured task (during which the subjects took simple steps to familiarize with Sifter) followed by an unstructured task (during which the subjects employed their own knowledge of Sifter for problem solving).

At the beginning of each study session, the subject was told that she would learn how to use something called Sifter herself but was given no particular instructions on how to use it. This was patterned after the study on the Flamenco system in which the subjects were not introduced to the system in order to better mimic real world situations [76].

Task #1 required the subject to:

- follow a sequence of simple steps to use the Sifter pane to sort and filter a collection of 48 items spread over 3 web pages obtained by searching Amazon.com for “jeffrey archer.” The desired final result was the sub-collection of only hardcovers and paperbacks published in 2004 or later, sorted in descending order by their used & new prices. The sequence of steps consisted of high-level “filter by date” and “sort by price” instructions, not low-level UI “click this button” and “select that list item” actions.
- use the Sifter pane by herself to obtain the list of 3 cheapest (if bought used) paperbacks by John Grisham in 2005 from Amazon.com.
- spend no more than 5 minutes using only the Amazon web site, but not Sifter, to find the 3 cheapest (if bought used) hardcovers by John Grisham in 2004.

Task #2 required the subject to:

- use the Sifter pane to decide whether the sale on Prada products on Ashford.com was good or not.
- use the Sifter pane to list 2 or 3 products among those that the subject considered good deals.
- use only the Ashford.com web site to judge whether the sale on Gucci products on Ashford.com was good or not, using the same criteria that the subject had used in judging the sale on Prada products.

Amazon.com was chosen for Task #1 as its search results were very structured, containing many fields useful for filtering and sorting. Furthermore, Amazon.com is popular and the subjects were more likely to be familiar with it. Ashford.com was chosen for Task #2 as its search results contained only two fields (price and percent saving), making it simpler to perform the high-level task of judging its sales.

At the end of the session, the subject rated her agreement/disagreement with 12 statements (on a 9-point Likert scale) regarding her experience learning and using Sifter.

4.3.2.2 *Participants*

Eight subjects (4 male, 4 female) were recruited by sending an e-mail message to a mailing list and posting paper ads around a local college campus. Six were in their 20s, the other two were 30s and 40s. All 8 subjects used the Web almost everyday, and all subjects visited Amazon.com at least a few times a month. None had ever visited Ashford.com.

4. EXTRACTING DATA

| | Web Site | Collection within Web Site | #Items/Page x #Pages | Has Page TOC? | Total #Items |
|-----|----------------------------|----------------------------------|----------------------|---------------|--------------|
| 1. | acehardware.com | Kitchen faucets (plumbing) | 9 x 5 | ● | 44 |
| 2. | adesso.us | Keyboards | 16 x 2 | ○ | 26 |
| 3. | alibris.com | Search for "John Grisham" | 25 x 4 | ● | 96 |
| 4. | amazon.com | Search for "Jeffrey Archer" | 16 x 3 | ● | 48 |
| 5. | ashford.com | Prada products | 9 x 7 | ● | 59 |
| 6. | bargainoutfitters.com | Women's footwear | 12 x 25 | ● | 290 |
| 7. | bestbuy.com | Point & shoot digital cameras | 25 x 5 | ● | 111 |
| 8. | buy.com | Box sets | 12 x 6 | ○ | 72 |
| 9. | cameraworld.com | SLR lens over \$400 | 25 x 4 | ● | 95 |
| 10. | reviews.cnet.com | Dell desktops | 10 x 10 | ○ | 93 |
| 11. | compusa.com | Search for "hard drive" | 20 x 15 | ● | 287 |
| 12. | dealttime.com | Lavender (flowers and plants) | 21 x 9 | ● | 179 |
| 13. | drugstore.com | Hand creams (lotions) | 15 x 6 | ○ | 87 |
| 14. | antiques.listings.ebay.com | Globes (maps, atlases, globes) | 50 x 9 | ● | 444 |
| 15. | essentialapparel.com | Women's sportswear | 10 x 6 | ○ | 55 |
| 16. | froogle.google.com | Search for "rebel xt" | 10 x 10+ | ● | >100 |
| 17. | newegg.com | Notebooks/laptops, \$1500-\$2000 | 20 x 3 | ● | 42 |
| 18. | nextag.com | Hotels in Boston, 3* | 15 x 4 | ● | 58 |
| 19. | nordstrom.com | Women's wallets & accessories | 21 x 4 | ● | 74 |
| 20. | officedepot.com | Fully adjustable chairs | 10 x 6 | ● | 55 |
| 21. | overstock.com | Coins & stamps (collectibles) | 24 x 3 | ○ | 58 |
| 22. | radioshack.com | All MP3 players & ipods | 10 x 5 | ● | 50 |
| 23. | rochesterclothing.com | Casual pants | 20 x 3 | ● | 53 |
| 24. | shoebuy.com | Adidas, womens, 6.5 | 12 x 3 | ● | 26 |
| 25. | shopping.com | Stainless steel rings < \$50 | 30 x 8 | ● | 236 |
| 26. | smartbargains.com | Ties (men's apparel) | 16 x 2 | ● | 28 |
| 27. | target.com | Clearance "table" | 20 x 14 | ● | 276 |
| 28. | tigerdirect.com | Digital photo printers | 10 x 4 | ● | 38 |
| 29. | walmart.com | Houseware, \$20-\$50 | 20 x 4 | ● | 76 |
| 30. | shopping.yahoo.com | PDA cell phones | 15 x 15 | ● | 222 |
| | | | | | 3378 |

Table 4.3. Thirty popular retailer web sites were used to test Sifter's web data extraction algorithm. A search or browse operation was performed on each site and the number of result items per page and the number of result pages were noted. Whether the first search result page contained a table of content of links to subsequent result pages (as opposed to "Next" links) was also recorded as it affected the algorithm.

| Pages Detected? - Correctable By Users? | | | | | | | |
|---|---|----|--|-----------------------|-------|---|--|
| Items Detected? - Correctable By Users? | | | | Total Original #Items | | | |
| Has Page TOC? | | | Total #Items Extracted - Perfect extraction? | | | | |
| Web Site | | | | | | Useful Fields | |
| 1. acehardware.com | ● | ● | ● | 44 | 44 ✓ | price | |
| 2. adesso.us | ○ | ● | ○○ | 26 | 16 | | |
| 3. alibris.com | ● | ● | ● | 96 | 96 ✓ | author, old price, new price, quick buy price, signed copies? | |
| 4. amazon.com | ● | ● | ● | 48 | 48 ✓ | author, type, date, total # items, buy new original price, buy new current price, used & new price, shipping info | |
| 5. ashford.com | ● | ● | ● | 59 | 59 ✓ | price, percent saving | |
| 6. bargainoutfitters.com | ● | ● | ○● | 290 | 290 ✓ | price | |
| 7. bestbuy.com | ● | ○○ | ● | 111 | 0 | | |
| 8. buy.com | ○ | ● | ○● | 72 | 72 ✓ | artist, price, saving | |
| 9. cameraworld.com | ● | ○● | ○● | 95 | 95 ✓ | price | |
| 10. reviews.cnet.com | ○ | ● | ○● | 93 | 93 ✓ | price | |
| 11. compusa.com | ● | ● | ○○ | 287 | 287 ✓ | brand | |
| 12. dealtime.com | ● | ● | ○● | 179 | 179 ✓ | # store reviews | |
| 13. drugstore.com | ○ | ○● | ○○ | 87 | 15 | price, size | |
| 14. antiques.listings.ebay.com | ● | ● | ● | 444 | 444 ✓ | # bids, price | |
| 15. essentialapparel.com | ○ | ● | ○○ | 55 | 110 | old price, current price | |
| 16. froogle.google.com | ● | ● | ● | >100 | 100 ✓ | minimum price | |
| 17. newegg.com | ● | ● | ○○ | 42 | 20 | shipping fee, # reviews | |
| 18. nextag.com | ● | ● | ● | 58 | 55 | city, zip code, price 1, price 2 | |
| 19. nordstrom.com | ● | ● | ● | 74 | 74 ✓ | price | |
| 20. officedepot.com | ● | ○○ | ● | 55 | 0 | | |
| 21. overstock.com | ○ | ● | ○○ | 58 | 24 | | |
| 22. radioshack.com | ● | ● | ● | 50 | 50 ✓ | has ratings?, out of stock/in store/on-line, 1-2 or 2-3 days shipping | |
| 23. rochesterclothing.com | ● | ○● | ● | 53 | 52 | price | |
| 24. shoebuy.com | ● | ○● | ○● | 26 | 26 ✓ | on sale?, price, % off, original price | |
| 25. shopping.com | ● | ○○ | ○○ | 236 | 0 | | |
| 26. smartbargains.com | ● | ● | ○● | 28 | 28 ✓ | retail value, our price, % saving, # left | |
| 27. target.com | ● | ● | ● | 276 | 276 ✓ | list price, our price, save amount, save percent, shipping info | |
| 28. tigerdirect.com | ● | ● | ● | 38 | 38 ✓ | price, in stock or shipping info, brand | |
| 29. walmart.com | ● | ● | ○● | 76 | 76 ✓ | | |
| 30. shopping.yahoo.com | ● | ● | ○● | 222 | 210 | exact price | |
| | | | 77%, 90% | 43%, 77% | 3378 | 2820 | |

Table 4.4. The results of testing Sifter's web data extraction algorithm. Whenever the algorithm failed, user intervention was attempted and the intervention's success was recorded. User intervention brought the overall success of locating items to 90% and of locating subsequent pages to 77%. The success rate for recovering *whole* collections was $19/30 = 63\%$.

4. EXTRACTING DATA

All subjects had used the Web for more than just shopping. They had searched for some combinations of the following types of information: news; images; contact information of people and organizations; maps and driving directions; hobbies (e.g., recipes, chess); reviews (on restaurants, movies, books, products); tutorials (e.g., languages, logics); and professional research (e.g., publications, scientific data).

4.3.2.3 Apparatus

Subjects received \$10 each for participating in a 30 – 45 minute study session. All sessions were conducted by one investigator on a single computer (Pentium 4 2.53GHz, 1.00GB) with an 18" LCD flat panel at 1600×1200 resolution in 32-bit color and a Microsoft Wireless IntelliMouse Explorer 2.0 (with mousewheel), running Microsoft Windows XP. UI events were recorded in a timestamped log and the investigator observed the subjects and took written notes.

4.3.2.4 Results

All subjects completed the parts of Task #1 involving Sifter. Only 5 out of 8 completed the parts involving using the Amazon web site *without* Sifter. The other subjects could not learn how to use Amazon to perform sophisticated queries within 5 minutes. Among the 5 subjects who succeeded, only one made use of Amazon's Advanced Search feature. The other 4, despite their previous experience with the Amazon web site, could only sort the items by one criterion and manually scan the list for items satisfying the other criteria. This indicates that advanced browsing features implemented by the web browser in a unified manner across web sites may be more discoverable, learnable, and usable than those same advanced features officially supported by individual web sites but have been suppressed in favor of more commonly used functionality.

Seven subjects completed Task #2 and one refused to finish Task #2 as he said he had no knowledge of Prada and Gucci products and thus could not judge their sales. For the first part of Task #2, 6 out of the 7 subjects used Sifter to look at the distribution of the percent saving. One subject could not understand how Sifter would help her judge the sale.

Table 4.2 shows encouraging evidence that the subjects found Sifter powerful yet easy to learn and use. However, the extraction process was thought to be slow. Data extraction speed depends on network performance and web server responsiveness, but on average, each test collection of 50 or so items took 30 seconds to extract.

Although there was no show-stopper problem with the user interface, some users were taken aback by the verification step (when the system announced its estimate of the items to be extracted and asked for confirmation). As they saw no other choice except clicking "Continue," they did so just to see what would happen next, in hope but not certain that that route would ultimately allow them to sort and filter. This behavior was not a surprise as web site augmentation was a new experience and the necessity for extracting data before augmentation could take place was poorly understood, if understood at all. To fix this problem, the accuracy of the algorithms must be boosted, subsequent pages must be pre-loaded and pro-

cessed even before the user clicks “Filter/Sort Items,” and the user must be allowed to make corrections from the augmentation UI.

Sorting operations took very little time and the re-shuffling of items inside the web page was too fast and subtle to shift the user’s attention from the Sifter pane where she just invoked a sorting command to the web page. The user often had to double-check the resulting list of items herself. To fix this, we can slow down or animate the changes in the web page. Filtering operations produced more drastic changes and did not suffer from the same problem.

One subject opened too many browsing control boxes and became confused as to which field each box corresponded to. He was not able to notice the synchronized highlighting of browsing control boxes and field asterisks. To fix this, we can color-code the asterisks and the browsing control boxes as well as use a variety of shapes rather than just asterisks.

Asked to filter for only items published in 2005, some subjects had to manually find one sample item published in 2005 in order to click on the asterisk next to its publishing date. Other subjects simply clicked on the asterisk next to any publishing date. If we are able to derive meaningful field names, this problem will be resolved.

Five of the 8 subjects interacted with the faded areas of the web pages when they needed to use the web sites’ functionality (e.g., performing a search for “john grisham”). The other three subjects either refreshed the web pages or retyped their URLs. In the future, Sifter’s UI can be changed to let users feel more comfortable making use of the original web sites’ features, knowing the difference between those original features and the added functionality.



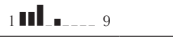

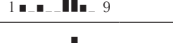







| | |
|--|---|
| strongly disagree 1  9 strongly agree | 1. Sifter is hard to learn how to use. |
| 1  9 | 2. Sifter is tedious to use. |
| 1  9 | 3. The filtering and sorting features in Sifter are slow. |
| 1  9 | 4. Sifter shows redundant information (easily found on the Web sites). |
| 1  9 | 5. After clicking “Continue,” I need to wait for a long time before I can use Sifter. |
| 1  9 | 6. Sifter is simple to use. |
| 1  9 | 7. Sifter is powerful (providing advanced features). |
| 1  9 | 8. Sifter displays interesting information. |
| 1  9 | 9. Sifter displays useful information. |
| 1  9 | 10. Sifter is enjoyable to use. |
| 1  9 | 11. Sifter adds value to the Web sites in this user study. |
| strongly disagree 1  9 strongly agree | 12. I believe Sifter will add value to some other Web sites I have used. |

Table 4.2. Results from the exit survey of the formative evaluation show encouraging evidence that the Sifter pane is usable (#1, #2, #6) and useful (#8, #9, #11, #12) even when it is considered to offer advanced functionality (#7) .

4. EXTRACTING DATA

One subject—the only one who used Amazon’s Advanced Search feature—asked when she would be able to use Sifter in her own browser. She mentioned that there were a number of sites she used frequently which did not offer the browsing functionality that she needed. Another subject said that although he never had to perform the kinds of task in this study on the Web, he had to perform similar tasks in spreadsheets.

4.4 Summary

This chapter puts forth ideas in which presentational elements of web pages can be retained and used in web data extraction toward saving users from the tedious field labeling task as well as enhancing the usage of the extracted data. Advanced browsing features can be added right inside the original pages, preserving visual context and leveraging custom presentational designs on the original pages rather than downgrading to a generic presentation template.