# Automatically adapting web pages to heterogeneous devices

**Chinmay Kulkarni**
Stanford University HCI Group
Stanford, CA 94305-9035
chinmay@cs.stanford.edu

**Scott R Klemmer**
Stanford University HCI Group
Stanford, CA 94305-9035
srk@cs.stanford.edu


Figure 1a: An article from the New York Times as seen on an iPhone


Figure 1b: Output from our algorithm


*CHI 2011*, May 7–12, 2011, Vancouver, BC, Canada.
ACM 978-1-4503-0268-5/11/05.

## Abstract
Smartphones and other handheld devices have become popular and powerful Internet access devices, yet the Web is still largely optimized for the desktop. We describe a system that automatically transforms desktop-optimized pages to ones better suited to the target device. The system leverages existing platform-customized sites as examples of good design, identifies consistent components across these sites, and renders the desktop page into these components.

## Keywords
Mobile phones, Internet access, UI adaptation

## ACM Classification Keywords
H.5.4 [Information Interfaces And Presentation]: Hypertext/Hypermedia—User issues

## General Terms
Algorithms, human factors.

## Introduction
While computing devices are highly heterogeneous – including smartphones, tablets, laptops, large monitors, and wall-scale displays, much of the Web is still optimized for viewing on desktop/laptop screens and input devices. This may be because the desktop is still the

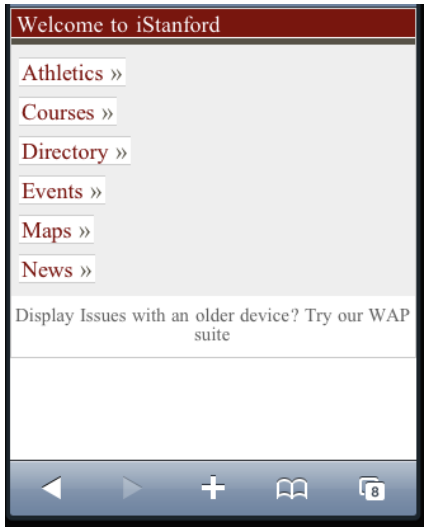



Figure 2 The current mobile version of Stanford University's homepage.

| Category | # of sites |
|---|---|
| News | 7 |
| Weather | 1 |
| Transport | 1 |
| Finance | 5 |
| Reference | 2 |
| Travel | 3 |
| Soc Networking | 3 |
| Health | 1 |
| Shopping | 7 |
| Entertainment | 4 |
| Search | 2 |
| Misc | 5 |
| **Total** | **40** |

Table 1: Distribution of websites used for identifying consistent design elements.

dominant mode of access for most sites. Viewing pages on mobile devices usually requires the user to view a large document through the small viewport of the device screen (see, for e.g. Figure 1(a)). Practitioners have realized that standard (desktop-optimized) versions of pages often offer a degraded experience to a user on other devices; and have tried to solve this problem in two ways.

The first approach is to pick a lowest-common-denominator device and create a minimal website that is accessible from it (*e.g.*, see Figure 2). While such a site is suitable for users on a low-end device, it leads to a degraded user-experience for everyone else.

The second approach is to create a device-specific version (or native application) optimized for a particular mobile device (*e.g.*, the *New York Times* has apps for the iPhone and iPad). However, as handheld devices become increasingly diverse with different capabilities and constraints, this solution may not scale. Furthermore, capabilities of mobile devices have improved rapidly, and will probably continue to do so, so even a customized website could quickly become outdated.
Both approaches also require the designer to constantly update and maintain multiple versions of the site. Few website owners have such resources.

Could we automatically adapt websites to multiple platforms? This work-in-progress describes a data-driven approach to page adaptation. This method is based on the hypothesis that popular websites with existing platform-customized versions provide examples of effective page design.
Three factors motivate this hypothesis. First, popular websites are more likely to be to be designed better. Second, popular websites can optimize their design using the larger amount of data at their disposal. Lastly, users are more likely to be familiar with such a design as they may have previously interacted with these sites.
We implemented this method in a prototype system that transforms pages for smartphones. The prototype is written in Ruby. Given a URL of a target page, the system adapts it on-the-fly and sends the modified page to the device.

## Adapting pages

There are two phases to our page adaptation method. The first needs to be performed once per target device; the other is performed per page. Below, we describe each stage in detail.

*Phase 1: Identify consistent design elements*
We start off by identifying consistent design patterns that are repeated across many websites. For our prototype system, we manually looked at 40 English-language websites that had either iPhone optimized sites or native applications that presented content from their website. The rest of this section treats them equivalently.

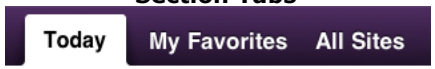
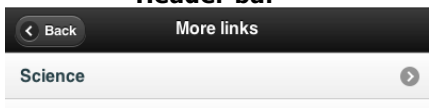
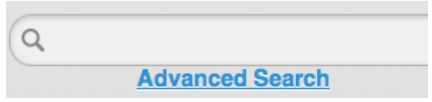
| Component | Description | Partial Order |
|---|---|---|
| **Section Tabs**<br>  | Tabs indicate different sections of the website | 0 |
| **Header bar***<br>  | Usually at top of page, with navigation actions | 1 |
| **Logo**<br>  | Logo for the site or section | 2 |
| **Navigation Bar**<br>  | Usually near the top of the page | 3 |
| **Search field**<br>  | Textfield for site specific search | 4 |
| **Article Text**<br>  | If the page has a single article, the article content | 5 |
| **Link collection**<br>  | A list of links, possibly with thumbnails | 5 |
| **Slideshow**<br>  | Collection of image-description pairs that each link to a different page | 5 |
| **Desktop version toggle** | Link to full, desktop-version of the page | 6 |

Table 2: Consistent components identified. Partial order of 0 refers to top of page.

Kane et al [6] identify categories of web sites frequently accessed by mobile users. Our sites were drawn from these categories. We excluded the Email and Maps categories, however, due to their complexity and because most smart phone platforms already have custom Email and Maps applications. All sites were among Alexa's list of most popular 500 US websites. Several of these sites have also won design awards. Table 1 lists the distribution of these websites across the different domains.

We manually inspected these sites and found nine design elements used consistently across these sites (we consider an element consistent if it was present on 25% or more of the websites). We call each such consistent design element a "component".

Table 2 lists components we obtained and the partial ordering on the page (e.g., the navigation bar appeared before the search field).

This work-in-progress did not attempt to automate this component identification phase, since it only needs to be completed once per target platform and human intervention could help ensure the components identified are meaningful.

*Phase 2: Learn mappings between elements*

Once we have a list of components suitable for the target platform, we now identify which parts of a desktop page correspond to each component.

To do this, we build a recognizer for each component. In our prototype, the recognizer is a support vector machine based classifier. Each classifier is trained with

| Feature name | Description |
|---|---|
| **Numeric features:** contentLength, numLinks, linkLength, numSiblings, numChildren, numComma, numAncestors | Numeric features capture the size of the element (contentLength, numComma), its linkdensity and local DOM structure |
| **Boolean features:** containsImg, containsForm, containsDiv, containsLi HasSpecialName | Boolean features capture whether the element contains specific elements, or has an informative name (e.g. an image with id 'logo') |

Table 3: Features used for SVM classifiers

| Component | Heuristics used |
|---|---|
| Logo | id, class, or image filename contains logo |
| Navigation Bar | id, class contain the word "nav" or "menu", link density is high |
| Article text | score based on length of text, depth in DOM, link-density, and score of child elements |
| Search field | id/action of form contains "search", form method is GET. |
| Link collection | score based on length of element-text, link-density, score of siblings. |

Table 4: Heuristics used to identify components.

human annotated data. For instance, to learn the "navigation bar" classifier, we used a set of pages on which the "navigation bar"s which were previously annotated by hand.

Our training module then takes the set of pages, and breaks each page out into its DOM elements (such as `div`, `img`, `ul`). For each element, it then obtains a feature vector based on its attributes, and whether or not the DOM element is the target component (this acts as the training label). This data is then used to train the classifier. Table 3 lists the features we currently use. The same set of features is used to identify all components.

We create a separate classifier for each component, so the classifier can be reused if the same component is present on multiple target platforms. For instance, navigation bars may be suitable interface elements for both tablets and smartphones, so the classifier can be reused.

Lastly, some components, such as the "desktop version toggle" don't need a classifier since they have no analogue on the desktop, and no data needs to be transferred from the desktop page.

HEURISTIC APPROXIMATIONS

We also explored how effective it would be to use heuristic approximations instead of SVM based classifiers. These classifiers are inspired by projects like Readability (http://lab.arc90.com/experiments/readability/) that identify "content text" on a page using heuristics. We find that these heuristics appear to work reasonably well in practice. We built a set of heuristic classifiers for some of the components we identified (see Table 4). On a set of 30 pages, heuristics identified at least one component on 19 of these pages, the article text and navigation bar classifiers being the most reliable (working on 13 and 11 pages, respectively).

*Page layout*

Transformed components obtained as the output from the previous step are rendered to obtain the platform-optimized version of the page. Our prototype renders the components in a single-column layout based on the partial ordering in Table 2. If more than one component of the same partial order was present, they are rendered in the same order that they appeared in the desktop-optimized page.

## Examples and Limitations

Figure 1 shows a page from the *New York Times*, and the page generated by our prototype. In this case, the system has identified a navigation bar, article text and a search box.


Figure 3: The iPhone homepage as seen on a desktop. The "standard" mobile version scales this page to fit the phone screen. The adapted page primarily uses regions highlighted in red.

Figure 3 shows Apple's iPhone home page (http://www.apple.com/iphone/) as seen on a desktop. Our system identified a navigation bar, search box and a link collection on the page, but failed to locate the Apple logo (Figure 4). Figure 5 shows the navigation pane revealed on clicking the "More" link on the navigation bar.

The current prototype has some limitations: the classifiers do not use any visual features—this affects both the accuracy of the classifiers and the kind of information that can be learned. Rendered pages currently do not use the color scheme of the original page, for instance, since we only extract features from the HTML source.

In addition, there are inherent limitations to the method. First, it relies on the existence of platform-customized examples. Therefore, it cannot generate pages for completely novel platforms, though these platforms could be approximated with a similar platform. Second, components based on page structure alone do not explicitly consider content semantics. Third, we do not utilize information about how users interact with the rendered pages to inform the transformation process. We anticipate that such data (especially click-through and zoom data) would improve the page layout of the rendered page.

## Related work

Prior work for rendering pages on mobile phones falls into two major areas. First, automated methods address technical limitations of the form factor (small screens, low resolution etc) with efficient page thumbnails [8], better page-segmentation [1, 4] and ways to more effectively focus on informative areas of the page [2].

Second, in contrast to these automatic methods are systems that allow users to adapt the content of the website. PageTailor [3] allows smartphone users to customize a page by moving, resizing, or removing page-elements to show only the relevant portions. Viewing a page on PageTailor for the first time involves significant customization effort (around 10 minutes per page), and page changes can cause customizations to break (this occurs around once a month according to the authors). Similarly, Merlion [9] can create mobile mash-ups from desktop sites based on user annotation.

Our method looks at existing pages to learn how best to adapt pages, and implicitly considers both technical limitations of the device and how users interact with pages. Furthermore, our system is automatic and has no customization effort. Lastly, our method is robust to changes in page structure.

Prior work has also explored automatic adaptation of user-interfaces in other applications areas. For instance, SUPPLE [5] adapts interfaces to better serve users with limited motor ability. Systems like SUPPLE are mostly complementary to our own work. Our system focuses on identifying components on a platform and rendering them with data from desktop version of the page. It currently renders a simple, one-column layout for mobile devices. It could be extended to more complex layouts using a system like SUPPLE, which

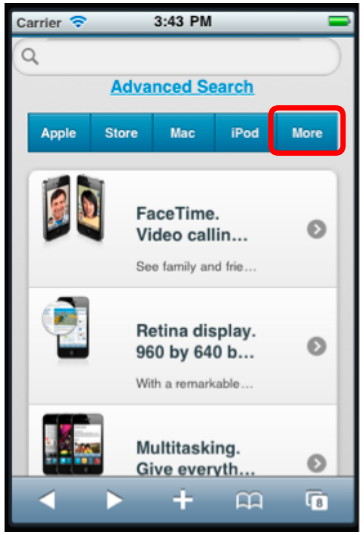


Figure 4: Adapted version of iPhone homepage (from Fig. 3)

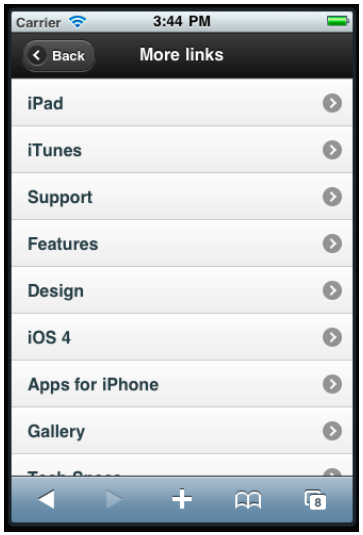


Figure 5: Clicking the "More" link on the navigation bar in Fig 5 reveals

assumes components are known, and lays them out with different screen and input constraints.

Machine learning algorithms have also been previously explored to build "wrappers" and to "understand" and extract information such as addresses and names from web pages (e.g. [10]), and work well for a large number of domains.

*Conclusion*

This work-in-progress presents a novel approach to adapt pages to mobile devices. Using existing pages on the target platform as a source for examples, our approach produce pages that both address the technical limitations of the platform and are based on what users actually do on these devices. We have observed that even a small set of training examples produces reasonable results; and heuristic methods may help rapidly prototype the classifiers used.

We are collecting more training data and building better features (especially visual features, such as those used in [7]) that should improve the performance of the system. After that, the next step is to deploy the prototype as a proxy that automatically transforms pages requested from a mobile device; and use this proxy as an experimental platform to conduct empirical studies.

## Acknowledgments

This research is supported by NSF Grant IIS-0745320.

## References

[1] S. Baluja. Browsing on small screens: recasting web page segmentation into an efficient machine learning framework. In *Proc of the 15th international conf. on World Wide Web*, pages 33–42. ACM, 2006.

[2] Buyukkokten, O., Garcia-Molina, H. and Paepcke, A. Seeing the whole in parts: text summarization for web browsing on handheld devices. In *Proc. of the 10th international conference on World Wide Web*, 2001.

[3] N.Bila, T.Ronda, I.Mohomed, K.Truong, and E. de-Lara. Pagetailor: reusable end-user customization for the mo- bile web. In *Proc. of the 5th international conference on Mobile systems, applications and services*, pages 16–29. ACM, 2007.

[4] Y. Chen, X. Xie, W. Ma, and H. Zhang. Adapting web pages for small-screen devices. *Internet Computing, IEEE*, 9(1):50–56, 2005.

[5] K. Gajos and D. Weld. SUPPLE: automatically generating user interfaces. In *Proceedings of the 9th international conference on Intelligent user interfaces*, pages 93–100. ACM, 2004.

[6] S. Kane, A. Karlson, B. Meyers, P. Johns, A. Jacobs, and G. Smith. Exploring Cross-Device Web Use on PCs and Mobile Devices. In *Proc of the 12th IFIP TC 13 International Conference on Human-Computer Interaction: Part I*, pages 722–735. Springer-Verlag, 2009.

[7] Kumar R., Talton J., Ahmad S. and Klemmer S. Bricolage: A Structured-Prediction Algorithm for Example-Based Web Design. In *Proc. CHI 2011.*

[8] H. Lam and P. Baudisch. Summary thumbnails: readable overviews for small screen web browsers. In *CHI* 2005.

[9] I. Mohomed. Enabling mobile application mashups with Merlion. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*, pages 72–77. ACM, 2010.

[10] Zhu, J., Zhang, B., Nie, Z., Wen, J.R. and Hon, H.W. Webpage understanding: an integrated approach. In *Proc. KDD 2007*.