Bricolage: A Structured-Prediction Algorithm for Example-Based Web Design

Ranjitha Kumar Jerry O. Talton Salman Ahmad Scott R Klemmer
Stanford University*

Abstract

The Web today provides a corpus of design examples unparalleled in human history. However, leveraging existing designs to produce new pages is currently difficult. This paper introduces the *Bricolage* algorithm for automatically transferring design and content between Web pages. Bricolage introduces a novel structured-prediction technique that learns to create coherent mappings between pages by training on human-generated exemplars. The produced mappings can then be 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.

1 INTRODUCTION

Designers in many fields rely on examples for inspiration [Herring et al. 2009], and examples can facilitate better design work [Lee et al. 2010]. Examples can illustrate the space of possible solutions, and also how to implement those possibilities [Brandt et al. 2009; Buxton 2007]. Furthermore, repurposing successful elements from prior ideas can be more efficient than reinventing them from scratch [Gentner et al. 2001; Kolodner and Wills 1993; Hartmann et al. 2007].

The Web today provides a corpus of design examples unparalleled in human history. Unfortunately, this powerful resource is underutilized. While current systems assist with browsing examples [Lee et al. 2010] and cloning individual design elements [Fitzgerald 2008], adapting the gestalt structure of Web designs remains a time-intensive, manual process.

Most design reuse today is accomplished via templates, which are specially created for this purpose [Gibson et al. 2005]. With templates' standardized page semantics, people can 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 automatically transferring design and content between Web pages. Bricolage matches visually and semantically similar elements in pages to create coherent mappings between them. These mappings can then be used to transfer the content from one page into the style and layout of the other, without any user intervention (Figure 1).

Bricolage learns how to transfer content between pages by training on a corpus of exemplar mappings. To generate this corpus, we created a Web-based crowdsourcing interface for collecting human-generated mappings. The collector was populated with 50 popular Web pages, and 39 participants with some Web design experience were recruited to specify correspondences between two to four pairs of pages each. After matching every fifth element, participants also answered a free-response question about their rationale. The resulting data was used to guide the development of the algorithm, train Bricolage's machine learning components, and verify the results.

*e-mail:{ranju,jtalton,saahmad,srk}@cs.stanford.edu
UNPUBLISHED TECHNICAL REPORT

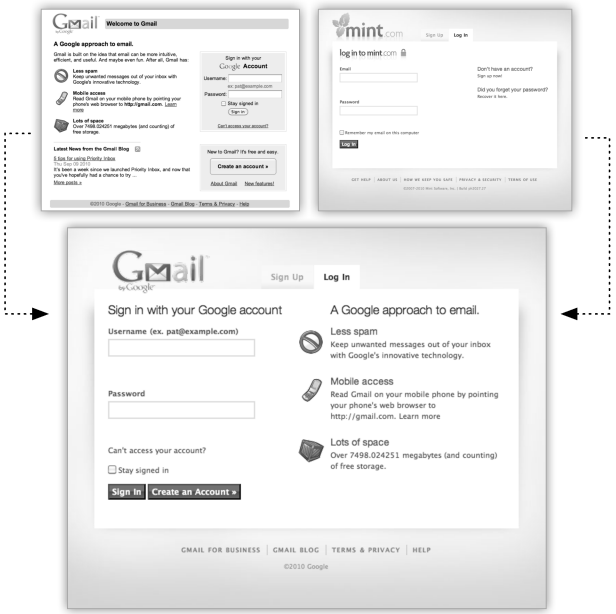


Figure 1: *Bricolage computes coherent mappings between Web pages by matching visually and semantically similar page elements. The produced mapping can then be used to guide the transfer of content from one page into the design and layout of the other.*

Since the mappings collected in our study are highly structured and hierarchical, Bricolage employs structured prediction techniques to make the mapping process tractable. Each page is segmented into a tree of contiguous regions, and mappings are predicted between pages by identifying elements in these trees. Bricolage introduces a novel tree matching algorithm that allows local and global constraints to be optimized simultaneously. The algorithm matches similar elements between pages while preserving important structural relationships across the trees.

This paper presents the 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 reproduce human mappings with a high degree of accuracy. Lastly, it gives examples of Bricolage being used for creating design alternatives, including rapid prototyping and retargeting content to alternate form factors such as mobile devices.

2 PAGE SEGMENTATION

Creating mappings between Web pages is facilitated by having some abstract representation of each page's structure that is amenable to matching. One candidate for this representation is the Document Object Model tree of the page, which provides a direct correspondence between each page region and the HTML that comprises it. However, the DOM may contain many nodes that have no visual effect on the rendered page, and lack other high-level structures that human viewers might expect.

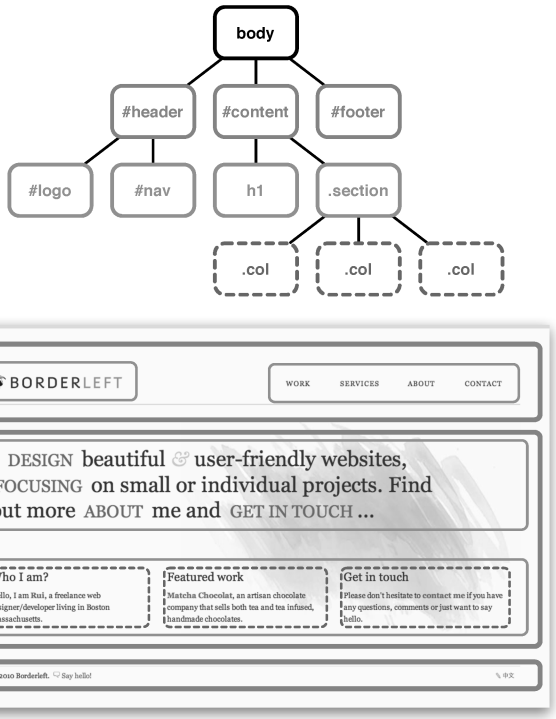


Figure 2: *A consistent page segmentation like the ones produced by our algorithm, and the associated DOM tree.*

Several page segmentation algorithms seek to partition the DOM in order to decompose Web pages into discrete sets of visually-coherent 2D regions [Cai et al. 2003; Chakrabarti et al. 2008; Kang et al. 2010], These algorithms produce good results as long as the page's DOM tree 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. Similarly, inlined text elements 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.

Bricolage introduces a novel page segmentation algorithm that "re-DOMs" the input page in order to produce clean and consistent segmentations (Figure 2). 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 labelled with its rendered page coordinates, and the algorithm checks whether each child's parent is the smallest region that 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 structure. This is accomplished by computing a set of VIPS-style separators across each page region [Cai et al. 2003], and inserting enclosing DOM nodes accordingly.

At the end of these four steps, all page content is assigned to some leaf node in the DOM tree, and every non-leaf node properly contains its children. In practice, this algorithm can produce standardized segmentations even for complex, design-oriented pages.

3 ANALYZING HUMAN MAPPINGS

The difficulty of developing automatic methods for generating mappings between Web pages has been noted [Fitzgerald 2008]. Rather than attempting to formulate an algorithm for page mapping *a priori*, we hypothesize that a more promising approach is to use machine learning techniques to train on a set of human-generated mappings.

To this end, we created the Bricolage Collector, a Web application for gathering human page mappings. We used the collector to solicit a corpus of mappings online, constructed between a variety of different Web pages, and analyzed these mappings to answer questions about how people map pages. To establish a baseline for the algorithm, we examined the consistency of the collected mappings and tested them for structural and hierarchical patterns. To facilitate the selection of discriminative features for the learning, we investigated the factors people consider when deciding which page elements to match. The corpus of generated mappings was then used to train and test the Bricolage algorithm.

3.1 Study Design

We selected a diverse corpus of 50 popular Web pages chosen from the Alexa Top 100, recent Webby nominees and award winners, highly regarded design blogs, and our own personal bookmark collections. The set omits pages which rely heavily on Flash, since they cannot be segmented effectively. To avoid overwhelming human mappers, it also omits pages which contain more than a hundred or so distinct elements.

From this corpus, we preselected a focus set of eight page pairs which seemed like good candidates for content transfer. 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. In this way, 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 for the study through email lists and online advertisements. Each reported some prior 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. Zoom in for detail.*

3.2 Procedure

Participants were first instructed to watch a tutorial video demonstrating the Bricolage Collector interface and describing the task (Figure 3). Users were asked to produce mappings to transfer the content from the left page into the layout of the right. The tutorial

emphasizes that participants can use any criteria they deem appropriate to match elements between pages. Upon completion of the tutorial, participants were redirected to the Collector Web application, and presented with the first pair.

The interface iterates over the segmented regions in the content page one at a time, and asks participants to find a matching region in the layout page. The user selects this region via the mouse or keyboard, and confirms it by clicking the "match" button at the top of the app. 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.

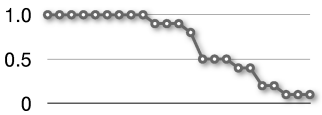
3.3 Results

In total, the 39 participants generated 117 mappings between 52 pairs of pages: 73 mappings for the 8 pairs in the focus set, and 44 covering the rest of the corpus. The collection also generated rationale explanations for 227 individual region assignments, averaging 4.7 words in length. Participants averaged 10.5 seconds finding a match for each page region ($\sigma^2 = 4.23s$, $\min = 4.42s$, $\max = 25.0s$), and about five minutes per pair of pages ($\mu = 5.38m$, $\sigma^2 = 3.03m$, $\min = 1.52m$, $\max = 20.7m$).

3.3.1 Consistency

Given two mappings for a particular pair of pages, we define the consistency to be the percentage of page regions which are given identical assignments. For the eight focus pairs, the average inter-mapping consistency was 78.3% ($\sigma^2 = 10.2\%$, $\min = 58.8\%$, $\max = 89.8\%$).

Moreover, 37.8% of page regions were mapped identically by all participants. For instance, inset is an ordered plot of the frequency of the region assignments used in mappings for the focus pair `http://ncadpostgraduate.com` and `http://31three.com`. Nearly half of these assignments were present in all of the human mappings.



3.3.2 Rationale

To gain intuition about *how* people map Web pages, we analyzed the rationale participants provided for the matches they made. One of the most popular and effective tools for mining textual data of this sort is Latent Semantic Analysis (LSA) [Deerwester et al. 1990], which provides an automatic mechanism for extracting contextual usage of language in a set of documents.

LSA takes a "bag of words" approach to textual analysis: each document is treated as an unordered collection of words without regard to grammar or punctuation. We followed the standard approach, treating each rationale as a document, forming the term-document matrix, and extracting is eigenvectors. We used Euclidean normalization to make annotations of different lengths comparable, and inverse document-frequency weighting to deemphasize common words like *a* and *the*. The principal components of the term-document matrix represent the semantic "dimensions" of the rationales, and words with the largest projections onto each component are its descriptors.

For the first component, the words with the largest projections include: *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 include: *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.

3.3.3 Structure and Hierarchy

Two statistics examine the structural and hierarchical properties of the 81 collected mappings: one measuring the degree to which mapped nodes preserve ancestry, and the other measuring the degree to which the mapping keeps groups of siblings together.

We define two matched regions to be *ancestry preserving* if their parent regions are also matched. The degree of ancestry preservation in a mapping is the number of ancestry preserving regions divided by the total number of matched regions. Participants mappings preserved ancestry 53.3% of the time ($\sigma^2 = 19.6\%$, $\min = 7.6\%$, $\max = 95.5\%$).

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. Participants produced mappings that were 83.9% sibling preserving ($\sigma^2 = 8.13\%$, $\min = 58.3\%$, $\max = 100\%$).

3.4 Analysis

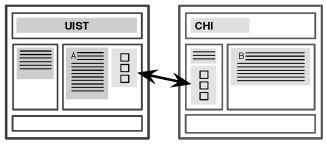
While users do not map pages in exactly the same way, the mappings produced by different people are highly consistent. Additionally, many assignments between pages were unanimous: there is a "method to the madness."

LSA found two factors to be highly predictive of human mappings: matching visual and semantic counterparts across pages, and preserving meaningful patterns and arrangements between elements.

This analysis helps clarify the kind of machine learning algorithm needed to produce good mappings between pages. It must incorporate semantic and structural constraints, and learn how to balance between them. It should produce mappings that are consistent with the ones in the training corpus, at least to the degree that those mappings are internally consistent themselves.

4 COMPUTING PAGE MAPPINGS

Since the abstract page representation produced by Bricolage's segmentation algorithm is a modified DOM tree, we turn to the tree matching literature which proposes several efficient methods for computing mappings between trees [Zhang and Shasha 1989; Shasha et al. 1994; Zhang 1996]. Unfortunately, by definition, these algorithms strictly preserve ancestry: once two nodes have been placed in correspondence, their descendants must be matched as well. Rigidly enforcing ancestry prevents semantic considerations from being balanced with structural ones. For instance, when two pages root their navigation elements differently, mapping the semantics of these regions is probably more important than preserving the overall page hierarchy (inset).



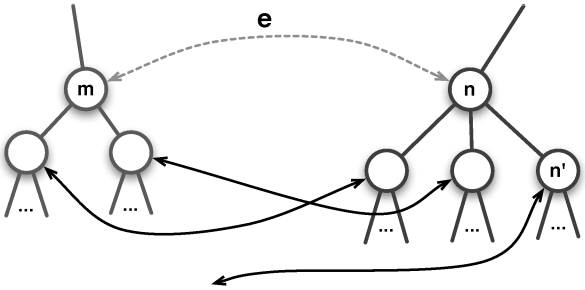


Figure 4: *To determine the ancestry penalty for an edge $e = [m, n]$, we count the children of m and n which induce ancestry violations. Here, only n' induces a cost on e.*

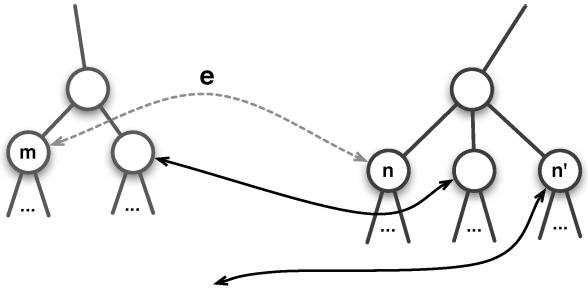


Figure 5: *To determine the sibling penalty for an edge $[m, n]$, we count the number of siblings of m that are not mapped to the siblings of n, and vice versa. In this example, only n' induces a cost on $[m, n]$.*

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 graph, and assigns a cost to each edge comprised of 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 perfect matching of the constructed graph. Bricolage uses structured prediction to learn a cost function under which the set of exemplar mappings are minimal [Collins 2002].

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. 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. The rest of this paper uses $M(m)$ to denote the image of a node m in the mapping regardless of the tree in which m resides.

To find the best mapping between pages, the algorithm assigns a cost $c(e)$ to each edge $e \in G$, 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 $M^\star = \text{argmin}_M\, c(M)$.

4.1 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 remaining edges in G, all of which involve no-match nodes, we fix the cost $c(e) = w_n$, where w_n is a constant no-match weight.

To compute $c_v([m, n])$, the algorithm compares visual and semantic properties of m and n by inspecting their DOM nodes. We describe the computation of this term in detail later in the paper.

The ancestry cost $c_a(\cdot)$ penalizes edges in the mapping which 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 to be the set

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

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

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

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

The sibling cost $c_s(\cdot)$ penalizes edges which fail to preserve sibling relationships between trees. To calculate this term, we must 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 $S(m) = \{C(P(m))\}$. Given a mapping M, the *sibling invariant subset* of m is the set

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

and the *sibling divergent subset* of m is the set

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

and the *distinct sibling groups* of m comprise the set $G(m) = \bigcup_{m' \in S(m)} P(M(m'))$. We define all corresponding terms for n symmetrically, and then compute

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

where w_s is a constant sibling violation weighting term. Intuitively, this term balances two important quantities: the degree to which the siblings of a node are "broken up" by the mapping, and the degree to which the node follows its siblings (see Figure 5).

4.2 Bounding Edge Costs

While this cost model provides an elegant way to balance semantic, ancestral, and sibling constraints, it cannot be used to search for the optimal mapping $M^\star$ directly. In particular, while $c_v([m, n])$ can be evaluated for an edge by inspecting m and n, $c_a(\cdot)$ and $c_s(\cdot)$ require information about the other edges are in the mapping. Therefore, to use this model to search for $M^\star$ directly, one would have to know the mapping *a priori*.

While we cannot evaluate $c_a(\cdot)$ and $c_s(\cdot)$ precisely, we can compute bounds for them on a per-edge basis [Chawathe and Garcia-Molina 1997]. Moreover, by computing tight upper and lower bounds for each edge, we can remove some edges from G that are guaranteed not to appear in $M^\star$. Each time we prune an edge in this way, the bounds for other nearby edges may be improved. Therefore, to find $M^\star$, we iteratively prune and bound edges until we have reduced the size of G as much as possible.

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.

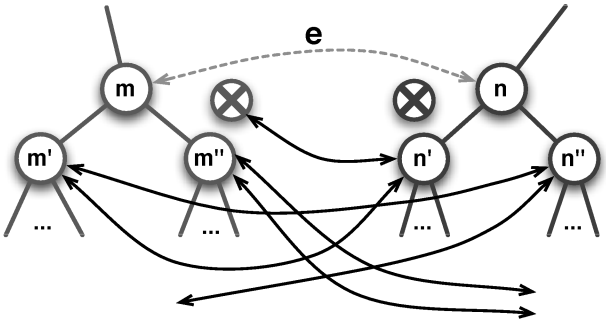


Figure 6: *To bound $c_a([m,n])$, observe that neither m' nor n' can induce an ancestry violation. Conversely, m'' is guaranteed to violate ancestry. No guarantee can be made for n''. Therefore, the lower bound for c_a is w_a, and the upper bound is $2w_a$.*

It is possible for a node $m' \in C(m)$ to induce an ancestry violation as long as there is some edge between it and a node in $T_2 \setminus (C(n) \cup \{\otimes_2\})$. Conversely, the node cannot be guaranteed to induce an ancestry violation as long as 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. Observe that pruning edges from G will cause the upper bound for $c_a([m,n])$ to decrease, and the lower bound to increase.

Bounds for $c_s([m,n])$ may be obtained in a similar way, by bounding each of the three terms $|D(\cdot)|$, $|I(\cdot)|$, and $|G(\cdot)|$. To bound $|D(m)|$, 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

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

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

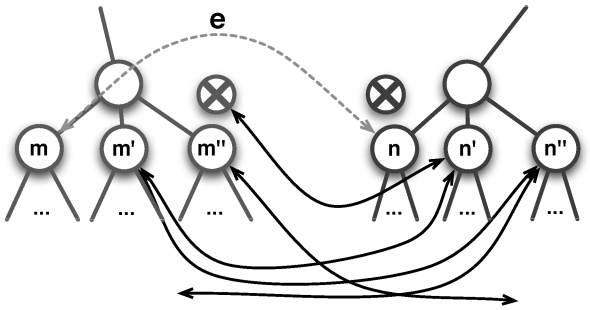


Figure 7: *To bound $c_s([m,n])$, observe that m' is guaranteed to be in $I(m)$, and m'' is guaranteed to be in $D(m)$. No guarantees can be made for n' and n''. Therefore, the lower bound for c_s is $w_s/4$, and the upper bound is $3w_s/4$.*

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

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

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

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 $|G(m)|$ 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 $c_{\mathcal{L}}(e) = c_v(e) + \mathcal{L}_a(e) + \mathcal{L}_s(e)$.

4.3 Pruning Edges Not In The Optimal Mapping

Under what conditions can an edge be safely removed from G? An edge can be pruned whenever it is certain that better choices exist for each of the edge's nodes. More formally, an edge $e = [m,n] \in T_1 \times T_2$ may be removed whenever there exist edges e_2 incident on m and e_3 incident on n such that

$$c_{\mathcal{L}}(e) \geq c^*_{\mathcal{U}}(e_2) + c^*_{\mathcal{U}}(e_3),$$

where

$$c^*_{\mathcal{U}}(e) = \begin{cases} c_{\mathcal{U}}(e) + w_a & \text{if } e \in T_1 \times T_2 \\ c_{\mathcal{U}}(e) & \text{else} \end{cases}.$$

Intuitively, if the worst possible combined cost of e_2 and e_3 is less than the best possible cost for e—even assuming that both e_2 and e_3 violate ancestry—e cannot appear in $M^\star$ and can be safely pruned. Similarly, edges of the form $e = [m, \otimes_2]$ may be pruned if there exists another edge e_2 incident on m such that $c_{\mathcal{L}}(e) \geq c_{\mathcal{U}}(e_2) + w_a$.

Once an edge $[m, n]$ has been pruned, bounds on all the edges incident on the parents and siblings of m and n can be updated. The only caveat during this process is that we must test each edge before pruning it to ensure that its removal will not leave G without a valid matching.

4.4 Approximating the Optimal Mapping

To find a low-cost mapping between pages, the bipartite graph G is constructed and the edge bounds initialized. Pruning begins, and iteratively removes edges from G until no more edges can be pruned. In general, this process will terminate well before a minimal perfect matching has been identified.

At this stage, a heuristic algorithm further filters the graph. Edges are iteratively fixed in order of increasing lower bound, and accepted into the mapping one at a time. After each edge is fixed, all other edges incident on its terminal nodes are removed, and another round of pruning is initiated.

After G has been decimated, we form a cost matrix for the assignment problem by averaging the upper and lower bounds for the remaining edges. This allows us to find a nearly optimal mapping between the two trees in polynomial time via the Hungarian algorithm.

5 LEARNING THE COST MODEL

Although this mapping algorithm can be used with any visual and semantic cost model and associated weights w_n, w_a, and w_v, the goal of Bricolage is to learn a model that will reproduce a corpus of human mappings. To do this, Bricolage 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.

5.1 Edge Features

The algorithm first computes a set of real-valued visual and semantic properties for each node in the page trees. Visual properties are based on a node's render-time appearance, and include attributes like width, font size, and mean RGB values. Semantic properties are inferred from the node's HTML, and take values in the Boolean domain $\{0, 1\}$ indicating conformance with some measurable semantic quality such as "is an image" or "is contained in the header." A full list of these properties is given in the Appendix.

To compute the total cost for an edge, the algorithm first calculates the relative 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.

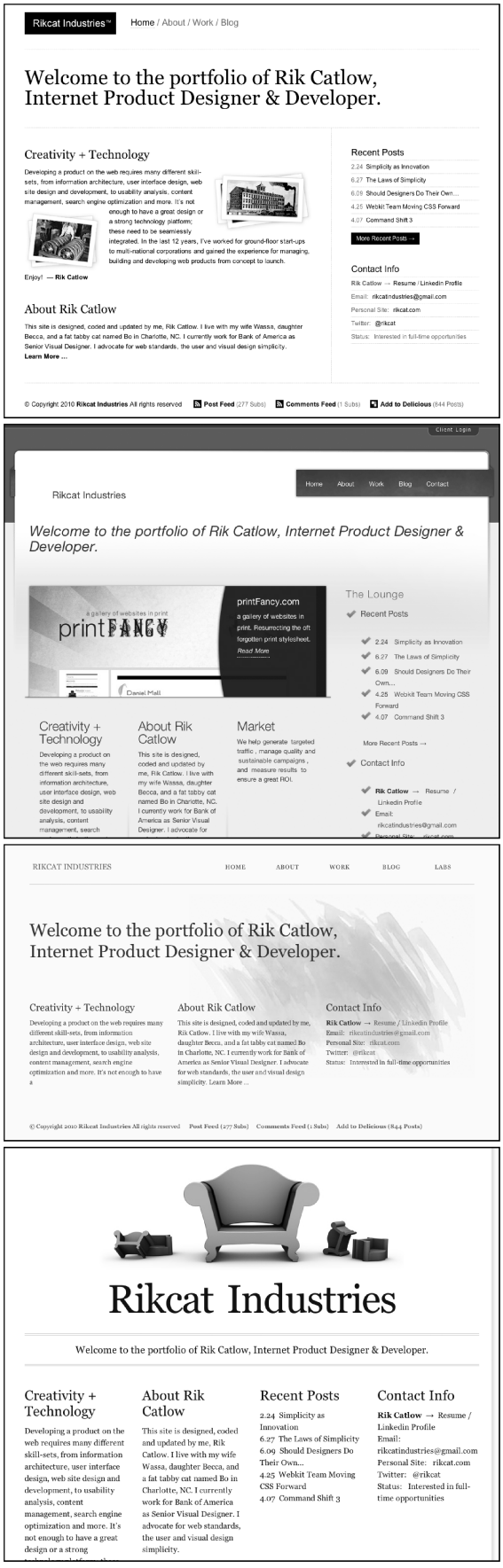


Figure 8: *Bricolage used to rapidly prototype many alternatives. (top) The original Web page. (bottom) The page automatically retargeted to three other layouts and styles.*

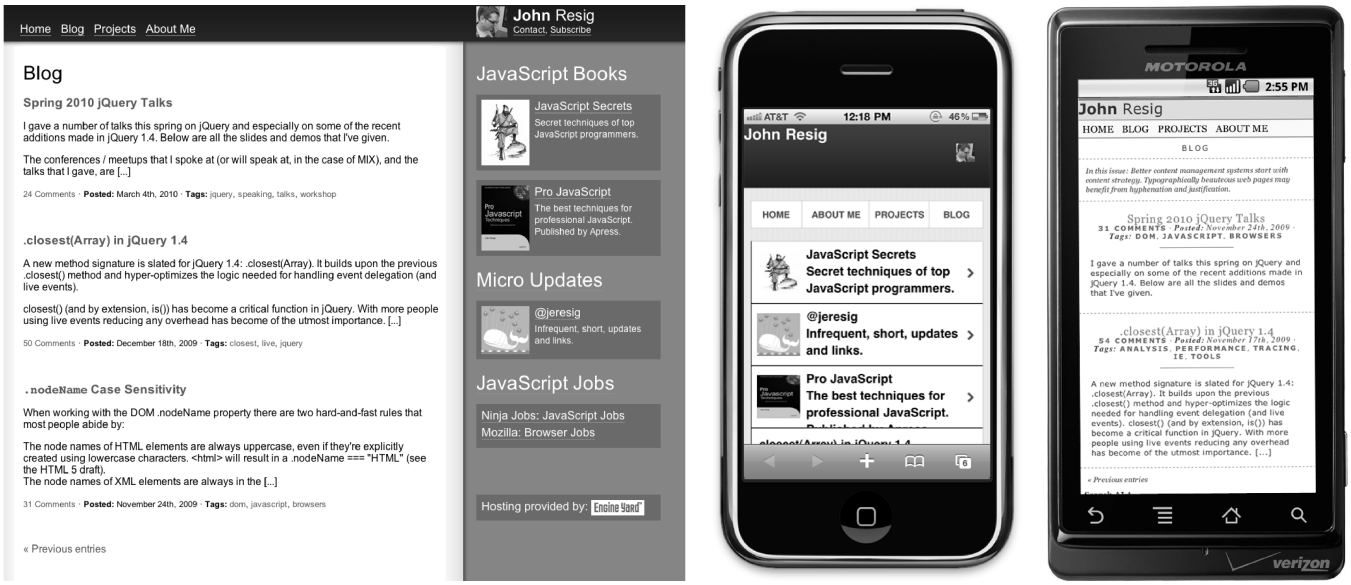


Figure 9: *Bricolage used for mobile retargeting. (left) The original Web page. (right) The page automatically retargeted to two different mobile layouts.*

5.2 Generalized Perceptron Algorithm

To learn a consistent assignment for $\bar{\mathbf{w}}$ under which the set of exemplar mappings are minimal, Bricolage uses the generalized perceptron algorithm for structured prediction [Collins 2002].

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

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

6 CONTENT TRANSFER

Once a mapping has been computed between pages, Bricolage uses it to guide the transfer of content from one page to the other. To do this, Bricolage searches the content Web page and identifies all matched nodes which do not have matched descendants. The children of these nodes are processed to inline their CSS properties and convert all contained URLs to absolute paths. Then, the nodes are cloned and inserted into the corresponding page in the locations indicated by the mapping, replacing any children already residing there. The transfer leaves unmatched regions in the layout page untouched, preserving its overall structure.

7 IMPLEMENTATION

Bricolage comprises several distinct components implemented using a wide variety of technologies. The 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 its browser engine.

To ensure that online Web updates do not interfere with the consistency of the results, all pages in the training corpus are archived using the Mozilla Archive File Format and uploaded to a centralized page 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 plugin library, allowing the set of features to be extended with minimal overhead, and mixed and matched at runtime.

The Bricolage Collector is written in CSS, Javascript, and HTML. Mapping results are sent to a centralized Ruby on Rails server as they are generated, and stored as XML in a database.

8 RESULTS

This section evaluates the efficacy of Bricolage in two ways. First, we show several practical examples of Bricolage in action. We also evaluate the machine learning components of the system by performing a cross-validation experiment on the gathered human mappings.

8.1 Examples

We show several practical examples of Bricolage being used for automatic retargeting. Figure 8 demonstrates the algorithm in a rapid prototyping scenario, in which an existing page is transformed into several potential replacement designs. This sort of parallel prototyping can significantly improve design performance [Dow et al. 2010]. Figure 9 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 10 illustrates an ancillary benefit of the cost model learned by Bricolage. 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 indiscernables), in practice it provides a useful mechanism for automatically differentiating between pages with similar and dissimilar designs.

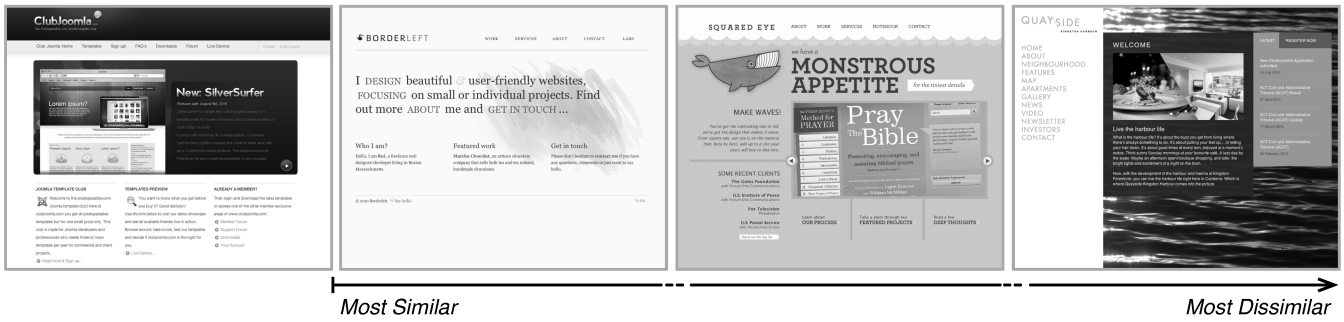


Figure 10: *Bricolage can be used to induce a distance metric on the space of Web designs. By mapping the leftmost page onto each of the pages in the corpus and examining the mapping cost, we can automatically differentiate between pages with similar and dissimilar designs.*

8.2 Machine Learning Results

To test the effectiveness of Bricolage's machine learning components, we trained Bricolage on the 44 collected human mappings not in the focus set. The perceptron was run for 400 iterations, and the weight vector averaged over the last 20. Then, the learned cost model was used to predict mappings for each of the 8 pairs in the focus set, and these mappings were compared to the reference mappings using three different metrics: average similarity, nearest neighbor similarity, and percentage of edges that appear in at least one mapping. Table 1 shows these results.

The mappings produced by Bricolage are not indistinguishable from those generated by humans. For a mapping algorithm to convincingly claim to have learned the space of human mappings, it would have to achieve an average similarity roughly equal to the 78% inter-mapping consistency of the focus set. Bricolage does about 15% worse.

However, Bricolage produces results that are often sufficient for it to masquerade as *a* human: the nearest neighbor similarity averages 73%. Moreover, almost all of the edges generated by Bricolage appear in some human mapping: with a larger sampling of human behaviors, it is likely that the 83% edge frequency would further increase.

A major motivation for the structured-prediction techniques at the heart of Bricolage was the hypothesis that ancestry and sibling relationships are crucial to predicting human mappings. To test this hypothesis, we also trained a cost model for Bricolage based purely on the visual and semantic metric. The resulting statistics validate our hypothesis: predicting mappings based purely on visual and semantic differences between page elements results in roughly a 30% decrease in performance.

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.

Cost Model	*Metric*	*%*
c_v, c_a, and c_s	Average Similarity	61.8
	Nearest Neighbor	73.0
	Edge Frequency	82.6
c_v alone	Average Similarity	44.6
	Nearest Neighbor	53.2
	Edge Frequency	64.4

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

9 CONCLUSIONS AND FUTURE WORK

This paper introduced the Bricolage algorithm for automatically transferring design and content between Web pages. It demonstrated that Bricolage can learn to closely reproduce human mappings, and presented examples of Bricolage being used to automatically retarget real-world Web pages. This work takes a first step towards a powerful new paradigm for example-based Web design, and opens up exciting areas for future research.

At present, the algorithm employs only about thirty simple visual and semantic features. Expanding this set to include more complex and sophisticated properties—such as those based on computer vision—will likely improve the robustness of the machine learning.

Additionally, the Bricolage prototype's content transfer implementation cannot handle many of the idiosyncrasies of modern HTML. Before the system can see widespread use, this situation must be remedied.

Because Bricolage can only manipulate Web page content that is part of the DOM, it cannot retarget pages that are authored entirely in Flash or Java. Extending Bricolage to handle these technologies remains future work; optimization approaches like Supple [Gajos and Weld 2004] may prove valuable.

Perhaps most exciting is the potential for creating an integrated design-based search and retargeting interface using Bricolage technology. Properly executed, such a system could have a profound effect on the Web design status quo.

References

BRANDT, J., GUO, P. J., LEWENSTEIN, J., DONTCHEVA, M., AND KLEMMER, S. R. 2009. Opportunistic programming: Writing code to prototype, ideate, and discover. *IEEE Software 26*, 5, 18–24.

BUXTON, B. 2007. *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann.

CAI, D., YU, S., WEN, J.-R., AND MA, W.-Y. 2003. VIPS: a vision-based page segmentation algorithm. Tech. Rep. MSR-TR-2003-79, Microsoft.

CHAKRABARTI, D., KUMAR, R., AND PUNERA, K. 2008. A graph-theoretic approach to Webpage segmentation. In *Proc. WWW*, 377–386.

CHAWATHE, S. S., AND GARCIA-MOLINA, H. 1997. Meaningful change detection in structured data. In *Proc. SIGMOD*, ACM, 26–37.

COLLINS, M. 2002. Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In *Proc. EMNLP*, ACL.

DEERWESTER, S., DUMAIS, S. T., FURNAS, G. W., LANDAUER, T. K., AND HARSHMAN, R. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science 41*, 6, 391–407.

DOW, S. P., GLASSCO, A., KASS, J., SCHWARZ, M., AND KLEMMER, S. R. 2010. The effect of parallel prototyping on design performance, learning, and self-efficacy. Tech. Rep. CSTR-2009-02, Stanford University.

FITZGERALD, M. 2008. CopyStyler: Web design by example. Tech. rep., MIT, May.

GAJOS, K., AND WELD, D. S. 2004. SUPPLE: automatically generating user interfaces. In *Proc. IUI*, ACM, 93–100.

GENTNER, D., HOLYOAK, K., AND KOKINOV, B. 2001. *The Analogical Mind: Perspectives From Cognitive Science*. MIT Press.

GIBSON, D., PUNERA, K., AND TOMKINS, A. 2005. The volume and evolution of Web page templates. In *Proc. WWW*, ACM, 830–839.

HARTMANN, B., WU, L., COLLINS, K., AND KLEMMER, S. R. 2007. Programming by a sample: rapidly creating Web applications with d.mix. In *Proc. UIST*, ACM, 241–250.

HERRING, S. R., CHANG, C.-C., KRANTZLER, J., AND BAILEY, B. P. 2009. Getting inspired!: understanding how and why examples are used in creative design practice. In *Proc. CHI*, ACM, 87–96.

KANG, J., YANG, J., AND CHOI, J. 2010. Repetition-based Web page segmentation by detecting tag patterns for small-screen devices. *IEEE Transactions on Consumer Electronics 56* (May), 980–986.

KOLODNER, J. L., AND WILLS, L. M. 1993. Case-based creative design. In *In AAAI Spring Symposium on AI and Creativity*, 50–57.

LEE, B., SRIVASTAVA, S., KUMAR, R., BRAFMAN, R., AND KLEMMER, S. R. 2010. Designing with interactive example galleries. In *Proc. CHI*, ACM.

SHASHA, D., WANG, J. T.-L., ZHANG, K., AND SHIH, F. Y. 1994. Exact and approximate algorithms for unordered tree matching. *IEEE Transactions on Systems, Man, and Cybernetics 24*, 4, 668–678.

ZHANG, K., AND SHASHA, D. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput. 18*, 6, 1245–1262.

ZHANG, K. 1996. A constrained edit distance between unordered labeled trees. *Algorithmica 15*, 3, 205–222.

APPENDIX: Features

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).