Auto-Suggesting Browsing Actions for Personalized Web Screen Reading

Vikas Ashok
Stony Brook University
vganjiguntea@cs.stonybrook.edu

Syed Masum Billah
Stony Brook University
sbillah@cs.stonybrook.edu

Yevgen Borodin
Charmtech Labs LLC
yevgen.borodin@team.captivoice.com

IV Ramakrishnan
Stony Brook University
ram@cs.stonybrook.edu

ABSTRACT

Web browsing has never been easy for blind people, primarily due to the serial press-and-listen interaction mode of screen readers – their "go-to" assistive technology. Even simple navigational browsing actions on a page require a multitude of shortcuts. Auto-suggesting the next browsing action has the potential to assist blind users in swiftly completing various tasks with minimal effort. Extant auto-suggest feature in web pages is limited to filling form fields; in this paper, we generalize it to any web screen-reading browsing action, e.g., navigation, selection, etc. Towards that, we introduce *SuggestOmatic*, a personalized and scalable unsupervised approach for predicting the most likely next browsing action of the user, and proactively suggesting it to the user so that the user can avoid pressing a lot of shortcuts to complete that action.

SuggestOmatic rests on two key ideas. First, it exploits the user's Action History to identify and suggest a small set of browsing actions that will, with high likelihood, contain an action which the user will want to do next, and the chosen action is executed automatically. Second, the Action History is represented as an abstract temporal sequence of operations over semantic web entities called Logical Segments - a collection of related HTML elements, e.g., widgets, search results, menus, forms, etc.; this semantics-based abstract representation of browsing actions in the Action History makes SuggestOmatic scalable across websites, i.e., actions recorded in one website can be used to make suggestions for other similar websites. We also describe an interface that uses an off-the-shelf physical Dial as an input device that enables SuggestOmatic to work with any screen reader. The results of a user study with 12 blind participants indicate that SuggestOmatic can significantly reduce the browsing task times by as much as 29% when compared with a hand-crafted macro-based web automation solution.

CCS CONCEPTS

• **Information systems → Personalization**; **Web interfaces**; • **Human-centered computing → Accessibility technologies**.

UMAP '19, June 9–12, 2019, Larnaca, Cyprus

ACM ISBN 978-1-4503-6021-0/19/06.
https://doi.org/10.1145/3320435.3320460

KEYWORDS

Personalized Web Automation, Auto Suggest, Web Accessibility; Blind, Vision Impairment; Dial-based Interaction.

ACM Reference Format:
Vikas Ashok, Syed Masum Billah, Yevgen Borodin, and IV Ramakrishnan. 2019. Auto-Suggesting Browsing Actions for Personalized Web Screen Reading. In *27th Conference on User Modeling, Adaptation and Personalization (UMAP '19), June 9–12, 2019, Larnaca, Cyprus.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3320435.3320460

1 INTRODUCTION

Blind people interact with applications including web browsers primarily using screen readers (e.g., NVDA [23], JAWS [17], Dolphin [31], Window-Eyes [33], VoiceOver [32], etc.). Screen readers enable blind users to sequentially navigate webpage content using keyboard shortcuts, with all content being either narrated using synthesized voice or converted into refreshable Braille. However, this serial "press-and-listen" interface of screen readers have been shown to be inadequate for providing a satisfactory and usable web-browsing experience for blind users ([12, 18]). Furthermore, most screen-reader users rely only on a small basic set of sequential navigation shortcuts [12]. As a consequence, they end up spending a lot of time in completing even simple day-to-day browsing tasks such as online shopping, flight reservation, etc.[3, 4].

One approach to overcome these screen-reader issues is to use web automation that performs browsing tasks on users' behalf. To automate browsing tasks, almost all extant web automation techniques ([8, 19]) typically use macros or scripts, containing predefined or prerecorded sequences of browsing actions. However, the use of macros makes the current web-automation techniques unsuitable for *ad hoc* browsing actions, i.e., tasks where users do not follow the same fixed sequence of browsing actions every time they do these tasks. For example, while booking flights online, users may choose different search-result filters depending on their preferences, go back and search again for flights with different input dates to get better price deals, compare prices with other similar travel websites, etc. In other words, there is no fixed sequence of actions; the users will continue to browse until they find flights that meet their requirements. In addition to the limitations of macros, most web-automation techniques require blind users to manually find and replay macros, and sometimes even create and manage the macros if personalization is desired. All these problems with extant automation techniques restrict their utility for blind people; none

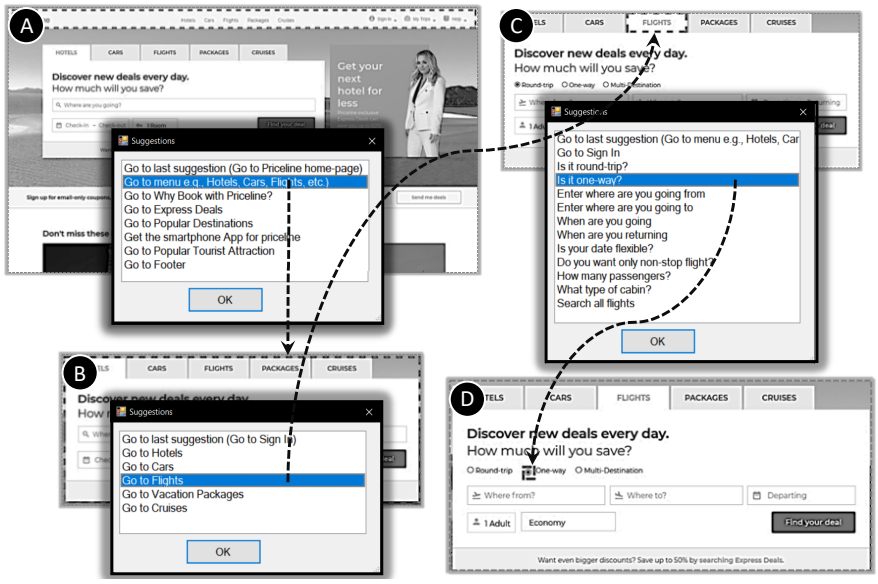

Figure 1: Illustration of auto-suggest as embodied in SuggestOmatic. Upon user request, the "Suggestions" dialog box appears at the middle of each view. (A) A user opens the homepage of a travel booking site. The "Suggestions" dialog shows available options the user can choose. The user chooses "Go to menu e.g., Hotels, Cars, Flight ..." option. (B-D) Intermediate steps along with suggestions at each step towards making a "one-way" flight reservation.

of the 12 blind participants in our user study reported knowing or using any web automation technology.

Instead of web automation as described above, we envision using auto-suggest as the means to predict and automate the next browsing action (e.g., navigate to next search-result item), thereby assisting blind users in swiftly completing various ad hoc web browsing tasks with minimal effort. The idea of auto-suggest does exist for web pages but is limited to filling form fields; in this paper, we generalize it to any browsing action, e.g., navigation, selection, etc. Towards that, we introduce SuggestOmatic, a flexible and scalable unsupervised approach for predicting the most likely next browsing action of the user, and proactively suggesting it to the user so that the user can avoid pressing a lot of shortcuts to complete that action. Figure 1 is an illustration of auto-suggest in action, as embodied in SuggestOmatic.

The distinguishing characteristics of SuggestOmatic are as follows: First, instead of using macros, it leverages user's *semantic* Action History – a temporal sequence of user's past browsing actions where each action is represented as an operation (e.g., navigate, select, fill, etc.) over some Logical Segment (a collection of related HTML elements such as widgets, search results, menus, forms, etc.). This semantics-based abstract representation (e.g., navigate to search results) instead of syntax-based HTML representation (e.g., move cursor to DOM node *abc*) enables SuggestOmatic to use actions recorded on one website to provide suggestions for other similar websites even if the user never visited those websites. Second, SuggestOmatic utilizes a custom scoring model, based on adapting a local sequence alignment algorithm [29] used in bioinformatics, to rate the actions in Action History according to their likelihood of being the next action that the user wants to do in the context of the user's most recent browsing actions. The top scoring actions are then suggested to the users for automation, and SuggestOmatic will execute the chosen browsing action on behalf of the user. Third, the scoring model is programmed to continuously and incrementally update the scores of browsing actions in Action History after every user action, so that the generated suggestions are kept *fresh* and not outdated.

We also present a novel interface for SuggestOmatic based on a separate off-the-shelf physical dial input device that supports simple rotate and press gestures with audio-haptic feedback using which users access the various features of SuggestOmatic. More importantly, the dial also makes SuggestOMatic work with any screen reader. We also report on a user study with 12 blind participants on the effectiveness and usability of SuggestOmatic.

2 RELATED WORK

In this section, we review some of the popular web automation techniques as well as other approaches that improve non-visual web access.

Traditional web automation approaches typically adopt the following two approaches – handcrafting, e.g., ([11, 22]) and Programming by Demonstration (PBD) ([2, 8, 19]). The handcrafting method involves writing scripts to: (a) customize the behavior of the browser or screen-reader, or (b) encode automation instructions as a feature in the browser or screen-reader, e.g., the JAWS screen-reader supports a shortcut to look up words in an online dictionary. While user interfaces for handcrafting are available in some cases (e.g., [22]), there is, however, an added burden on users to learn the required scripting language. For example, to create macros for the JAWS screen reader, users have to consult a 180+ page manual for learning and using the JAWS scripting language. Compared to handcrafting, PBD based approaches are more user-friendly in that a user

can demonstrate how to perform a task by simply executing the relevant browsing actions in a proper sequence; PBD approaches (e.g., [19]) automatically record this sequence as a macro that can be replayed anytime the user wants to do the same task again. Furthermore, PBD based macros from one website can sometimes be dynamically modified for reuse on other similar websites [8], thereby making them scalable across websites. However, in all the aforementioned PBD approaches, demonstrating the sequence of actions in a macro from the beginning to the end, involve considerable user time and effort. Smart Bookmarks [16] tries to ease this burden to a little extent by letting the users indicate only the end of macros whenever they have completed the corresponding tasks; Smart Bookmarks tries to automatically infer the beginning. However, this approach is not fully accurate. Nonetheless, all these approaches lack flexibility; if a user wants to make changes to the sequence of browsing actions in a macro, or if there are changes in the website itself, the user has to re-record the task macro.

The proposed design and development of SuggestOmatic was inspired by several ideas [25–27]. However, unlike the above PBD approaches, SuggestOmatic does not record macros; it instead depends on user's Action History for identifying most-probable subsequent user actions.

To improve flexibility and user experience, a few works [20, 22, 30] have looked at using state-transition models instead of macros. These state-transition models represent tasks as a collection of browsing states with browsing actions as transitions. These models can either be manually defined or automatically constructed from sequences of browsing actions using machine learning techniques such as clustering, classification, and automata learning. By allowing users to go back-and-forth between different states, a certain degree of flexibility and ad-hoc ness can be achieved. However, these models are static, the browsing actions are fixed, and therefore not scalable and adaptable; every time changes are required, the models have to be rebuilt.

Perhaps the closest related work to SuggestOmatic is the predictive automation Assistant proposed by Yury et. al.[28]. Like SuggestOmatic, the Assistant also uses a custom local alignment algorithm on users' past browsing history to predict and suggest various browsing actions for automation. However, the Assistant has several limitations. Firstly, the approach is not scalable across websites because browsing actions are represented as actions on HTML elements on the webpage. For example, an action Click flights link recorded in a travel website cannot be used on other travel websites since they may have a flights button instead of a link. Similarly, the search button may be labelled differently on different websites (e.g. find, go, arrow, etc.). In contrast, representing the same actions in a semantically abstract manner, e.g., Select flights menu option, Search for flights, etc., as is done in SuggestOmatic, enables actions to be reused across websites, since mapping from abstract logical entities (menu option) to actual DOM node elements (link, button) is done dynamically at runtime depending on the website and context. Also, the Assistant supports only automation of very basic browsing actions such as clicking and form filling; it does not provide any support for navigating the screen-reader cursor to content of interest, the main source of tedium and frustration for blind screen-reader users. SuggestOmatic, on the other hand, supports automated navigation.

Prediction of user browsing behavior, has also found applicability in other application domains such as prefetching files [15, 24], ecommerce, content personalization [7], etc., which are orthogonal to the work presented in this paper. These applications typically construct probability graphs similar to the previously mentioned task models, and hence suffer from the same issues of errors due to incorrect segmentations and clustering.

3 SUGGESTOMATIC DESIGN

3.1 Technical Preliminaries

In this section, we introduce some terminologies that will be used in the remainder of the paper.

3.1.1 Logical Segment Model. A Logical segment (LS) is a collection of related HTML elements that share common spatial and functional properties with a discernable visual boundary. For example, the high-level visually-segregated blocks such as menu, sidebar, search or login forms, main-content, search results, filter options, footer, user comments section, widgets, etc., are all examples of Logical Segments. A Logical Segment may have its own properties (e.g., dates of a calendar, price of an item, etc.) and can additionally be recursively partitioned into sub-LSs. Each of these segments can in turn have their own children which are segments themselves, and so on. In this way, the entire webpage can be viewed as a hierarchy of Logical Segments, and we refer to this hierarchy as a Logical Segment (LS) Model.

To construct the LS Model, we borrow existing techniques in the related literature [9, 10, 14]. These techniques build LS Models by leveraging extensive custom-defined Web-Entity Ontologies [6] that provide the necessary blueprints or meta information to identify and represent various Logical Segments, such as forms, filters, menu, search-results, and calendars, as well as their characteristics and relationships. Just like these techniques, we used the existing well-studied methodologies and algorithms [6, 21, 34–36] to extract the various Logical Segments from raw HTML DOM data, and then use the custom defined ontology to represent and organize them into an LS Model. Just like in [5], this Model is constructed dynamically every time a new page is loaded and continuously updated whenever new Logical Segments (e.g. widgets) are added to the DOM. The representation for each Logical Segment in this Model also stores the corresponding low-level mappings between that Segment and the actual HTML content. For example, pointers to the DOM nodes for various properties (e.g., price, brand, etc.) of a result item are stored in the representation of that item. These mappings are later used when the user picks a Action Suggestion.

3.1.2 User Action and Action History. We define any high-level semantically meaningful interaction with a Logical Segment in the extracted Logical Segment Model as a User Action. SuggestOmatic recognizes the following user actions: (i) navigating to an LS (e.g., product item) or a property (e.g., price of the product item) of an LS; (ii) filling out a form field; (iii) submitting a form; and (iv) selection of a LS or one of its properties (e.g., selecting a flight, choosing a date in the Calendar LS, etc.).

Storing User Actions as high-level LS operations as opposed to low-level HTML operations makes SuggestOmatic scalable across websites, since the same Logical Segment may not have the same

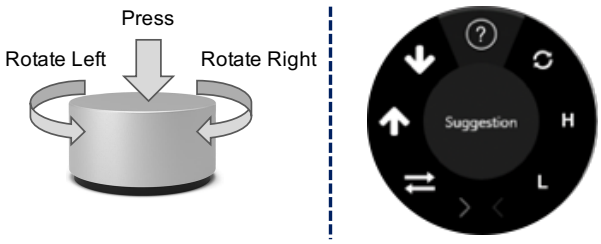

Figure 2: Left: The Surface Dial input device and gestures; Right: The onscreen SuggestOmatic dashboard.

underlying HTML structure in two different websites. The translation of User Actions to low-level HTML operations can then be done at run-time depending on the current context (i.e., website) using the mappings provided by the LS Model.

Action History is the chronologically ordered sequence of User Actions. SuggestOmatic segregates Action History based on the type (e.g., travel, shopping, social media, news, etc.) of website from which user-activity data is collected; e.g., all User Actions on flight reservation websites are stored separately from all User Actions on shopping websites. In other words, SuggestOmatic maintains different Action History for different types of websites. Determining the type of a website is a well-researched classification problem [1], and we simply leverage these existing APIs.

3.1.3 Action Suggestion. An Action Suggestion is like a *front-end* to a stored User Action. Specifically, it is a selectable option with a textual description of the corresponding User Action as its label; this label is read out whenever the suggestion is brought to focus in the SuggestOmatic interface. The users can customize the text descriptions by choosing between different templates for different types of User Action. For example, for suggestions corresponding to navigation User Actions, the templates can be *Navigate to X*, *Jump to X*, or simply *X*.

3.2 SuggestOmatic Interface Description

For SuggestOmatic, we designed a novel interface with a separate physical Microsoft Surface Dial (see Figure 2) as the input medium. This design choice was based on the findings of other works in the related literature [9, 10], that specifically demonstrated how Dial-based rotate and press gestures were natural, intuitive, and simple to execute, especially for web browsing.

Figure 3 presents an architectural schematic illustrating the workflow of the SuggestOmatic System. The user can either press regular screen-reader shortcuts or press the Dial once to request for suggestions. SuggestOmatic monitors shortcut presses, and leverages the Logical Segment Model to convert these screen-reader actions into high-level semantically meaningful logical User Actions. For example, if a screen-reader shortcut press moves the focus to the beginning of a Logical Segment *X*, then the User Action *Navigate to X* is stored in Action History. In case of suggestions, the users can rotate the Dial to focus on various choices and then pick the one they prefer by pressing the Dial. Choosing a selection will: (a) trigger SuggestOmatic to send instructions to screen reader to perform the corresponding (possibly a sequence) low-level actions using the previously mentioned mappings provided by the LS Model;

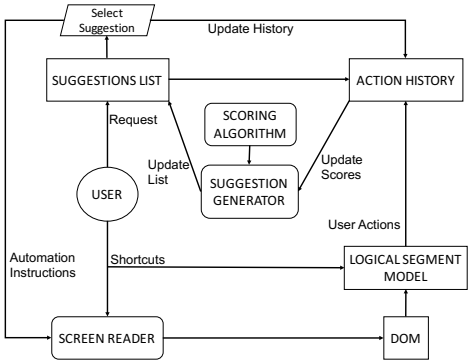

Figure 3: Architectural workflow schematic.

and (b) update the Action History by appending the User Action corresponding to this executed suggestion. Every time the Action History is updated either due to a shortcut press or selection of a suggestion, the suggestion generator uses the Smith-Waterman based scoring algorithm (described in next Section) to update scores and consequently update the suggestion list with the next set of suggestions. Also, before suggesting any User Action, SuggestOmatic first determines if it can locate the corresponding Logical Segment in its LS Model, and that the specified action can be performed on this Logical Segment. At any point in time, users can customize the interface, e.g., select the number of suggestions in the list, choose between different templates for the textual labels of suggestions corresponding to different types of User Actions, etc. Finally, while navigating the suggestions list, the users can choose not to pick any suggestion, and fall back to screen reader shortcuts by double pressing the Dial.

3.3 Generating Suggestions

To generate the suggestions, we adapted the Smith-Waterman local-sequence alignment algorithm [29]. Recall that the Smith Waterman algorithm works as follows: If $A = a_1 a_2 \ldots a_n$ and $B = b_1 b_2 \ldots b_m$ are two sequences to be locally aligned, then assuming linear gap penalty, the algorithm constructs a scoring matrix H of dimensions $(n+1) \times (m+1)$ using the equations below:

$$H_{k0} = H_{0l} = 0, 0 \leq k \leq n, 0 \leq l \leq m$$

$$H_{i,j} = \begin{cases} H_(i-1, j-1) + s(a_i, b_j) \\ H_(i-1, j) - W \\ H_(i, j-1) - W \\ 0 \end{cases}$$

Here W is the gap penalty or cost of single gap, and $s(a_i, b_j)$ is the match benefit or similarity score. In our adaptation, $W = 1$, and $s(a_i, b_j)$ is determined according to the equation below:

$$s_{a_i, b_j} = \begin{cases} 1, \text{ if } i \neq j \ \ \& \ \ a_i = b_j \\ -1, \text{ if } i \neq j \ \ \& \ \ a_i \neq b_j \\ 0, \text{ if } i = j \end{cases}$$

The main idea or trick we employ to generate scores, based on which action suggestions are made, is to match the Action History sequence with itself, so that the best local alignments of most recent

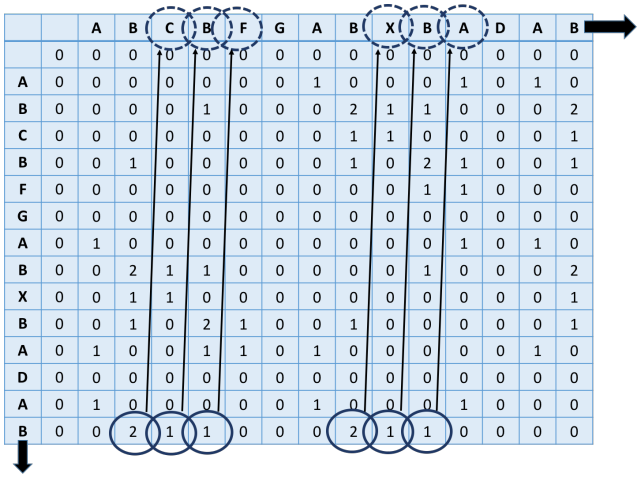

Figure 4: Generating suggestions using adapted Smith Waterman algorithm. The dotted circles represent candidates for Action Suggestions.

suffixes of User Actions can be located throughout the past Action History, and the User Actions following these matched alignments can be offered as Action Suggestions. An example illustrating this idea is provided in Figure 4. In this figure, the last row indicates the scores for local alignments between the entire History and small portions of the same History. For example, the value 2 in the third column of the last row corresponding to local-alignment results between the entire History sequence and the starting portion *AB* of the same sequence. Therefore, using the scores in the last row, we can identify potential candidates for subsequent automation suggestions. For instance, in Figure 4, since *AB* at the beginning of sequence aligns with the most recent actions *AB* at the end of the sequence, coupled with the corresponding top alignment score 2 in the last row, the next probable User Action may be *C*, and therefore it is a top candidate for Automation Suggestion.

Note that we do not assign any match benefit for elements on the leading diagonal of the scoring matrix (i.e., $s(a_i, b_j) = 0, \text{ if } i = j$). This is because the elements of both sequences come from the same Action History and therefore they will always match, thereby introducing a bias in the score computation by generating higher scores for most recent User Actions, and overshadowing other locally aligned parts of the sequence.

Using the LS Model, the top scoring candidates are then checked in descending order to determine if they can be executed (since sometimes a User Action recorded from one website cannot be used in other similar website), and if so, they are added to the suggestion list. Otherwise, these suggestions are deemed *ineligible*, and are therefore discarded, and the next ranked candidate is inspected. This process continues until either the required number of automation suggestions are generated or there are no more candidates.

Each time a User Action gets appended to the Action History, the internal scoring matrix H is updated using the above equations and the next set of suggestions are generated using the updated scores. Also, notice that given the above equations, we only need to store the values of the last row and column at any point in time instead of the entire matrix.

3.4 Evaluation

3.4.1 ***Methodology.*** To evaluate the algorithm, we first collected browsing data (denoted by D_1) to generate suggestion lists, and then measured the goodness of the generated suggestions by comparing them against a test browsing dataset (denoted by D_2). D_1 was gathered by observing 20 users perform 2 different browsing tasks, each on a different website. The 2 tasks were: (a) flight reservation on Expedia website, and (b) product purchase on Amazon website. These tasks were chosen such that they were comparable in difficulty and also represented typical use scenarios for blind screen-reader users.

For pragmatic reasons, we recruited sighted users for collecting the data. It is well known that, on an average, it takes blind users 5-10 times longer to finish the tasks compared to sighted people, thereby making data collection process expensive. However, since SuggestOmatic operates at a *semantic* level of Logical Segments and is independent of input modality (e.g., keyboard, mouse), it works the same for both sighted and blind users. Additionally, the actions suggested by the algorithm are based only on the website content; they are not specific to non-visual web browsing.

To facilitate data collection from sighted users, the participants were asked to think aloud while doing the tasks, i.e., explicitly indicate what action they wanted to do next, e.g., "I want to look at the search results". These utterances were then manually translated into user actions on Logical Segments, e.g., *Navigate to Search Results* before appending them to Action History.

Each participant was asked to do each browsing task 20 times, each time with different task parameters, e.g., shop for different products on Amazon. Since the participants were encouraged to do ad-hoc browsing while doing these tasks, each time they varied their browsing actions, e.g., filling form fields in different order, choosing different filters for search results, sorting search results based on different criteria, reviewing product details and customer comments, changing the travel dates and reloading the search-results page, randomly exploring the page, etc., thereby resulting in 20 different action sequences for each task per user. The resulting D_1 dataset contained a total of 800 sequences (40 per participant; 400 sequences each for flight reservation and product purchase) with the distribution of User Actions being 80.5% navigation, 3.4% form-field filling, 1.9% form submission, and 14.2% Logical Segment selection.

To gather the test dataset D_2, we asked the participants to do the same tasks after a gap of one week. Additionally, they were also asked to do these tasks on other similar websites (Priceline for flight reservation, and Walmart for product purchase). All participants stated that they couldn't recollect the exact action sequences they did a week before; however, they could remember some of the user actions they performed while collecting D_1. D_2 dataset consisted of 80 action sequences (4 per participant, 40 each for flight reservation and product purchase) with the distribution of user actions being 83.6% navigation, 4.2% form-field filling, 1.4% form submission, and 10.8% Logical Segment selection.

We used the *mean reciprocal rank* (MRR) metric to evaluate the goodness of the suggestion lists generated from D_1. If $A = a_1 \ldots a_n$ is an action sequence generated by a user in D_2, and $\mathcal{S} = S_1 \ldots S_{n-1}$ is the corresponding sequence of suggestion lists generated from

	Flight Reservation						Product Purchase					
	Same Website			Similar Website			Same Website			Similar Website		
Algorithm	μ_1	μ_{10}	μ_{20}	μ_1	μ_{10}	μ_{20}	μ_1	μ_{10}	μ_{20}	μ_1	μ_{10}	μ_{20}
Bi-gram	0.11	0.22	0.29	0.07	0.15	0.17	0.08	0.13	0.17	0.06	0.13	0.18
F(Bi-gram)	0.14	0.26	0.37	0.12	0.18	0.19	0.12	0.19	0.25	0.07	0.20	0.25
Smith-Waterman	0.19	0.37	0.47	0.15	0.21	0.29	0.20	0.27	0.36	0.12	0.36	0.41
F(Smith-Waterman)	0.21	0.44	0.52	0.19	0.30	0.45	0.23	0.34	0.48	0.15	0.46	0.51

Table 1: Accuracy of predicting subsequent User Actions from past user Action History using different algorithms. Average MRR values μ_i have been reported for the first, tenth and the last iteration of the evaluation.

that user's Action History for the same task in D_1, where each S_i is a list of suggestions for the next action a_{i+1}, then MRR is given by:

$$MRR(A, \mathcal{S}) = \frac{1}{n-1} \sum_{i=1}^{n-1} \phi_i$$

ϕ_i is the utility function that is defined as follows:

$$\phi_i = \begin{cases} \frac{1}{rank(a_{i+1})}, & \text{if } a_{i+1} \in S_i \\ 0, & \text{otherwise} \end{cases}$$

$rank(a_{i+1})$ is the position of a_{i+1} in the suggestions list S_i. If the *ground truth* action a_{i+1} is not present in S_i, then utility $\phi_i = 0$. Therefore, higher MRR values indicate better suggestion lists, i.e., a higher proportion of User Actions in sequences of D_2 are present in the corresponding suggestion lists generated from D_1, and furthermore the ranks of these actions in the lists are higher.

The evaluation was done in iterations. In the first iteration, only one action sequence (from same user for the same task) in D_1 was used to generate the suggestion lists $\mathcal{S}$ to compare against the corresponding ground-truth action sequence A in D_2; in the second iteration, two action sequences in D_1 were used to generate the suggestion lists. Specifically, a randomly selected action sequence was appended to the Action History of the first iteration before generating the suggestion lists; in the third iteration – three action sequences we used; and so on. Also, in our evaluation, we used a fixed size 5 for all suggestion lists $S_i \in \mathcal{S}$.

3.4.2 ***Results.*** Table 1 shows the MRR values for two algorithms used for generating Action Suggestions: (a) Bigram frequency – predict the next action only based on the current action (as done by Trailblazer [8] for form filling tasks); and (b) Smith Waterman algorithm as described in Section 3.3. $F(\cdot)$ indicates that *ineligible* suggestions were excluded while generating the Suggestion Lists as described in Section 3.3. Also, we report the average MRR values obtained at the first (μ_1), tenth (μ_{10}), and the last (μ_{20}) iterations of evaluation.

The results clearly demonstrate the *benefit of using a sequence of past User Actions (i.e., Smith Waterman) instead of just the most-recent action (i.e., bigram)* for generating suggestions. For example, the users typically refined their search criteria (e.g, by departure time, by airlines, by manufacturer, etc.) if the first few search results did not match their needs. Therefore, the action sequence before applying the search filters typically consisted of alternating user actions of the following two types – (a) look at one or two property values (e.g., price) of an search-result item, and (b) go to the next item. Our Smith-Waterman based approach was able to use this contextual information provided by the most-recent action sequence to suggest filtering the search results; the ranking of this suggestion increased as the user browsed through more search-result items. Bigram frequency based suggestion model on the other hand, could not capture this context, and hence could not suggest filtering the results when the users needed it.

Also, the MRR scores improved as more and more action sequences were appended to the Action History before generating the suggestions (since $\mu_{20} > \mu_{10} > \mu_1$ in all cases), which indicates that the quality of suggestions improve over time with *longer Action History*, i.e., as more User Actions are appended to the Action History. Unsurprisingly, the results were better when filtering was applied in both algorithms. The results also demonstrate the scalability of our approach across similar websites, especially when ineligible suggestions were excluded.

4 USER STUDY

4.1 Participants

We recruited participants for our study through local mailing lists and word-of-mouth. After preliminary screening via phone interviews, we recruited 12 participants (P1 to P12) who were completely blind, and conducted the study at the *Lighthouse Guild* in *New York* city. The inclusion criteria included familiarity with Internet Explorer Web browser and one of the following two screen readers, JAWS and NVDA. None of the participants had any kind of motor impairment. The participants varied in age from 30 to 58 (mean: 37.3, median: 35, SD: 8.3), gender (8 men, 4 women), and web-browsing experience with screen readers (6 adepts, 3 intermediates, and 3 beginners). On average, the participants indicated they browsed the internet for 3.7 hours daily.

4.2 Study Setup

We designed real-world browsing tasks to evaluate SuggestOmatic. These tasks were transactional in nature, requiring a sequence of steps. The websites for these tasks were selected from 2 categories: flight reservation and online shopping. For flight reservation, we chose Priceline and Kayak, and for shopping, we chose Walmart and eBay.On each of these websites, the participants were asked to complete the following 2 tasks: **T1.** Find the search form, fill-out the form with experimenter-provided data (e.g., flight-reservation details, product name) and hit the search button; **T2.** On the search-results page, find the search-result item that satisfies certain experimenter-provided criteria (e.g., number of stops,

layover duration, departure times, etc. for flight reservation; price, rating, vendor, etc., for shopping).

The participants performed these tasks under the following 2 conditions (one website in each category per condition): (i) **Macro.** Participants can use the Dial-based web automation interface to access a sequence of lists of (static) pre-determined Action Suggestions. Each selection of a suggestion will perform the corresponding automation and refresh the interface content with the next set of pre-determined suggestions. Therefore, the system is always in control. This was also the baseline condition that we used to simulate the extant script-based web automation solutions; (ii) **SuggestOmatic.** Participants can use the same Dial-based interface to access the list of dynamically generated Action Suggestions based on past Action History, and select one of them if desired.

Prior to the study, participants were asked to do similar tasks as above on Expedia and Amazon websites using regular screen-reader shortcuts in order to build Action History profiles. Then, they were given sufficient instructions and time (~30 min) to familiarize themselves with both conditions on these websites, while at the same time extending their Action History profile. The Action History built during this practice session was then used as a bootstrap for generating Action Suggestions during actual tasks on unfamiliar task websites – Priceline, Kayak, Walmart and eBay.

To minimize the learning effect, we counterbalanced the ordering of websites and conditions, while ensuring that no two websites belonging to the same category (e.g., flight reservation) was assigned to the same condition. To avoid confounds, the number of suggestions given at any instant in both conditions were set to 5. Also, the cached versions of the websites were used so that all participants interact with the same search results. The macros were designed in a focus group with accessibility experts where the sequence of lists of Action Suggestions, as well as the ranking of each suggestion in every list, were decided for all the websites used in the study.

4.3 Findings

4.3.1 Task Completion time. We measured completion times separately for 2 categories of websites used in the study – *Flight reservation* and *Shopping*.

Flight reservation. We found significant effect of study conditions on completion time for task T2 ($t(11) = 4.9595$, $p = 0.0004$). However, no such statistically significant difference was observed for task T1 ($t(11) = 0.7515$, $p = 0.4681$). The mean completion times for tasks T1 and T2 under each condition are shown in Figure 5. For task T2, when using SuggestOmatic, the mean completion time (122.58s) was reduced by 17.4% compared to that of the baseline Macro condition (148.33s).

Shopping. We found significant effect of study conditions on completion time for task T1($t(11) = 6.916$, $p < 0.001$) and T2 ($t(11) = 7.937$, $p < 0.001$). The mean completion times for tasks T1 and T2 under each condition are also shown in Figure 5. For task T1, when using SuggestOmatic, the average mean completion time (29.75s) was increased by 53.2% compared to that of the Macro condition (19.41s). However, for Task2, the average mean completion time (145.83s) was reduced by 29.0% compared to the baseline Macro condition (205.25s).

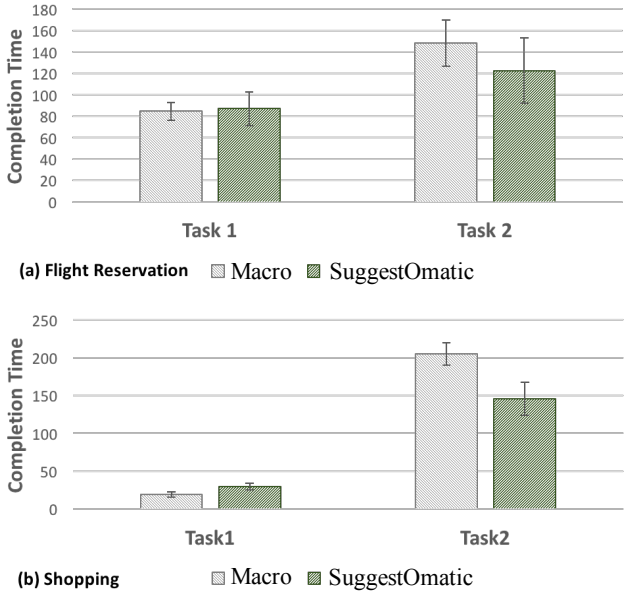

Figure 5: Completion times for 2 tasks using Macro and SuggestOmatic for: (a) flight reservation and (b) shopping. Error bars show +1/-1 SD.

Task T1: With SuggestOmatic, the participants initially spent a little time navigating with shortcuts before requesting suggestions and then choosing the "go to search form" suggestion. However, with Macro, they straightaway went for the suggestions and completed the task. While such user behavior was seen for both flight reservation and shopping tasks, the difference in task completion times is more noticeable in shopping tasks since there was only one form field to fill; in the flight-reservation tasks, this initial overhead of pressing shortcuts was negligible compared to the time spent on filling multiple form fields.

Task T2: For both flight reservation and shopping tasks, under SuggestOmatic condition, the participants cleverly alternated between choosing automation suggestions and pressing screen-reader shortcuts, i.e., use suggestions to navigate between items and then shortcuts to navigate within items. However, under the system-controlled Macro condition, they performed all actions by going through the suggestions list and picking one each time, which took more time while navigating the content within each search-result.

4.3.2 SuggestOmatic Suggestions and User Behavior. The average number of times SuggestOmatic suggestions were requested for Tasks T1 and T2 were 1.123 (SD 0.78) and 6.041 (SD 2.5) respectively. For Task T1, SuggestOmatic was just used to automate navigation to the search form; form filling and navigating between fields were mostly done using screen-reader shortcuts. For Task T2, almost all participants (11 out of 12) chose to request SuggestOmatic for suggestions to navigate between different search-result items. This enabled them to directly jump to the next/previous item without having to listen to the rest of content of a result item that they were not interested in. Only P7 chose to navigate the content of the search results one by one using standard shortcuts.

Also in Task T2, except P3 and P10, all other participants did not go through the entire list of suggestions before making a choice, thereby sometimes missing out on suggestions that could have helped them complete Task T2 faster. For example, based on participants' Action History, the suggestion "go to next flight item" was found to be always ranked higher than the suggestion "go to the duration of the next flight item". Therefore, the participants who hurriedly picked the former suggestion had to do a few extra steps to navigate to the duration property value of the next flight item. Furthermore, 10 out of 12 participants completed task T2 by simply navigating the result items one by one; only P10 and P12 chose suggestions to select different filter options (e.g., brand, airline, departure time, etc.) and narrow down the search results.

We also found significant effect of study conditions on ranking of selected Action Suggestions for Task T2 (Wilcoxon rank-sum test, $Z - score = 7.731$, $p < 0.0001$). No such significant difference was observed for Task T1 ($Z - score = -0.484$, $p = 0.631$). The average ranking of suggestions selected in the SuggestOmatic condition for Tasks T1 and T2 were 1.74 (SD 0.43) and 1.49 (0.94) respectively, compared to 1.65 (SD 0.47) and 2.82 (SD 1.13) in the Macro condition. In Task T1, the suggestions mostly selected were top suggestions such as "go to search form", "go to next field", etc., so there was not much difference between the two conditions. However, in Task T2 that involved navigating result items, the dynamic generation of suggestions (based on Action History) by SuggestOmatic at each step helped push user-desired subsequent actions up the suggestion list, unlike the Macro condition where the ordering of suggestions was statically fixed.

Lastly, the average number of shortcuts used under the SuggestOmatic condition for Task T1 and Task T2 were 9.08 (SD 7.52) and 31.41 (SD 13.07). As mentioned earlier, the shortcuts were used mostly to navigate between form fields in Task T1, and navigate the content (e.g., price, duration, etc.) within search-result items.

4.3.3 Usability Rating. At the end of each study session, every participant was administered the standard System Usability Scale (SUS) questionnaire [13] where they rated positive and negative statements about each study condition on a Likert scale from 1 for strongly disagree to 5 for strongly agree, with 3 being neutral. There was no significant difference in the SUS score between SuggestOmatic ($\mu = 86.6$, $\sigma = 3.72$) and Macro ($\mu = 85.6$, $\sigma = 4.22$) conditions, $t(11) = 0.698$, $p = 0.499$. In fact, the average score of SuggestOmatic is slightly higher than that of Macro. This can be attributed to the majority of participants stating that they could find the suggestions they were looking for relatively quicker in SuggestOmatic compared to Macro, and also being able to alternate between shortcuts and suggestions with SuggestOmatic.

5 DISCUSSION

Our results suggest that overall, participants had a very favorable opinion of the Dial-based SuggestOmatic interface. Interestingly, all participants mentioned that the interface felt like a surrogate mouse. For instance, P2 stated that she could use the Dial to quickly move the screen reader focus to the desired segment through Action Suggestions, akin to how a mouse is used by sighted people to quickly point to the desired Logical Segment on the page.

Almost all participants stated that limitations of screen readers make them restrict their web browsing to a few select websites that they are familiar and comfortable with. Therefore, the participants stated that they were surprised when SuggestOmatic was able to generate suggestions for websites that they had never visited before. After knowing that these suggestions were generated from the activity data collected from other similar websites during practice, they acknowledged that SuggestOmatic could help them increase the range of websites they can interact with comfortably.

Our results also indicate that all participants (except P3) preferred to be in control while performing a web task. They indicated that this enables ad-hoc exploration, which is very important when making decisions (e.g., selecting a flight, buying a product). In this regard, they especially liked the ad-hoc single-step suggestions provided by SuggestOmatic, that they can rely on when they get stuck or lose their orientation in a website. However, a few participants also indicated that multi-step "goal-oriented" macros are more useful in certain scenarios. For example, P3 and P4 stated that using a macro to fill out a form is more reliable, given that they don't have to worry about missing any form field.

Limitations of SuggestOmatic. The primary limitation of SuggestOmatic is its dependence on the initially seeded Action History data for generating suggestions. However, in our study, we observed that data gathered in a short practice time of around 30 minutes was sufficient to generate relevant suggestions; only in 5 times out of 172, the participants did not pick any of the suggestions. Interestingly, all these 5 cases occurred when the participants were browsing through the properties (e.g., flight duration, price, etc.) of search-result items and they wanted to compare these values with the corresponding property values of one of the previous items. Secure and privacy-preserving sharing of Action History data may also help with the cold start issue of SuggestOmatic, and may also help in generating relevant suggestions that may not have been previously recorded in the user's own Action History.

6 CONCLUSION

This paper describes SuggestOmatic, an auto-suggest model for blind users to effortlessly browse the Web. SuggestOmatic executes user's low-level browsing operations on demand, thereby letting them focus on what they want to do rather than how to get it done. To facilitate this, SuggestOmatic leverages user's past Action History, represented as an abstract temporal sequence of operations over logical segments, to identify and proactively suggest a small set of User Actions that will, with high likelihood, contain the action that the user will want to do next. This unique and novel representation of history elevates the level of interaction from operating on (syntactic) HTML elements, as is done now in screen readers, to operating on "semantic" Logical Segments. Besides addressing some of the serious shortcomings underpinning interaction with screen readers, the semantic representation makes SuggestOmatic scalable across web sites. A user study suggests that SuggestOmatic has the potential to transform web accessibility.

7 ACKNOWLEDGEMENT

This work was supported by NSF Award: 1806076, NEI/NIH Awards: R01EY02662, R44EY021962 and NIDILRR Award: 90IF0117-01-00.

REFERENCES

[1] [n. d.]. Website category API. https://www.webshrinker.com/website-category-api/

[2] 1993. *Watch what I do: programming by demonstration.* MIT Press.

[3] Vikas Ashok, Yevgen Borodin, Yuri Puzis, and I. V. Ramakrishnan. 2015. Capti-Speak: A Speech-Enabled Web Screen Reader. In *Proceedings of the 12th Web for All Conference*. ACM, 327–328.

[4] Vikas Ashok, Yevgen Borodin, Svetlana Stoyanchev, Yuri Puzis, and I. V. Ramakrishnan. 2014. Wizard-of-Oz evaluation of speech-driven web browsing interface for people with vision impairments. In *Proceedings of the 11th Web for All Conference*. ACM, 2596699, 1–9. https://doi.org/10.1145/2596695.2596699

[5] Vikas Ashok, Yevgen Borodin, Svetlana Stoyanchev, and I. V. Ramakrishnan. Year. Dialogue Act Modeling for Non-Visual Web Access. In *15th Annual SIGdial Meeting on Discourse and Dialogue (SIGDIAL)*. ACM, 123–132.

[6] Vikas Ashok, Yury Puzis, Yevgen Borodin, and IV Ramakrishnan. 2017. Web Screen Reading Automation Assistance Using Semantic Abstraction. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces*. ACM, 407–418.

[7] Jiang Bian, Anlei Dong, Xiaofeng He, Srihari Reddy, and Yi Chang. 2013. User Action Interpretation for Online Content Optimization. *IEEE Trans. on Knowl. and Data Eng.* 25, 9 (2013), 2161–2174. https://doi.org/10.1109/tkde.2012.130

[8] Jeffrey P. Bigham, Tessa Lau, and Jeffrey Nichols. 2009. Trailblazer: enabling blind users to blaze trails through the web. In *Proceedings of the 13th International Conference on Intelligent User Interfaces*. ACM. https://doi.org/10.1145/1502650.1502677

[9] Syed Masum Billah, Vikas Ashok, Donald E. Porter, and I.V. Ramakrishnan. 2017. Speed-Dial: A Surrogate Mouse for Non-Visual Web Browsing. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, 3132531, 110–119. https://doi.org/10.1145/3132525.3132531

[10] Syed Masum Billah, Vikas Ashok, Donald E. Porter, and IV Ramakrishnan. 2018. SteeringWheel: A Locality-Preserving Magnification Interface for Low Vision Web Browsing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/3173574.3173594

[11] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. 2005. Automation and customization of rendered web pages. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*. ACM, 1095062, 163–172. https://doi.org/10.1145/1095034.1095062

[12] Yevgen Borodin, Jeffrey P. Bigham, Glenn Dausch, and I. V. Ramakrishnan. 2010. More Than Meets the Eye: A Survey of Screen-reader Browsing Strategies. In *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A) (W4A '10)*. ACM, New York, NY, USA, Article 13, 10 pages. https://doi.org/10.1145/1805986.1806005

[13] John Brooke. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in industry* 189 (1996), 194.

[14] Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. 2004. *VIPS: A vision based page segmentation algorithm.* Report. Microsoft technical report.

[15] Josep Domenech, Bernardo de la Ossa, Julio Sahuquillo, Jose A. Gil, and Ana Pont. 2012. Short Survey: A taxonomy of web prediction algorithms. *Expert Syst. Appl.* 39, 9 (2012), 8496–8502. https://doi.org/10.1016/j.eswa.2012.01.140

[16] Darris Hupp and Robert C. Miller. 2007. Smart bookmarks: automatic retroactive macro recording on the web. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*. ACM, 1294226, 81–90. https://doi.org/10.1145/1294211.1294226

[17] JAWS. [n. d.]. Screen reader from Freedom Scientific. http://www.freedomscientific.com/products/fs/jaws-product-page.asp

[18] Jonathan Lazar, Aaron Allen, Jason Kleinman, and Chris Malarkey. 2007. What Frustrates Screen Reader Users on the Web: A Study of 100 Blind Users. *International Journal of human-computer interaction* 22, 3 (2007), 247–269.

[19] Gilly Leshed, Eben M Haber, Tara Matthews, and Tessa Lau. 2008. CoScripter: automating & sharing how-to knowledge in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1719–1728.

[20] Jalal Mahmud, Yevgen Borodin, I. V. Ramakrishnan, and C. R. Ramakrishnan. 2009. Automated construction of web accessibility models from transaction click-streams. In *Proceedings of the 18th International Conference on World Wide Web*. ACM. https://doi.org/10.1145/1526709.1526826

[21] Valentyn Melnyk, Vikas Ashok, Yury Puzis, Andrii Soviak, and Yevgen Borodin. 2014. Widget Classification with Applications to Web Accessibility. In *International Conference on Web Engineering (ICWE)*.

[22] Paula Montoto, Alberto Pan, Juan Raposo, Fernando Bellas, and Javier López. 2009. Automating Navigation Sequences in AJAX Websites. In *Proceedings of the 9th International Conference on Web Engineering*. Springer-Verlag, 1574455, 166–180. https://doi.org/10.1007/978-3-642-02818-2_12

[23] NVDA. [n. d.]. NonVisual Desktop Access. http://www.nvda-project.org/

[24] Venkata N. Padmanabhan and Jeffrey C. Mogul. 1996. Using predictive prefetching to improve World Wide Web latency. *SIGCOMM Comput. Commun. Rev.* 26, 3 (1996), 22–36. https://doi.org/10.1145/235160.235164

[25] Yury Puzis. 2012. Accessible web automation interface: a user study. In *Proceedings of the 14th international ACM SIGACCESS conference on Computers and accessibility*. ACM, 2384999, 291–292. https://doi.org/10.1145/2384916.2384999

[26] Yury Puzis. 2012. An interface agent for non-visual, accessible web automation. In *Adjunct proceedings of the 25th annual ACM symposium on User interface software and technology*. ACM, 2380319, 55–58. https://doi.org/10.1145/2380296.2380319

[27] Yury Puzis, Yevgen Borodin, Faisal Ahmed, Valentyn Melnyk, and I.V. Ramakrishnan. 2011. An Intuitive Accessible Web Automation User Interface. In *Proceedings of the 13th International ACM SIGACCESS Conference on Computers and Accessibility*. ACM.

[28] Yury Puzis, Yevgen Borodin, Rami Puzis, and I.V. Ramakrishnan. 2013. Predictive web automation assistant for people with vision impairments. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, 2488478, 1031–1040. https://doi.org/10.1145/2488388.2488478

[29] T. F. Smith and M. S. Waterman. 1981. Identification of common molecular subsequences. *Journal of Molecular Biology* 147, 1 (1981), 195–197. https://doi.org/10.1016/0022-2836(81)90087-5

[30] Zan Sun, Jalal Mahmud, Saikat Mukherjee, and I. V. Ramakrishnan. 2006. Model-directed web transactions under constrained modalities. In *Proceedings of the 15th international conference on World Wide Web*. ACM. https://doi.org/10.1145/1135777.1135843

[31] SuperNova. [n. d.]. Screen Reader from Dolphin. http://www.yourdolphin.com/productdetail.asp?id=1

[32] VoiceOver. [n. d.]. Screen reader from Apple. https://www.apple.com/accessibility/iphone/vision/

[33] Window-Eyes. [n. d.]. Screen Reader GW Micro. http://www.gwmicro.com/Window-Eyes

[34] Yanhong Zhai and Bing Liu. 2005. Web data extraction based on partial tree alignment. In *Proceedings of the 14th international conference on World Wide Web*. ACM, 76–85.

[35] Jun Zhu, Zaiqing Nie, Ji-Rong Wen, Bo Zhang, and Wei-Ying Ma. 2006. Simultaneous record detection and attribute labeling in web data extraction. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 494–503.

[36] Manuel ĂĄlvarez, Alberto Pan, Juan Raposo, Fernando Bellas, and Fidel Cacheda. 2010. Finding and extracting data records from web pages. *Journal of Signal Processing Systems* 59, 1 (2010), 123–137.