Bricolage: Example-Based Retargeting for Web Design

Ranjitha Kumar **Jerry O. Talton** **Salman Ahmad** **Scott R. Klemmer**
Stanford University HCI Group
Department of Computer Science
{ranju,jtalton,saahmad,srk}@cs.stanford.edu

ABSTRACT

The Web provides a corpus of design examples unparalleled in human history. However, leveraging existing designs to produce new pages is often difficult. This paper introduces the *Bricolage* algorithm for transferring design and content between Web pages. Bricolage employs a novel, structured-prediction technique that learns to create coherent mappings between pages by training on human-generated exemplars. The produced mappings are then used to automatically transfer the content from one page into the style and layout of another. We show that Bricolage can learn to accurately reproduce human page mappings, and that it provides a general, efficient, and automatic technique for retargeting content between a variety of real Web pages.

Author Keywords
Web design, examples, retargeting, structured prediction.

ACM Classification Keywords
H.5.4 Information interfaces and presentation: Hypertext/ Hypermedia - Theory.

General Terms
Algorithms, Design, Human Factors

INTRODUCTION

Designers in many fields rely on examples for inspiration [17], and examples can facilitate better design work [22]. Examples can illustrate the space of possible solutions and how to implement those possibilities [2, 3]. Furthermore, repurposing successful elements from prior ideas can be more efficient than reinventing them from scratch [12, 21, 14].

The Web provides a corpus of design examples unparalleled in human history: by 2008, Google had indexed more than one trillion unique URLs [1]. However, we hypothesize that this rich resource is underutilized for design tasks. While current systems assist with browsing examples and cloning individual design elements, adapting the gestalt structure of Web designs remains a time-intensive, manual process [22, 10].

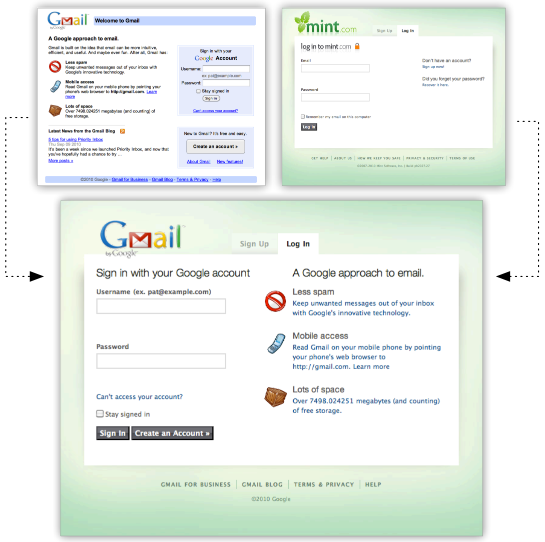

Gmail
Sign Up
Log In
Sign in with your Google account
Username (ex. pat@example.com)
Password
Can't access your account?
Stay signed in
Sign In
Create an Account »
A Google approach to email.
Less spam
Keep unwanted messages out of your inbox with Google's innovative technology.
Mobile access
Read Gmail on your mobile phone by pointing your phone's web browser to http://gmail.com. Learn more
Lots of space
Over 7498.024251 megabytes (and counting) of free storage.
GMAIL FOR BUSINESS | GMAIL BLOG | TERMS & PRIVACY | HELP
©2010 Google

Figure 1. Bricolage computes coherent mappings between Web pages by matching visually and semantically similar page elements. The produced mapping can then be used to guide the transfer of content from one page into the design and layout of the other.

Most design reuse today is accomplished with templates [13]. Templates use standardized page semantics to render content into predesigned layouts. This strength is also a weakness: templates homogenize page structure, limit customization and creativity, and yield cookie-cutter designs. Ideally, tools should offer both the ease of templates *and* the diversity of the entire Web. What if *any* Web page could be a design template?

This paper introduces the *Bricolage* algorithm for transferring design and content between Web pages. The term "bricolage" refers to the creation of a work from a diverse range of things that happen to be available. Bricolage matches visually and semantically similar elements in pages to create coherent mappings between them. These mappings can then be used to automatically transfer the content from one page into the style and layout of the other (Figure 1).

Bricolage uses structured prediction [7] to learn how to transfer content between pages. It trains on a corpus of human-generated mappings, collected using a Web-based crowdsourcing interface, the *Bricolage Collector*. The Collector was seeded with 50 popular Web pages that were decomposed into a visual hierarchy by a novel, constraint-based page segmentation algorithm, *Bento*. In an online study, 39 participants with some Web design experience specified correspondences between page regions and answered free-response questions about their rationale.

These mappings guided the design of Bricolage's matching algorithm. We found consistent structural patterns in how people created mappings between pages. Participants not only identified elements with similar visual and semantic properties, but also used their location in the pages' hierarchies to guide their assignments. Consequently, Bricolage employs a novel tree-matching algorithm that flexibly balances visual, semantic, and structural considerations. We demonstrate that this yields significantly more human-like mappings.

This paper presents the Bento page segmentation algorithm, the data collection study, the mapping algorithm, and the machine learning method. It then shows results demonstrating that Bricolage can learn to closely produce human mappings. Lastly, it illustrates how Bricolage is useful for a diverse set of design applications: for rapidly prototyping alternatives, retargeting content to alternate form factors such as mobile devices, and measuring the similarity of Web designs.

THE BENTO PAGE SEGMENTATION ALGORITHM

To transfer content between Web pages, Bricolage first segments each page into a hierarchy of salient regions that can be extracted and manipulated. The page's Document Object Model (DOM) tree, which describes the page's content, structure, and style, provides a convenient starting point for this segmentation [18].

Existing page segmentation algorithms begin by partitioning the DOM into discrete, visually-salient regions [4, 5, 19]. These algorithms produce good results whenever a page's DOM closely mirrors its visual hierarchy, which is the case for many simple Web pages.

However, these techniques fail on more complex pages. Modern CSS allows content to be arbitrarily repositioned, meaning that the structural hierarchy of the DOM may only loosely approximate the page's visual layout. In our 50-page corpus, we found disparities between the DOM and the visual layout an average of 2.3 times per page. Similarly, inlined text blocks are not assigned individual DOM elements, and therefore cannot be separated from surrounding markup. In practice, these issues render existing segmentation algorithms poorly suited to real-world Web pages.

This paper introduces Bento, a page segmentation algorithm that "re-DOMs" the input page in order to produce clean and consistent segmentations. The algorithm comprises four stages. First, each inlined element is identified and wrapped inside a `<span>` tag to ensure that all page content is contained within a leaf node of the DOM tree. Next, the hierarchy is reshuffled so that parent-child relationships in the tree correspond to visual containment on the page. Each DOM node is labeled with its rendered page coordinates and checked to verify that its parent is the smallest region that

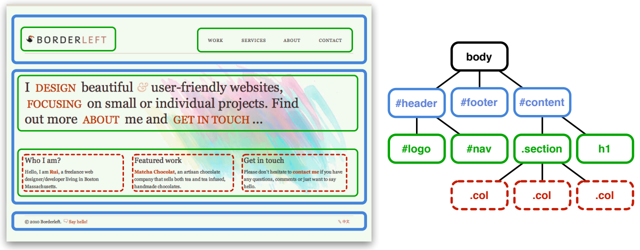

BORDERLEFT
WORK
SERVICES
ABOUT
CONTACT
I DESIGN beautiful & user-friendly websites, FOCUSING on small or individual projects. Find out more ABOUT me and GET IN TOUCH ...
Who I am?
Featured work
Get in touch
body
#header
#footer
#content
#logo
#nav
.section
h1
.col
.col
.col

Figure 2. *Left:* The colored boxes illustrate Bento's segmentation. *Right:* Bento's output tree and associated DOM nodes for this page.

contains it. When this constraint is violated, the DOM is adjusted accordingly, taking care to preserve layout details when nodes are reshuffled. Third, redundant and superfluous nodes that do not contribute to the visual layout of the page are removed. Fourth, the hierarchy is supplemented to introduce missing visual structures. These structural elements are added by computing horizontal and vertical separators across each page region and inserting enclosing DOM nodes accordingly, similar to VIPS [4]. At the end of these four steps, all page content is assigned to a leaf node in the DOM tree, and every non-leaf node contains its children in screen space (Figure 2).

Bento is available as a web service and a BSD open-source C++ library at `http://hci.stanford.edu/bento`.

COLLECTING AND ANALYZING HUMAN MAPPINGS

Retargeting Web pages is closely related to automatic document layout and UI generation. In these domains, the state-of-the-art is constraint-based synthesis [18, 11], which begins with the designer building an abstract data model for each individual class of designs. While this strategy works well in highly-structured domains, the heterogeneous nature of the Web makes model construction impracticable.

We hypothesized that a more general retargeting scheme could be produced by training a machine learning algorithm on human-generated mappings between pages. To this end, we created the Bricolage Collector, a Web application for gathering human page mappings from online workers. We deployed the Collector online, and analyzed the produced mappings to understand how people map Web pages.

Study Design

We selected a diverse corpus of 50 popular Web pages chosen from the Alexa Top 100, Webby award winners, highly-regarded design blogs, and personal bookmarks. Within this corpus, we selected a focus set of eight page pairs. Each participant was asked to match one or two pairs from the focus set, and one or two more chosen uniformly at random from the corpus as a whole. The Collector gathered data about how different people map the same pair of pages, and about how people map many different pairs. We recruited 39 participants through email lists and online advertisements. Each reported some Web design experience.

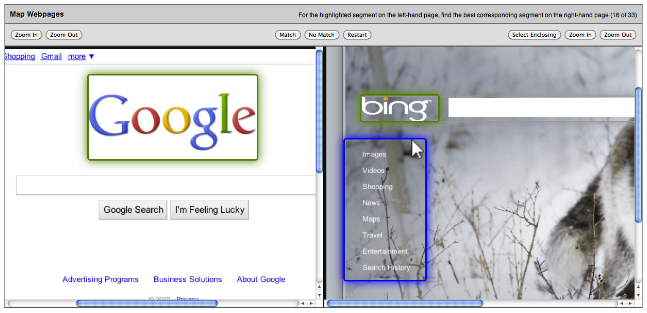


Figure 3. The Bricolage Collector Web application asks users to match each highlighted region in the *left* (content) page to the corresponding region in the *right* (layout) page.

Procedure

Participants watched a tutorial video demonstrating the Collector interface and describing the task. The video instructed participants to produce mappings for transferring the left page's *content* into the right page's *layout*. It emphasized that participants could use any criteria they deemed appropriate to match elements. After the tutorial, the Collector presented participants with the first pair of pages (Figure 3).

The Collector interface iterates over the segmented regions in the content page one at a time, asking participants to find a matching region in the layout page. The user selects a matching region via the mouse or keyboard, and confirms it by clicking the MATCH button. If no good match exists for a particular region, the user clicks the NO MATCH button. After every fifth match, the interface presents a dialog box asking,"Why did you choose this assignment?" These rationale responses are logged along with the mappings, and submitted to a central server.

Results

Participants generated 117 mappings between 52 unique pairs of pages: 73 mappings for the 8 pairs in the focus set, and 44 covering the rest of the corpus. They averaged 10.5 seconds finding a match for each page region ($\min = 4.42s$, $\max = 25.0s$), and 5.38 minutes per page pair ($\min = 1.52m$, $\max = 20.7m$). Participants provided rationales for 227 individual region assignments, averaging 4.7 words in length.

Consistency

We define the consistency of two mappings for the same page pair as the percentage of page regions with identical assignments. The average inter-mapping consistency of the focus pairs was 78.3% ($\min = 58.8\%$, $\max = 89.8\%$). 37.8% of page regions were mapped identically by all participants.

Rationale

Participants provided rationales like "Title of rightmost body pane in both pages." We analyzed these rationales with Latent Semantic Analysis (LSA), which extracts contextual language usage in a set of documents [8]. LSA takes a bag-of-words approach to textual analysis: each document is treated as an unordered collection of words, ignoring grammar and punctuation. We followed the standard approach, treating each rationale as a document and forming the term-document matrix where each cell's value counts the occurrences of a term in a document. We used Euclidean normalization to make annotations of different lengths comparable, and inverse document-frequency weighting to deemphasize common words like *a* and *the*.

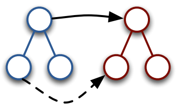
LSA decomposes the space of rationales into semantic "concepts." Each concept is represented by a principal component of the term-document matrix, and the words with the largest projections onto the component are the concept's descriptors.

For the first component, the words with the largest projections are: *footer*, *link*, *menu*, *description*, *videos*, *picture*, *login*, *content*, *image*, *title*, *body*, *header*, *search*, and *graphic*. These words pertain primarily to visual and semantic attributes of page content.

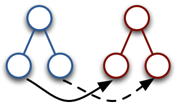
For the second component, the words with the largest projections are: *both*, *position*, *about*, *layout*, *bottom*, *one*, *two*, *three*, *subsection*, *leftmost*, *space*, *column*, *from*, and *horizontal*. These words are mostly concerned with structural and spatial relationships between page elements.

Structure and Hierarchy

Two statistics examine the mappings' structural and hierarchical properties: one measuring how frequently the mapping preserves *ancestry*, and the other measuring how frequently it preserves *siblings*.



We define two matched regions to be *ancestry preserving* if their parent regions are also matched (see inset). A mapping's degree of ancestry preservation is the number of ancestry-preserving regions divided by the total number of matched regions. Participants' mappings preserved ancestry 53.3% of the time ($\min = 7.6\%$, $\max = 95.5\%$).



Similarly, we define a set of page regions sharing a common parent to be *sibling preserving* if the regions they are matched to also share a common parent (see inset). Participants produced mappings that were 83.9% sibling preserving ($\min = 58.3\%$, $\max = 100\%$).

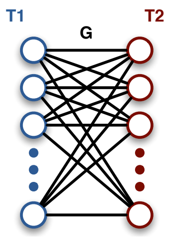
COMPUTING PAGE MAPPINGS

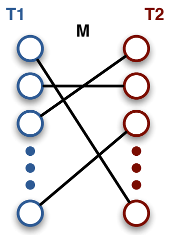
The study's results suggest that mappings produced by different people are highly consistent: there is a "method to the madness" that may be learned. Moreover, the results suggest that algorithmically producing human-like mappings requires incorporating both semantic and structural constraints, and learning how to balance between them.

Prior work in mapping HTML documents presents two distinct approaches. The first ignores structural relationships between DOM nodes and maps elements irrespective of their locations in the pages' visual hierarchy [16]. The second uses tree-matching techniques [20], which strictly preserve hierarchical relationships: once two nodes have been placed in correspondence, their descendants must be matched as well [28, 24, 27]. The results from our study suggest that neither extreme is desirable.

Bricolage introduces a novel optimization algorithm which flexibly balances semantic and structural constraints. The algorithm connects the nodes of the two page trees to form a complete bipartite graph, and for each edge, assigns a cost comprising three terms. The first term measures visual and semantic differences between the corresponding page elements, the second penalizes edges that violate ancestry relationships, and the third penalizes edges that break up sibling groups. Determining the best page mapping then reduces to finding a minimum-cost matching of the constructed graph. Bricolage uses structured prediction to learn a cost function under which the set of exemplar mappings are minimal [7].

Formally, given two page trees with nodes T_1 and T_2, we construct a complete bipartite graph G between $T_1 \cup \{\otimes_1\}$ and $T_2 \cup \{\otimes_2\}$, where $\otimes_1$ and $\otimes_2$ are *no-match* nodes. These two no-match nodes enable the model to track which nodes in one tree have no counterpart in the other. We then define a page mapping M to be a set of edges from G such that every node in $T_1 \cup T_2$ is covered by precisely one edge. In this paper, given a tree node m, $M(m)$ denotes its image (*i.e.*, its counterpart in the other tree). The algorithm assigns a cost $c(e)$ to each edge $e \in M$, and aggregates them to compute the total mapping cost $c(M) = \sum_{e \in M} c(e)$. Bricolage then searches for the least-cost mapping $M^\star = \text{argmin}_M\, c(M)$.





Exact Edge Costs

We define the cost of an edge $e \in T_1 \times T_2$ to be the sum of the visual, ancestry, and sibling costs

$$c(e) = c_v(e) + c_a(e) + c_s(e).$$

For the edges in G connecting tree nodes to no-match nodes, we fix the cost $c(e) = w_n$, where w_n is a constant no-match weight. The edge between the two no-match nodes is assigned a cost of 0 to prevent it from influencing the final mapping.

To compute $c_v([m,n])$, the algorithm compares visual and semantic properties of m and n by inspecting their DOM nodes. The *Learning the Cost Model* section describes this computation in detail.

The ancestry cost $c_a(\cdot)$ penalizes edges that violate ancestry relationships between the pages' elements. Consider a node $m \in T_1$, and let $C(m)$ denote the children of m. We define the *ancestry-violating children* of m, $V(m)$, to be the set of

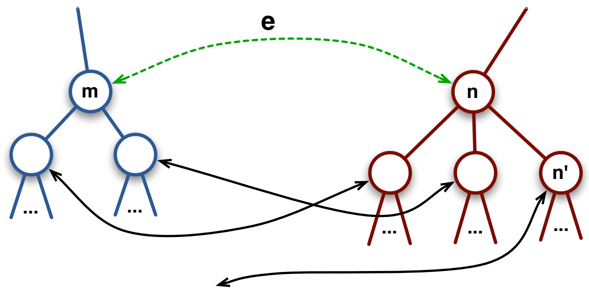


Figure 4. To determine the ancestry penalty for an edge $e = [m,n]$, Bricolage counts the children of m and n which induce ancestry violations. In this example, n' is an ancestry-violating child of n because it is not mapped to a child of m; therefore, n' induces an ancestry cost on e.

m's children that map to nodes that are not $M(m)$'s children, *i.e.,*

$$V(m) = \{m' \in C(m) \mid M(m') \in T_2 \setminus C(M(m))\}\,,$$

and define $V(n)$ symmetrically. Then, the ancestry cost for an edge is proportional to the number of ancestry violating children of its terminal nodes

$$c_a([m,n]) = w_a\,(|V(m)| + |V(n)|)\,,$$

where w_a is a constant ancestry violation weight (see Figure 4).

The sibling cost $c_s(\cdot)$ penalizes edges that fail to preserve sibling relationships between trees. To calculate this term, we first define a few tree-related concepts. Let $P(m)$ denote the parent of m. Then, the *sibling group* of a node m is the set comprising the children of its parent: $S(m) = \{C(P(m))\}$. Given a mapping M, the *sibling-invariant subset* of m, $I(m)$, is the set of nodes in m's sibling group that map to nodes in $M(m)$'s sibling group, *i.e.,*

$$I(m) = \{m' \in S(m) \mid M(m') \in S(M(m))\}\,;$$

the *sibling-divergent subset* of m, $D(m)$, is the set of nodes in m's sibling group that map to nodes in T_2 not in $M(m)$'s sibling group, *i.e.,*

$$D(m) = \{m' \in S(m) \setminus I(m) \mid M(m') \in T_2\}\,;$$

and the set of *distinct sibling families* that m's sibling group maps into is

$$F(m) = \bigcup_{m' \in S(m)} P(M(m')).$$

We define all corresponding terms for n symmetrically, and then compute the total sibling cost:

$$c_s([m,n]) = w_s \left(\frac{|D(m)|}{|I(m)||F(m)|} + \frac{|D(n)|}{|I(n)||F(n)|} \right),$$

where w_s is a constant sibling violation weight. The two ratios increase when siblings are broken up (*i.e.,* their images have different parents), and decrease when more siblings are kept together (see Figure 5).

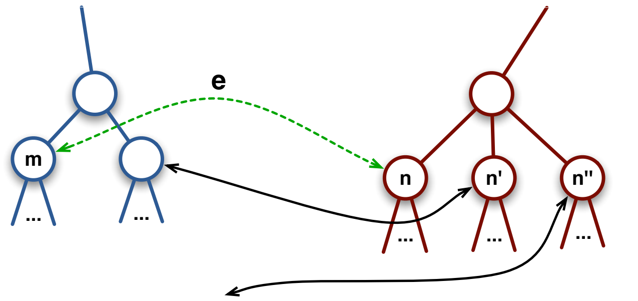


Figure 5. To determine the sibling penalty for an edge $e = [m, n]$, Bricolage computes the sibling-invariant and sibling-divergent subsets of m and n. In this example, $I(n) = \{n'\}$ and $D(n) = \{n''\}$; therefore, n' decreases the sibling cost on e and n'' increases it.

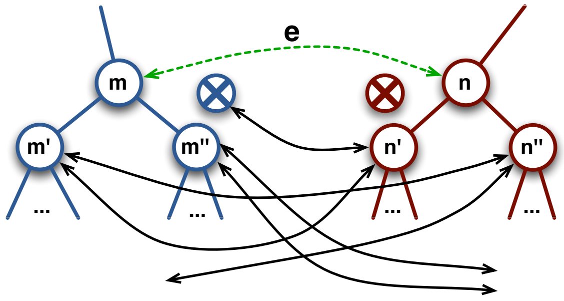


Figure 6. To bound $c_a([m, n])$, observe that neither m' nor n' can induce an ancestry violation. Conversely, m'' is guaranteed to violate ancestry. No guarantee can be made for n''. Therefore, the lower bound for c_a is w_a, and the upper bound is $2w_a$.

Bounding Edge Costs

While this cost model balances semantic, ancestral, and sibling constraints, it cannot be used to search for the optimal mapping $M^\star$ directly. Although $c_v([m, n])$ can be evaluated for an edge by inspecting m and n, $c_a(\cdot)$ and $c_s(\cdot)$ require information about the other edges in the mapping.

While we cannot precisely evaluate $c_a(\cdot)$ and $c_s(\cdot)$ *a priori*, we can compute bounds for them on a per-edge basis [6]. Moreover, each time we accept an edge $[m, n]$ into M, we can remove all the other edges incident on m and n from G. Each time we prune an edge in this way, the bounds for other nearby edges may be improved. Therefore, we employ a Monte Carlo algorithm to approximate $M^\star$, stochastically fixing an edge in G, pruning away the other edges incident on its nodes, and updating the bounds on those that remain.

To bound the ancestry cost of an edge $[m, n] \in G$, we must consider each child of m and n and answer two questions. First, is it *impossible* for this node to induce an ancestry violation? Second, is it *unavoidable* that this node will induce an ancestry violation? The answer to the first question informs the upper bound for $c_a(\cdot)$; the answer to the second informs the lower.

A node $m' \in C(m)$ can induce an ancestry violation if there is some edge between it and a node in $T_2 \setminus (C(n) \cup \{\otimes_2\})$. Conversely, m' is not *guaranteed* to induce an ancestry violation if some edge exists between it and a node in $C(n) \cup \{\otimes_2\}$. Accordingly, we define indicator functions

$$\mathbf{1}_a^{\mathcal{U}}(m', n) = \begin{cases} 1 & \text{if } \exists [m', n'] \in G \text{ s.t. } n' \notin C(n) \cup \{\otimes_2\} \\ 0 & \text{else} \end{cases},$$

$$\mathbf{1}_a^{\mathcal{L}}(m', n) = \begin{cases} 1 & \text{if } \nexists [m', n'] \in G \text{ s.t. } n' \in C(n) \cup \{\otimes_2\} \\ 0 & \text{else} \end{cases}.$$

Then, the upper and lower bounds for $c_a([m, n])$ are

$$\mathcal{U}_a([m,n]) = w_a \left(\sum_{m' \in C(m)} \mathbf{1}_a^{\mathcal{U}}(m', n) + \sum_{n' \in C(n)} \mathbf{1}_a^{\mathcal{U}}(n', m) \right),$$

and

$$\mathcal{L}_a([m,n]) = w_a \left(\sum_{m' \in C(m)} \mathbf{1}_a^{\mathcal{L}}(m', n) + \sum_{n' \in C(n)} \mathbf{1}_a^{\mathcal{L}}(n', m) \right).$$

Figure 6 illustrates the computation of these bounds. Pruning edges from G causes the upper bound for $c_a([m, n])$ to decrease, and the lower bound to increase.

Similarly, we can bound $c_s([m, n])$ by bounding the number of divergent siblings, invariant siblings, and distinct families: $|D(\cdot)|$, $|I(\cdot)|$, and $|F(\cdot)|$. Let $\bar{S}(m) = S(m) \setminus \{m\}$ and consider a node $m' \in \bar{S}(m)$. It is possible that m' is in $D(m)$ as long as some edge exists between it and a node in $T_2 \setminus (\bar{S}(n) \cup \{\otimes_2\})$. Conversely, m' cannot be guaranteed to be in $D(m)$ as long as some edge exists between it and a node in $\bar{S}(n) \cup \{\otimes_2\}$. Then, we have

$$\mathbf{1}_D^{\mathcal{U}}(m', n) = \begin{cases} 1 & \text{if } \exists [m', n'] \in G \text{ s.t. } n' \notin \bar{S}(n) \cup \{\otimes_2\} \\ 0 & \text{else} \end{cases},$$

$$\mathcal{U}_D(m, n) = \sum_{m' \in \bar{S}(m)} \mathbf{1}_D^{\mathcal{U}}(m', n),$$

and

$$\mathbf{1}_D^{\mathcal{L}}(m', n) = \begin{cases} 1 & \text{if } \nexists [m', n'] \in G \text{ s.t. } n' \in \bar{S}(n) \cup \{\otimes_2\} \\ 0 & \text{else} \end{cases},$$

$$\mathcal{L}_D(m, n) = \sum_{m' \in \bar{S}(m)} \mathbf{1}_D^{\mathcal{L}}(m', n).$$

The bounds for $|I(m)|$ are similarly given by

$$\mathbf{1}_I^{\mathcal{U}}(m', n) = \begin{cases} 1 & \text{if } \exists [m', n'] \in G \text{ s.t. } n' \in \bar{S}(n) \\ 0 & \text{else} \end{cases},$$

$$\mathcal{U}_I(m, n) = 1 + \sum_{m' \in \bar{S}(m)} \mathbf{1}_I^{\mathcal{U}}(m', n),$$

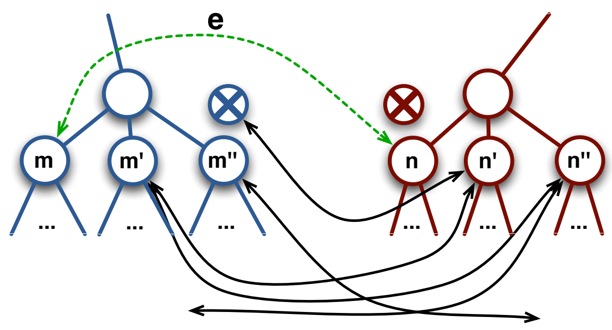


Figure 7. To bound $c_s([m,n])$, observe that m' is guaranteed to be in $I(m)$, and m'' is guaranteed to be in $D(m)$. No guarantees can be made for n' and n''. Therefore, the lower bound for c_s is $w_s/4$, and the upper bound is $3w_s/4$.

and

$$\mathbf{1}_I^{\mathcal{L}}(m',n) = \begin{cases} 1 & \text{if } \forall [m',n'] \in G,\ n' \in \bar{S}(n) \\ 0 & \text{else} \end{cases},$$

$$\mathcal{L}_I(m,n) = 1 + \sum_{m' \in \bar{S}(m)} \mathbf{1}_I^{\mathcal{L}}(m',n).$$

For all nonzero sibling costs, the lower bound for $|F(m)|$ is 2 and the upper bound is $\mathcal{L}_D(m,n)+1$. All remaining quantities are defined symmetrically. Then, upper and lower bounds for $c_s([m,n])$ are given by

$$\mathcal{U}_s([m,n]) = \frac{w_s}{2}\left(\frac{\mathcal{U}_D(m,n)}{\mathcal{L}_I(m,n)} + \frac{\mathcal{U}_D(n,m)}{\mathcal{L}_I(n,m)}\right)$$

and

$$\mathcal{L}_s([m,n]) = w_s\left(\frac{\mathcal{L}_D(m,n)}{\mathcal{U}_I(m,n)\left(\mathcal{L}_D(m,n)+1\right)} + \frac{\mathcal{L}_D(n,m)}{\mathcal{U}_I(n,m)\left(\mathcal{L}_D(n,m)+1\right)}\right).$$

Figure 7 illustrates these computations.

With bounds for the ancestry and sibling terms in place, upper and lower bounds for the total edge cost may be trivially computed as $c_{\mathcal{U}}(e) = c_v(e) + \mathcal{U}_a(e) + \mathcal{U}_s(e)$ and $c_{\mathcal{L}}(e) = c_v(e) + \mathcal{L}_a(e) + \mathcal{L}_s(e)$.

Approximating the Optimal Mapping

To approximate the optimal mapping M^*, we use the Metropolis algorithm [23]. We represent each matching as an ordered list of edges M, and define a Boltzmann-like objective function

$$f(M) = \exp\left[-\beta\, c(M)\right],$$

where β is a constant. At each iteration of the algorithm, a new mapping $\hat{M}$ is proposed, and becomes the new reference mapping with probability

$$\alpha(\hat{M}|M) = \min\left(1, \frac{f(\hat{M})}{f(M)}\right).$$

The algorithm runs for N iterations, and the mapping with the lowest cost is returned.

To initialize M, the bipartite graph G is constructed and the edge bounds initialized. Then, the edges in G are traversed in order of increasing bound. Each edge is considered for assignment to M with some fixed probability γ, until an edge is chosen. If the candidate edge can be fixed and at least one complete matching still exists, it is appended to M, the other edges incident on its terminal nodes are pruned, and the bounds for the remaining edges in G are tightened.

To propose $\hat{M}$, we choose a random index $j \in [1, |M|]$. Then, we re-initialize G, and fix the first j edges in M. To produce the rest of the matching, we repeat the iterative edge selection process described above. In our implementation, we take $\gamma = .7$ and $N = 100$; β is chosen on a per-domain basis, based on the size of the trees.

LEARNING THE COST MODEL

While this mapping algorithm can be used with any visual and semantic cost model and weights w_n, w_a, and w_s, Bricolage seeks to learn a model that will produce human-like mappings. It employs a feature-based approach to compute the visual and semantic cost $c_v(\cdot)$ between nodes, and trains the weights of these features and those for the no-match, ancestry, and sibling terms.

Edge Features

The algorithm computes a set of visual and semantic properties for each node in the page trees. Visual properties are computed using a node's render-time appearance, and include attributes like width, font size, and mean RGB values. Semantic properties take Boolean values, computed by attribute tests such as "is an image" or "is contained in the header." The Appendix gives a full list of these properties.

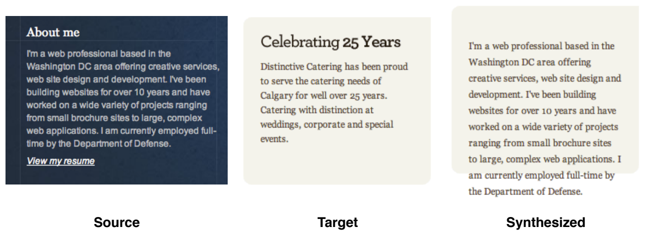
To compute an edge's total cost, the algorithm first calculates the difference between each property for m and n, and concatenates these values—along with the exact ancestry and sibling costs and a Boolean no-match indicator—into a feature vector $\mathbf{f}_e$. Given a set of weights $\mathbf{w}_f$ for each visual and semantic feature, the edge cost is then computed as $c(e) = \bar{\mathbf{w}}^T\mathbf{f}_e$, where $\bar{\mathbf{w}} = \langle \mathbf{w}_f, w_a, w_s, w_n \rangle$.

Given a mapping M, the algorithm assembles an aggregate feature vector $\mathbf{F}_M = \sum_{e \in M} \mathbf{f}_e$ to calculate $c(M) = \bar{\mathbf{w}}^T\mathbf{F}_M$. Training the cost model then reduces to finding a set of weights under which the mappings in the training set have minimal total cost.

Generalized Perceptron Algorithm

To learn a consistent assignment for the weight vector $\bar{\mathbf{w}}$ under which the set of exemplar mappings are minimal, Bricolage uses the generalized perceptron algorithm for structured prediction [7].

The perceptron begins by initializing $\bar{\mathbf{w}}_0 = 0$. In each subsequent iteration, the perceptron randomly selects a pair of page trees and an associated human mapping M from the training set. Next, using the current weight vector $\bar{\mathbf{w}}_i$, it computes a new mapping $\hat{M} \approx \operatorname{argmin}_M \bar{\mathbf{w}}_i^T\mathbf{F}_M$. Based on the resultant mapping, a new aggregate feature



About me

I'm a web professional based in the Washington DC area offering creative services, web site design and development. I've been building websites for over 10 years and have worked on a wide variety of projects ranging from small brochure sites to large, complex web applications. I am currently employed full-time by the Department of Defense.

View my resume

Celebrating 25 Years

Distinctive Catering has been proud to serve the catering needs of Calgary for well over 25 years. Catering with distinction at weddings, corporate and special events.

I'm a web professional based in the Washington DC area offering creative services, web site design and development. I've been building websites for over 10 years and have worked on a wide variety of projects ranging from small brochure sites to large, complex web applications. I am currently employed full-time by the Department of Defense.

Figure 8. A current limitation of the content transfer algorithm illustrating the challenges of HTML/CSS. The target page's CSS prevents the bounding beige box from expanding. This causes the text to overflow (synthesized page). Also, the target page expects all headers to be images. This causes the "About Me" header to disappear (synthesized page). An improved content transfer algorithm could likely address both of these issues.

vector $\mathbf{F}_{\hat{M}}$ is calculated, and the weights are updated by $\bar{\mathbf{w}}_{i+1} = \bar{\mathbf{w}}_i + \alpha_i \left(\mathbf{F}_{\hat{M}} - \mathbf{F}_M\right)$, where $\alpha_i = 1/\sqrt{i+1}$ is the learning rate.

The perceptron algorithm is only *guaranteed* to converge if the training set is linearly separable; in practice, it produces good results for many diverse data sets [7]. Since the weights may oscillate during the final stages of the learning, the final cost model is produced by averaging over the last few iterations.

CONTENT TRANSFER

Once a cost model is trained, it is fed to the matching algorithm, which uses it to predict mappings between any two pages. Bricolage then uses these computed mappings to automatically transfer the content from one page into the style and layout of another. In its segmented page representation, page content (text, images, links, form fields) lives on the leaf nodes of the page tree. Before transferring content, the inner HTML of each node in the source page is preprocessed to inline CSS styles and convert embedded URLs to absolute paths. Then, content is moved between mapped nodes by replacing the inner HTML of the target node with the inner HTML of the source node.

Content matched to a no-match node can be handled in one of two ways. In the simplest case, unmatched source nodes are ignored. However, if important content in the source page is not mapped, it may be more desirable to insert the unmatched node into the target page parallel to its mapped siblings, or beneath its lowest mapped ancestor.

This approach works well for many pages. Occasionally, the complexity and diversity of modern Web technologies pose practical challenges to resynthesizing coherent HTML. Many pages specify style rules and expect certain markup patterns, which may cause the new content to be rendered incorrectly (Figure 8). Furthermore, images and plugin objects (*e.g.,* Flash, Silverlight) have no CSS style information that can be borrowed; when replaced, the new content will not exhibit the same visual appearance and may seem out of place. Lastly, embedded scripts are often tightly coupled with the original page's markup and break when naïvely transferred. Consequently, the current implementation ignores them, preventing dynamic behavior from being borrowed. A more robust content transfer algorithm is required to address these issues and remains future work.

RESULTS

We demonstrate the efficacy of Bricolage in two ways. First, we show several practical examples of Bricolage in action. Second, we evaluate the machine learning components by performing a hold-out cross-validation experiment on the gathered human mappings.

Examples

Figure 10 demonstrates the algorithm in a rapid prototyping scenario, in which an existing page is transformed into several potential replacement designs. Creating multiple alternatives facilitates comparison, team discussion, and design space exploration [9, 15, 25]. Figure 11 demonstrates that Bricolage can be used to retarget content across form factors, showing a full-size Web page automatically mapped into two different mobile layouts.

Figure 9 illustrates an ancillary benefit of Bricolage's cost model. Since Bricolage searches for the optimal mapping between pages, the returned cost can be interpreted as an approximate distance metric on the space of page designs. Although the theoretical properties of this metric are not strong (it satisfies neither the triangle inequality nor the identity of indiscernibles), in practice it may provide a useful mechanism for automatically differentiating between pages with similar and dissimilar designs.

Machine Learning Results

To test the effectiveness of Bricolage's machine learning components, we ran a hold-out test. We used the 44 collected mappings outside the focus set as training data, and

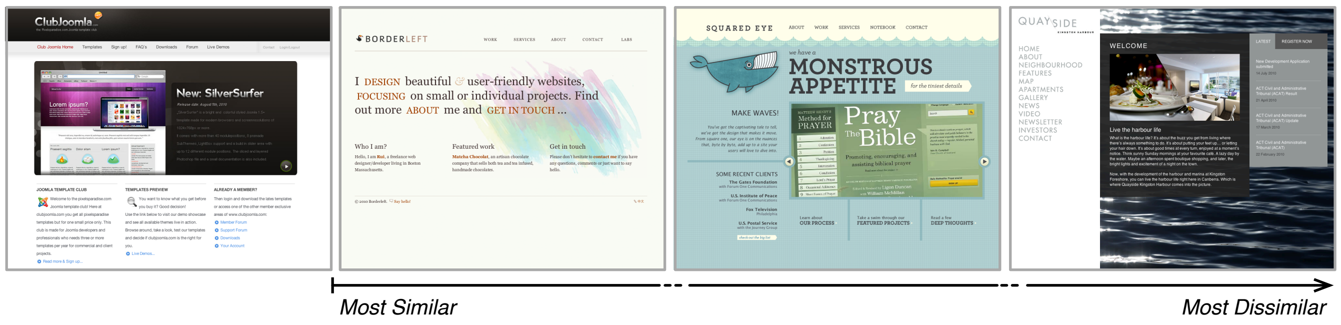
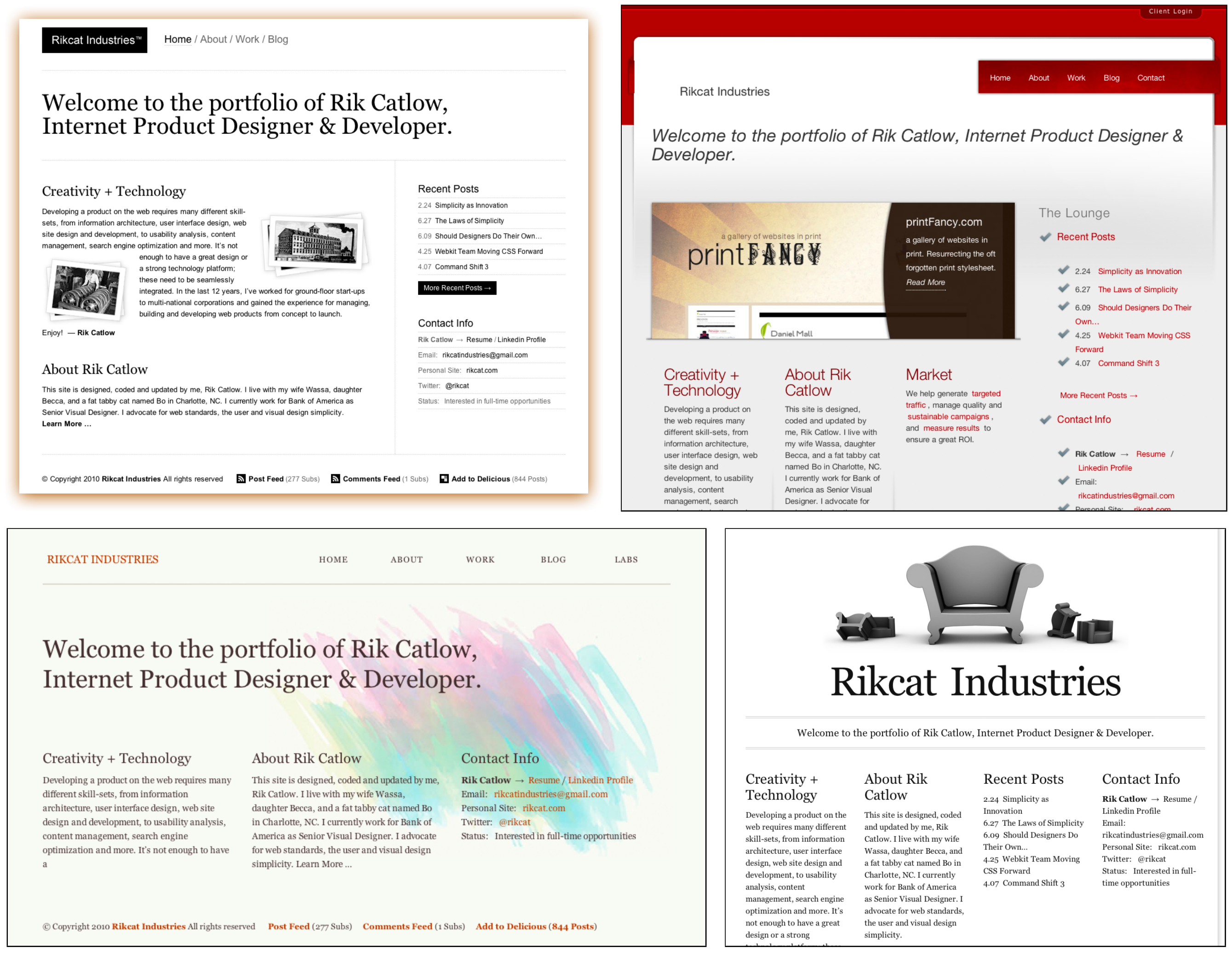


Figure 9. Bricolage can be used to induce a distance metric on the space of Web designs. By mapping the leftmost page onto each of the pages in the corpus and examining the mapping cost, we can automatically differentiate between pages with similar and dissimilar designs.


Rikcat Industries™ Home / About / Work / Blog

Welcome to the portfolio of Rik Catlow, Internet Product Designer & Developer.

Creativity + Technology

Developing a product on the web requires many different skill-sets, from information architecture, user interface design, web site design and development, to usability analysis, content management, search engine optimization and more. It's not enough to have a great design or a strong technology platform; these need to be seamlessly integrated. In the last 12 years, I've worked for ground-floor start-ups to multi-national corporations and gained the experience for managing, building and developing web products from concept to launch.

Enjoy! — Rik Catlow

About Rik Catlow

This site is designed, coded and updated by me, Rik Catlow. I live with my wife Wassa, daughter Becca, and a fat tabby cat named Bo in Charlotte, NC. I currently work for Bank of America as Senior Visual Designer. I advocate for web standards, the user and visual design simplicity. Learn More ...

Recent Posts
2.24 Simplicity as Innovation
6.27 The Laws of Simplicity
6.09 Should Designers Do Their Own...
4.25 Webkit Team Moving CSS Forward
4.07 Command Shift 3
More Recent Posts →

Contact Info
Rik Catlow → Resume / Linkedin Profile
Email: rikcatindustries@gmail.com
Personal Site: rikcat.com
Twitter: @rikcat
Status: Interested in full-time opportunities

© Copyright 2010 Rikcat Industries All rights reserved | Post Feed (277 Subs) | Comments Feed (1 Subs) | Add to Delicious (844 Posts)

Client Login
Home About Work Blog Contact
Rikcat Industries

Welcome to the portfolio of Rik Catlow, Internet Product Designer & Developer.

printFancy.com
a gallery of websites in print. Resurrecting the oft forgotten print stylesheet.
Read More

The Lounge
Recent Posts
2.24 Simplicity as Innovation
6.27 The Laws of Simplicity
6.09 Should Designers Do Their Own...
4.25 Webkit Team Moving CSS Forward
4.07 Command Shift 3
More Recent Posts →
Contact Info
Rik Catlow → Resume / Linkedin Profile
Email: rikcatindustries@gmail.com
Personal Site: rikcat.com

Creativity + Technology
Developing a product on the web requires many different skill-sets, from information architecture, user interface design, web site design and development, to usability analysis, content management, search

About Rik Catlow
This site is designed, coded and updated by me, Rik Catlow. I live with my wife Wassa, daughter Becca, and a fat tabby cat named Bo in Charlotte, NC. I currently work for Bank of America as Senior Visual Designer. I advocate for

Market
We help generate targeted traffic, manage quality and sustainable campaigns, and measure results to ensure a great ROI.

RIKCAT INDUSTRIES HOME ABOUT WORK BLOG LABS

Welcome to the portfolio of Rik Catlow, Internet Product Designer & Developer.

Creativity + Technology
Developing a product on the web requires many different skill-sets, from information architecture, user interface design, web site design and development, to usability analysis, content management, search engine optimization and more. It's not enough to have a

About Rik Catlow
This site is designed, coded and updated by me, Rik Catlow. I live with my wife Wassa, daughter Becca, and a fat tabby cat named Bo in Charlotte, NC. I currently work for Bank of America as Senior Visual Designer. I advocate for web standards, the user and visual design simplicity. Learn More ...

Contact Info
Rik Catlow → Resume / Linkedin Profile
Email: rikcatindustries@gmail.com
Personal Site: rikcat.com
Twitter: @rikcat
Status: Interested in full-time opportunities

© Copyright 2010 Rikcat Industries All rights reserved Post Feed (277 Subs) Comments Feed (1 Subs) Add to Delicious (844 Posts)

Rikcat Industries

Welcome to the portfolio of Rik Catlow, Internet Product Designer & Developer.

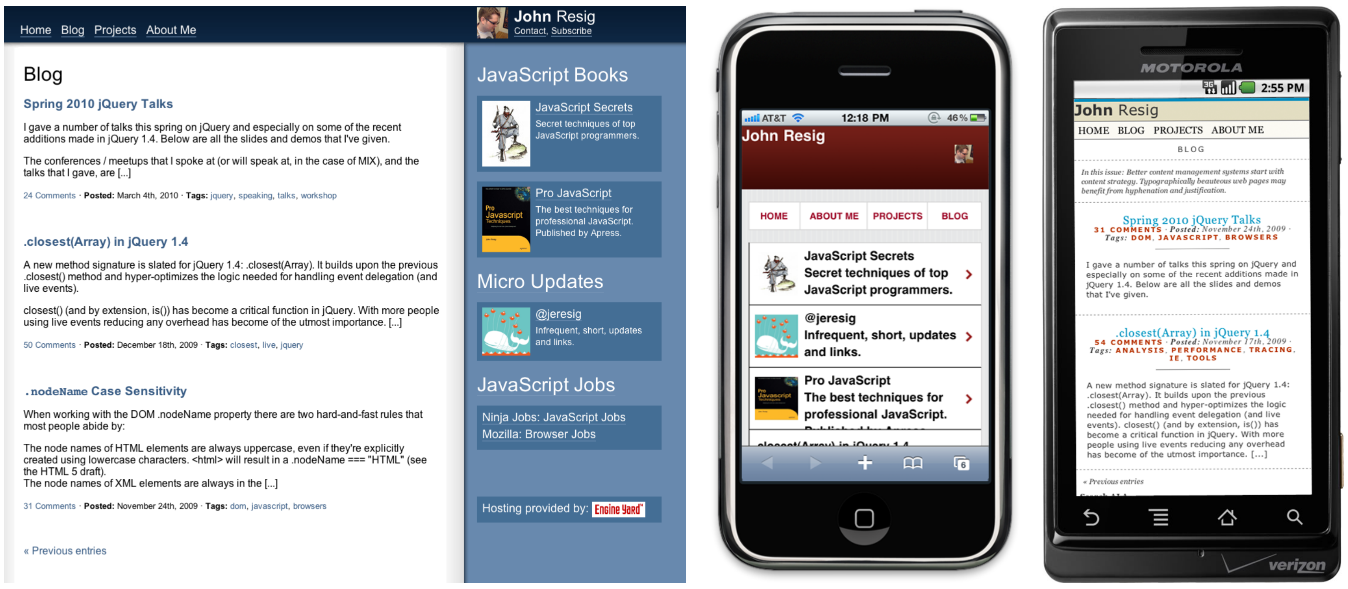
Creativity + Technology
Developing a product on the web requires many different skill-sets, from information architecture, user interface design, web site design and development, to usability analysis, content management, search engine optimization and more. It's not enough to have a great design or a strong

About Rik Catlow
This site is designed, coded and updated by me, Rik Catlow. I live with my wife Wassa, daughter Becca, and a fat tabby cat named Bo in Charlotte, NC. I currently work for Bank of America as Senior Visual Designer. I advocate for web standards, the user and visual design simplicity.

Recent Posts
2.24 Simplicity as Innovation
6.27 The Laws of Simplicity
6.09 Should Designers Do Their Own...
4.25 Webkit Team Moving CSS Forward
4.07 Command Shift 3

Contact Info
Rik Catlow → Resume / Linkedin Profile
Email: rikcatindustries@gmail.com
Personal Site: rikcat.com
Twitter: @rikcat
Status: Interested in full-time opportunities

Figure 10. Bricolage used to rapidly prototype many alternatives. *Top-left:* the original Web page. *Rest:* the page automatically retargeted to three other layouts and styles.


Home Blog Projects About Me | John Resig | Contact, Subscribe

Blog

Spring 2010 jQuery Talks

I gave a number of talks this spring on jQuery and especially on some of the recent additions made in jQuery 1.4. Below are all the slides and demos that I've given.

The conferences / meetups that I spoke at (or will speak at, in the case of MIX), and the talks that I gave, are [...]

24 Comments · Posted: March 4th, 2010 · Tags: jquery, speaking, talks, workshop

.closest(Array) in jQuery 1.4

A new method signature is slated for jQuery 1.4: .closest(Array). It builds upon the previous .closest() method and hyper-optimizes the logic needed for handling event delegation (and live events).

closest() (and by extension, is()) has become a critical function in jQuery. With more people using live events reducing any overhead has become of the utmost importance. [...]

50 Comments · Posted: December 18th, 2009 · Tags: closest, live, jquery

.nodeName Case Sensitivity

When working with the DOM .nodeName property there are two hard-and-fast rules that most people abide by:

The node names of HTML elements are always uppercase, even if they're explicitly created using lowercase characters. <html> will result in a .nodeName === "HTML" (see the HTML 5 draft).
The node names of XML elements are always in the [...]

31 Comments · Posted: November 24th, 2009 · Tags: dom, javascript, browsers

« Previous entries

JavaScript Books
JavaScript Secrets — Secret techniques of top JavaScript programmers.
Pro JavaScript — The best techniques for professional JavaScript. Published by Apress.

Micro Updates
@jeresig — Infrequent, short, updates and links.

JavaScript Jobs
Ninja Jobs: JavaScript Jobs
Mozilla: Browser Jobs

Hosting provided by: Engine Yard

AT&T 12:18 PM 46%
John Resig
HOME ABOUT ME PROJECTS BLOG
JavaScript Secrets — Secret techniques of top JavaScript programmers.
@jeresig — Infrequent, short, updates and links.
Pro JavaScript — The best techniques for professional JavaScript.
.closest(Array) in jQuery 1.4

MOTOROLA 2:55 PM
John Resig
HOME BLOG PROJECTS ABOUT ME
BLOG
In this issue: Better content management systems start with content strategy. Typographically beauteous web pages may benefit from hyphenation and justification.

Spring 2010 jQuery Talks
31 COMMENTS · Posted: November 24th, 2009 · Tags: DOM, JAVASCRIPT, BROWSERS
I gave a number of talks this spring on jQuery and especially on some of the recent additions made in jQuery 1.4. Below are all the slides and demos that I've given.

.closest(Array) in jQuery 1.4
54 COMMENTS · Posted: November 17th, 2009 · Tags: ANALYSIS, PERFORMANCE, TRACING, IE, TOOLS
A new method signature is slated for jQuery 1.4: .closest(Array). It builds upon the previous .closest() method and hyper-optimizes the logic needed for handling event delegation (and live events). closest() (and by extension, is()) has become a critical function in jQuery. With more people using live events reducing any overhead has become of the utmost importance. [...]

« Previous entries

verizon

Figure 11. Bricolage can retarget Web pages designed for the desktop to mobile devices. *Left:* the original Web page. *Right:* the page automatically retargeted to two different mobile layouts.

the mappings in the focus set as test data. The perceptron was run for 400 iterations, and the weight vector averaged over the last 20. The learned cost model was used to predict mappings for each of the 8 focus pairs. Table 1 shows the comparison between the learned and reference mappings using three different metrics: average similarity, nearest neighbor similarity, and percentage of edges that appear in at least one mapping.

The online mapping experiment found a 78% inter-mapping consistency between the participants. This might be considered a gold standard against which page mapping algorithms are measured. Currently, Bricolage achieves a 69% consistency. By this measure, there is room for improvement. However, Bricolage's mappings overlap an average of 78% with their nearest human neighbor, and 88% of the edges generated by Bricolage appear in some human mapping.

This structured prediction approach was motivated by the hypothesis that ancestry and sibling relationships are crucial to predicting human mappings. We tested this hypothesis by training three additional cost models containing different feature subsets: visual terms only, visual and ancestry terms, and visual and sibling terms. Considering only local features yields an average nearest neighbor match of 53%; mapping with local and sibling features yields 67%; mapping with local and ancestry features yields 75%. Accounting for all of these features yields 78%, a result that dominates that of any subset. In short, flexibly preserving structure is crucial to producing good mappings.

IMPLEMENTATION

Bricolage's page segmentation, mapping, and machine learning libraries are implemented in C++ using the Qt framework, and use Qt's WebKit API in order to interface directly with a browser engine. Once a cost model has been trained, Bricolage produces mappings between pages in about 1.04 seconds on a 2.55 Ghz Intel Core i7, averaging roughly 0.02 seconds per node.

The corpus pages are archived using the Mozilla Archive File Format and hosted on a server running Apache. For efficiency, page segmentations and associated DOM node features are computed and cached for each page when it is added to the corpus. Each feature has its own dynamic plug-in library, allowing the set of features to be extended with minimal overhead, and mixed and matched at runtime. The Bricolage Collector is written in HTML, Javascript, and CSS. Mapping results are sent to a centralized Ruby on Rails server and stored in a SQLite database.

CONCLUSIONS AND FUTURE WORK

This paper introduced the Bricolage algorithm for automatically transferring design and content between Web pages. Bricolage's major algorithmic insight was a technique for capturing the structural relationships between elements, and using an optimization approach to balance local and global concerns. This work takes a first step towards a powerful new paradigm for example-based Web design, and opens up exciting areas for future research.

Metric	*Cost Model*	*%*
Average Similarity	c_v c_a and c_s	68.7
	c_v and c_a	65.8
	c_v and c_s	59.1
	c_v alone	44.6
Nearest Neighbor	c_v c_a and c_s	77.7
	c_v and c_a	75.1
	c_v and c_s	67.0
	c_v alone	53.2
Edge Frequency	c_v c_a and c_s	87.9
	c_v and c_a	84.5
	c_v and c_s	81.4
	c_v alone	64.4

Table 1. Results of the hold-out cross-validation experiment. Bricolage performs substantially worse without both the ancestry and sibling terms in the cost model.

The current prototype employs thirty visual and semantic features. Adding more sophisticated properties—such as those based on computer vision techniques—will likely improve the quality of the machine learning.

Future work could extend example-based design to other domains. The current Bricolage implementation is HTML specific. In principle, the retargeting algorithm can be applied to any document with hierarchical structure such as slide presentation and vector graphics files. With richer vision techniques (along the lines of [26]), the Bricolage approach might extend to documents and interfaces without accessible structure.

Finally, an important next step is to create a retargeting design tool that allows both novice and experts to more creatively use examples. Observing how people use such a tool will provide valuable research knowledge about the role examples can play in amplifying creativity.

ACKNOWLEDGEMENTS

We thank Tim Roughgarden for helpful discussions about the matching algorithm; Chuong Do for helpful discussions about structured prediction techniques; Amy Jang for implementing machine learning features; Juho Kim and Kanya Siangliulue for helping prototype the Bricolage Collector; Mira Dontcheva and the reviewers for their helpful comments. This research was supported in part by NSF Grant IIS-0745320.

REFERENCES

1. Alpert, J., Hajaj, N. We knew the web was big... [online]. 2008. Available from: `http://goo.gl/RtmG` [cited 14 January 2011].

2. Brandt, J., Guo, P. J., Lewenstein, J., Dontcheva, M., Klemmer, S. R. Two studies of opportunistic programming: interleaving Web foraging, learning, and writing code. *Proc. CHI* (2009), ACM, pp. 1589–1598.

3. Buxton, B. *Sketching User Experiences*. Morgan Kaufmann, 2007.
4. Cai, D., Yu, S., Wen, J.-R., Ma, W.-Y. VIPS: a vision-based page segmentation algorithm. Tech. Rep. MSR-TR-2003-79, Microsoft, 2003.
5. Chakrabarti, D., Kumar, R., Punera, K. A graph-theoretic approach to Webpage segmentation. *Proc. WWW* (2008), pp. 377–386.
6. Chawathe, S. S., Garcia-Molina, H. Meaningful change detection in structured data. *Proc. SIGMOD* (1997), ACM, pp. 26–37.
7. Collins, M. Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. *Proc. EMNLP* (2002), ACL.
8. Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., Harshman, R. Indexing by latent semantic analysis. *Journal of the American Society for Information Science 41*, 6 (1990), pp. 391–407.
9. Dow, S. P., Glassco, A., Kass, J., Schwarz, M., Schwartz, D. L., Klemmer, S. R. Parallel prototyping leads to better design results, more divergence, and increased self-efficacy. *ACM TOCHI 17(4)* (December 2010), pp. 1–24.
10. Fitzgerald, M. CopyStyler: Web design by example. Tech. rep., MIT, May 2008.
11. Gajos, K., Weld, D. S. SUPPLE: automatically generating user interfaces. *Proc. IUI* (2004), ACM.
12. Gentner, D., Holyoak, K., Kokinov, B. *The Analogical Mind: Perspectives From Cognitive Science*. MIT Press, 2001.
13. Gibson, D., Punera, K., Tomkins, A. The volume and evolution of Web page templates. *Proc. WWW* (2005), ACM, pp. 830–839.
14. Hartmann, B., Wu, L., Collins, K., Klemmer, S. R. Programming by a sample: rapidly creating Web applications with d.mix. *Proc. UIST* (2007), ACM.
15. Hartmann, B., Yu, L., Allison, A., Yang, Y., Klemmer, S. R. Design as exploration: creating interface alternatives through parallel authoring and runtime tuning. *Proc. UIST* (2008), ACM.
16. Hashimoto, Y., Igarashi, T. Retrieving web page layouts using sketches to support example-based web design. *2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling* (2005).
17. Herring, S. R., Chang, C.-C., Krantzler, J., Bailey, B. P. Getting inspired!: understanding how and why examples are used in creative design practice. *Proc. CHI* (2009), ACM, pp. 87–96.
18. Hurst, N., Li, W., Marriott, K. Review of automatic document formatting. *Proc. DocEng* (2009), ACM.
19. Kang, J., Yang, J., Choi, J. Repetition-based Web page segmentation by detecting tag patterns for small-screen devices. *IEEE Transactions on Consumer Electronics 56* (May 2010), pp. 980–986.
20. Kim, Y., Park, J., Kim, T., Choi, J. Web information extraction by html tree edit distance matching. *International Conference on Convergence Information Technology.* (2007), pp. 2455–2460.
21. Kolodner, J. L., Wills, L. M. Case-based creative design. *In AAAI Spring Symposium on AI and Creativity* (1993), pp. 50–57.
22. Lee, B., Srivastava, S., Kumar, R., Brafman, R., Klemmer, S. R. Designing with interactive example galleries. *Proc. CHI* (2010), ACM.
23. Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P. *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. Cambridge University Press, New York, NY, USA, 2007.
24. Shasha, D., Wang, J. T.-L., Zhang, K., Shih, F. Y. Exact and approximate algorithms for unordered tree matching. *IEEE Transactions on Systems, Man, and Cybernetics 24*, 4 (1994), pp. 668–678.
25. Talton, J. O., Gibson, D., Yang, L., Hanrahan, P., Koltun, V. Exploratory modeling with collaborative design spaces. *Proc. SIGGRAPH ASIA* (2009), ACM Press.
26. Wattenberg, M., Fisher, D. A model of multi-scale perceptual organization in information graphics. *Proc. INFOVIS* (2003), pp. 23 –30.
27. Zhang, K. A constrained edit distance between unordered labeled trees. *Algorithmica 15*, 3 (1996).
28. Zhang, K., Shasha, D. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput. 18*, 6 (1989), pp. 1245–1262.

APPENDIX

The Bricolage prototype uses the following DOM properties as features in the learning.

The visual properties include: *width*, *height*, *area*, *aspectRatio*, *fontSize*, *fontWeight*, *meanColor*, *numLinks*, *numColors*, *numChildren*, *numImages*, *numSiblings*, *siblingOrder*, *textArea*, *wordCount*, *treeLevel*, *verticalSidedness* (normalized distance from the horizon of the page), *horizontalSidedness* (normalized distance from the midline of the page), *leftSidedness* (normalized distance from the left border of the page), *topSidedness* (normalized distance from the top border of the page), and *shapeAppearance* (the minimum of the aspect ratio and its inverse).

The semantic properties include: *search*, *footer*, *header*, *image*, *logo*, *navigation*, *bottom* (if the node is in the bottom 10% of the page), *top* (if the node is in the top 10% of the page), *fillsHeight* (if the node extends more than 90% down the page), and *fillsWidth* (if the node extends more than 90% across the page).