Flexible Tree Matching

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

Abstract

Tree-matching problems arise in many computational domains. The literature provides several methods for creating correspondences between labeled trees; however, by definition, tree-matching algorithms rigidly preserve ancestry. That is, once two nodes have been placed in correspondence, their descendants must be matched as well. We introduce *flexible* tree matching, which relaxes this rigid requirement in favor of a tunable formulation in which the role of hierarchy can be controlled. We show that flexible tree matching is strongly $\mathcal{NP}$-complete, give a stochastic approximation algorithm for the problem, and demonstrate how structured prediction techniques can learn the algorithm's parameters from a set of example matchings. Finally, we present results from applying the method to tasks in Web design.

1 Introduction

The problem of comparing trees arises naturally in diverse fields, including computational biology, compiler optimization, natural language processing, and computer vision [Bille, 2005]. The most common measure for gauging the similarity of two labeled trees is the edit distance metric, first introduced by Tai [1979], which computes the cost of transforming one tree into another through a sequence of elementary node operations such as insertion, deletion, and renaming. To calculate this distance, a minimum-cost correspondence is established between the nodes of the two trees in a process known as *tree matching*.

In the classical formulation, these correspondences are *rigid*: they are not allowed to violate ancestor-descendant relationships between nodes, nor the left-to-right order of a node's children. As a result of these structural requirements, the problem admits an efficient dynamic programming algorithm, and the optimal matching can be found in cubic time [Demaine *et al.*, 2009]. If the ordering requirement is removed (but the ancestry requirement maintained), the edit distance computation becomes $\mathcal{NP}$-complete [Zhang *et al.*, 1992], and approximation algorithms are necessary [Shasha *et al.*, 1994].

In some domains, the most appropriate matchings may not strictly preserve ancestry. For instance, while reparenting even a single node in a phylogenetic tree of bacteria would destroy its validity, the ancestry relationships in the Document Object Model tree of a Web page are much less prescriptive: moving a search bar from header to footer results in a different—but largely equivalent—page. This pattern follows for many other tree structures encountered in design and data management, in which hierarchy plays an important—but not definitive—role [Chawathe and Garcia-Molina, 1997].

We introduce *flexible* tree matching, which relaxes the requirement that the produced correspondence strictly preserve ancestry relationships. Instead, the algorithm provides a parameterized framework for controlling the relative import of labeling and hierarchy. Introduced in Kumar *et al.* [2011], the algorithm uses an edge-based cost model to match nodes with similar labels while simultaneously penalizing matchings which induce violations of the hierarchy or break up sibling groups. Determining the minimum edit distance between trees then reduces to finding a minimum-cost matching under this model.

We prove that flexible tree matching is $\mathcal{NP}$-complete in the strong sense, and give a corresponding stochastic approximation algorithm. We also show how to learn the parameters of the model from a corpus of examples via standard structured prediction techniques, and summarize results from applying the method to automatic retargeting in Web design.

2 A Flexible Model for Tree Matching

A *tree matching* is an injective binary relation defined between two labeled trees T_1 and T_2. The relation can be viewed as a bipartite graph between the trees' nodes, with edges representing editing operations for transforming one tree into the other. Edges identifying nodes with dissimilar labels represent *relabeling* operations, while nodes which are not mapped correspond to *insertions* or *deletions*.

To differentiate between mappings, classical tree matching requires a model that defines the relabeling cost between nodes and the insertion/deletion cost for nodes which are not matched. Given such a model, the tree-matching problem is to find a lowest-cost mapping between trees which preserves ancestry: once two nodes $m \in T_1$ and $n \in T_2$ are matched, the descendants of m can only be matched to descendants of n, and vice versa.

Flexible matching relaxes this rigid ancestry requirement in favor of a tunable edge-based cost model. It forms a complete bipartite graph G between $T_1 \cup \{\otimes_1\}$ and $T_2 \cup \{\otimes_2\}$, where $\otimes_1$ and $\otimes_2$ are auxiliary *no-match* nodes. Each edge in G is assigned a cost comprising three terms: a relabeling term c_r, penalizing edges that connect nodes with different labels, an ancestry term c_a, penalizing edges that violate ancestry relationships, and a sibling term c_s, penalizing edges that break up sibling groups. The cost of an edge $c(e)$ is the sum of these three terms, and the goal of flexible tree matching is to produce a set of edges $M \subset G$ such that every node in $T_1 \cup T_2$ is covered by precisely one edge and the total mapping cost $c(M) = \frac{1}{|T_1|+|T_2|} \sum_{e \in M} c(e)$ is minimized.

3 Exact Edge Costs

We define the cost of an edge $e \in T_1 \times T_2$,

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

For 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 penalty weight.

The relabeling term $c_r([m,n])$ defines the cost of swapping the labels of nodes m and n. This function is domain-dependent, and user-specified. If the labels are identical, $c_r([m,n]) = 0$.

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

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

where w_a is a constant ancestry weight (see Figure 1).

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. The *sibling group* of a node m comprises the children of its parent: $S(m) = \{C(P(m))\}$. Given a mapping M, the *sibling-invariant subset* of m, $I_M(m)$, is the set of nodes in m's sibling group that map to nodes in $M(m)$'s sibling group, *i.e.,*

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

the *sibling-divergent subset* of m, $D_M(m)$, 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.,*

$$D_M(m) = \{m' \in S(m) \setminus I_M(m) \mid M(m') \neq \otimes_2\};$$

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

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

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

$$c_s([m,n]; M) = w_s \left(\frac{|D_M(m)|}{|I_M(m)||F_M(m)|} + \frac{|D_M(n)|}{|I_M(n)||F_M(n)|} \right),$$

where w_s is a constant sibling violation weight. The two ratios in the cost increase when siblings are broken up by the matching (*i.e.,* their images have different parents), and decrease when siblings groups are maintained (see Figure 2). The $F_M(\cdot)$ terms are included to guarantee that the total sibling penalty contributed by a tree is bounded by the number of nodes it contains.

4 Example Matchings

Figure 3 compares ordered, unordered, and flexible tree matching. In these examples, a simple relabeling function assigns a constant weight w_r to all edges between tree nodes with differing labels. In ordered and unordered matching, the rigid preservation of ancestry leaves many nodes unmatched. In flexible matching, more common structure is preserved between the trees. By varying the terms in the cost model, different mappings can be achieved.

5 Flexible Tree Matching is $\mathcal{NP}$-complete

We prove that flexible tree matching is strongly $\mathcal{NP}$-complete. We employ a simple version of the flexible cost

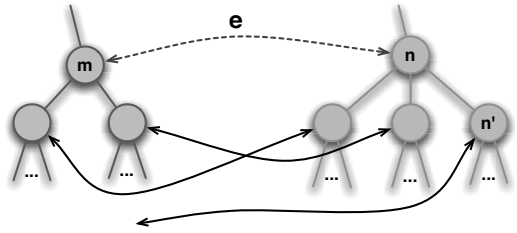

Figure 1: Flexible tree matching determines the ancestry penalty for an edge $e = [m,n]$ by counting the children of m and n which induce ancestry violations. In this example, n' is an ancestry-violating child of n because it is not mapped to a child of m; therefore, n' induces an ancestry cost on e.

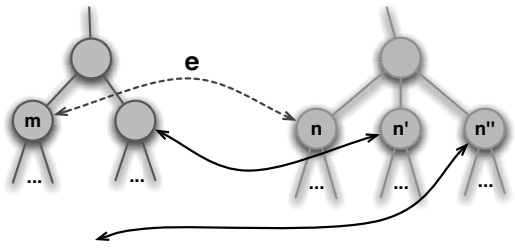

Figure 2: To determine the sibling penalty for an edge $e = [m,n]$, the algorithm computes the sibling-invariant and sibling-divergent subsets of m and n. In this example, $I_M(n) = \{n'\}$ and $D_M(n) = \{n''\}$; therefore, n' decreases the sibling cost on e and n'' increases it.

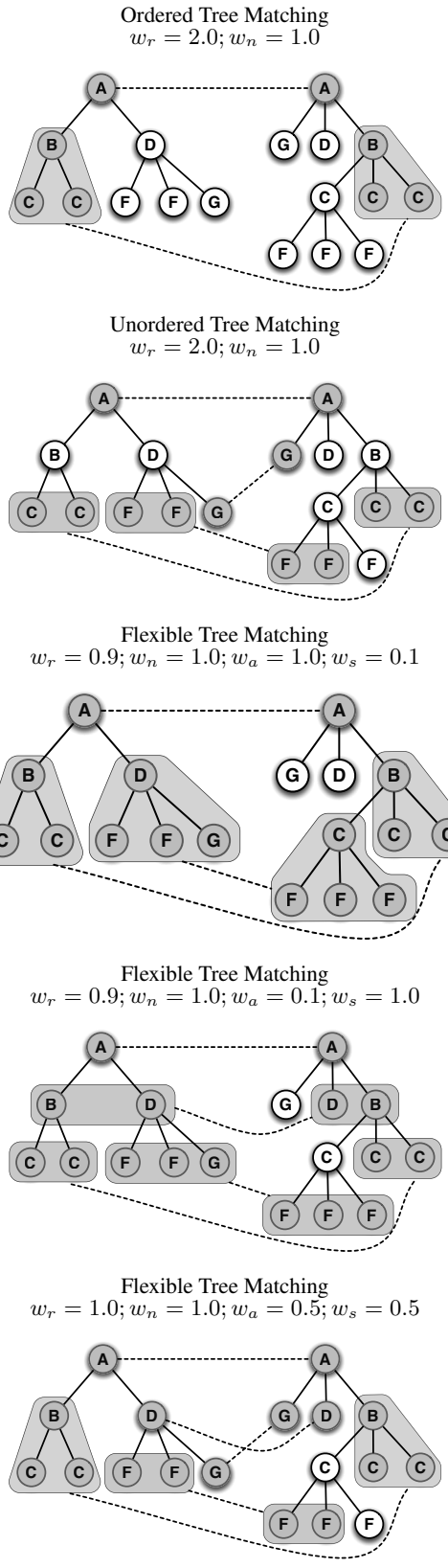

Figure 3: Examples of ordered, unordered, and flexible tree matchings.

model from Section 4, where $w_r = 1$, $w_n = 1$, $w_a = 0$, and $w_s = 1$. That is, ancestry violations are forgiven but all relabelings, no-matches, and sibling violations are costly. Given this model and two labeled trees T_1 and T_2, we address the decision problem "Does there exist a zero-cost flexible mapping between the trees?"

This problem is in $\mathcal{NP}$, since a proposed zero-cost mapping can be verified in polynomial time. To show that the problem is $\mathcal{NP}$-hard, we formulate a polynomial-time reduction from 3-PARTITION [Garey and Johnson, 1975].

The 3-PARTITION problem is to decide whether a given multiset of integers can be partitioned into triples that all have the same sum. More formally, each instance is a finite set S of $3m$ integers, where each integer $x_i \in S$ satisfies $K/4 < x_i < K/2$ for some bound $K \in \mathbb{Z}^+$ and $\sum x_i = 3K$. The question is to determine whether or not S can be partitioned into m disjoint sets $U_1, \ldots, U_m$, each with three elements, so that $\sum_{u \in U_i} u = K$ for every $i \in \{1, 2, \ldots, m\}$. This problem is $\mathcal{NP}$-complete even when K is polynomial in m [Garey and Johnson, 1975].

Given a 3-PARTITION instance, we construct trees T_1 and T_2 as in Figure 4. Each tree consists of three levels, and the nodes are labeled based on their level. T_1 represents the set S, and contains $3m$ subtrees on the second level, with subtree i possessing x_i leaf nodes. T_2 contains m subtrees on the second level with K children each, and $2m$ more one-node subtrees. Since K is polynomial in m, this construction takes time polynomial in the size of the 3-PARTITION instance.

Under what circumstances can a zero-cost matching exist between these two trees? Since $w_n = 1$, such a matching must map every node in T_1 to some node in T_2, and vice versa. Similarly, since $w_r = 1$, the matching cannot identify nodes in different levels of the trees. Most importantly, since $w_s = 1$, a zero-cost matching must preserve the $3m$ sibling groups of leaf nodes in T_1: each of the m leaf node sibling groups in T_2 must be matched to exactly three leaf node sibling groups in T_1. Thus, a zero-cost matching exists only if the corresponding 3-PARTITION instance has a solution. Conversely, a solution to the 3-PARTITION instance indicates which leaf node sibling groups in T_1 should be matched to a common leaf node sibling group in T_2, naturally inducing a zero-cost matching.

This reduction implies that no polynomial-time (or even pseudopolynomial-time) algorithm can exist for flexible tree matching. For this reason, we propose a stochastic optimization algorithm for approximating the optimal matching, based on bounding the edge costs.

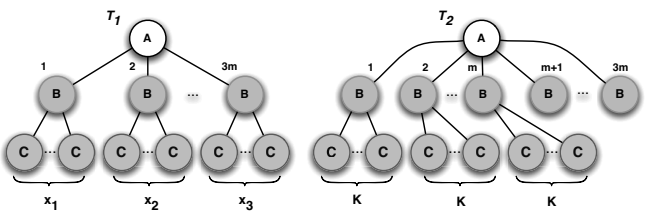

Figure 4: The tree construction for the reduction from 3-PARTITION.

6 Bounding Edge Costs

While the cost model described in Section 3 balances labeling and structural constraints, it cannot be used to search for an optimal mapping $M^\star$ directly. Although $c_r([m,n])$ 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. 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 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 $c_a([m,n];M)$ 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 5 illustrates the computation of these bounds. Pruning edges from G causes the upper bound for $c_a([m,n];M)$ to decrease, and the lower bound to increase.

Similarly, we can bound $c_s([m,n];M)$ 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 $D_M(m)$ 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 $D_M(m)$ 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 $|I_M(m)|$ 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 $|F_M(m)|$ 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 $c_s([m,n];M)$ 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

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

Figure 6 illustrates the computations of the bounds for the sibling cost term.

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

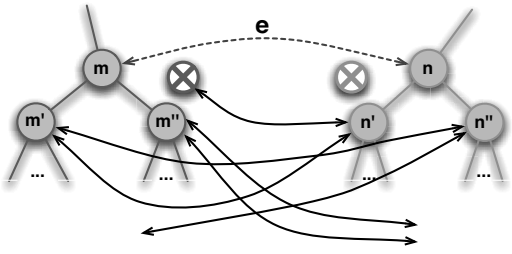

Figure 5: To bound $c_a([m,n];M)$, observe that neither m' nor n' can induce an ancestry violation. Conversely, m'' is guaranteed to violate ancestry. No guarantee can be made for n''. Therefore, the lower bound for c_a is w_a, and the upper bound is $2w_a$.

7 Approximating the Optimal Mapping

To approximate the optimal mapping M^*, we use the Metropolis algorithm [Press *et al.*, 2007]. 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.

8 Learning Cost Models

While flexible tree matching can be used with any cost model comprising weights w_r, w_a, w_s, and w_n, it is often desirable to *learn* a model that will produce mappings with domain-dependent characteristics. In particular, given a set of trees and exemplar matchings defined between them, we can use the generalized perceptron algorithm to learn weights under which the example matchings are minimal [Collins, 2002].

First, we reformulate the cost of a mapping $c(M)$ in terms of a weight vector $\mathbf{w} = \langle w_r, w_a, w_s, w_n \rangle$. For each edge, we compute the difference between the real-valued labels of its terminal nodes and the exact ancestry and sibling costs, and concatenate these values along with a Boolean no-match indicator into a feature vector $\mathbf{f}_e$. The edge cost can then be computed as $c(e) = \mathbf{w}^T \mathbf{f}_e$. Given a mapping M, the algorithm assembles an aggregate feature vector $\mathbf{F}_M = \frac{1}{|T_1|+|T_2|} \sum_{e \in M} \mathbf{f}_e$ to calculate the mapping cost $c(M) = \mathbf{w}^T \mathbf{F}_M$.

In each training iteration, the perceptron randomly selects a pair of trees and an associated mapping M from the training set. Next, it computes a new mapping $\hat{M} \approx \operatorname{argmin}_M \mathbf{w}_i^T \mathbf{F}_M$ using the current cost model $\mathbf{w}_i$. For the first iteration, $\mathbf{w}_0 = 0$. Based on the resultant mapping, a new aggregate feature vector $\mathbf{F}_{\hat{M}}$ is calculated, and the cost model is updated by $\mathbf{w}_{i+1} = \mathbf{w}_i + \alpha_i \left(\mathbf{F}_{\hat{M}} - \mathbf{F}_M\right)$, where $\alpha_i = 1/\sqrt{i+1}$ is the learning rate.

While the 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 [Collins, 2002]. 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.

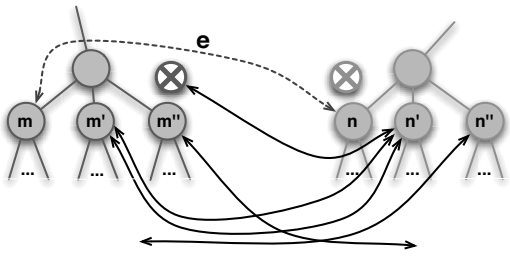

Figure 6: To bound $c_s([m,n]; M)$, observe that m' is guaranteed to be in $I_M(m)$, and m'' is guaranteed to be in $D_M(m)$. No guarantees can be made for n' and n''. Therefore, the lower bound for c_s is $w_s/4$, and the upper bound is $3w_s/4$.

9 Case Study

Flexible tree matching was inspired by the goal of automatically retargeting Web pages [Kumar *et al.*, 2011]. In this setting, the matching identifies semantically and structurally similar elements between pages to guide the transfer of content and design. We selected a corpus of 50 popular Web pages, and segmented the Document Object Model tree of each one to produce a hierarchy representative of its visual structure. In an online study, participants specified 117 matchings between 52 unique pairs of page trees. Then, using the method described in Section 8, we trained a cost model for flexible matching on this set.

In this domain, learning a consistent model for tree matching is complicated by the fact that humans do not produce identical mappings between pages. In our study, humans matchings are only 78.3% consistent, on average. Nonetheless, the flexible matching model is effective in this context, yielding matchings that average 68.7% similarity (and 77.7% under the nearest-neighbor metric) in a hold-out test.

Figure 7: Flexible tree matching can be used to 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.

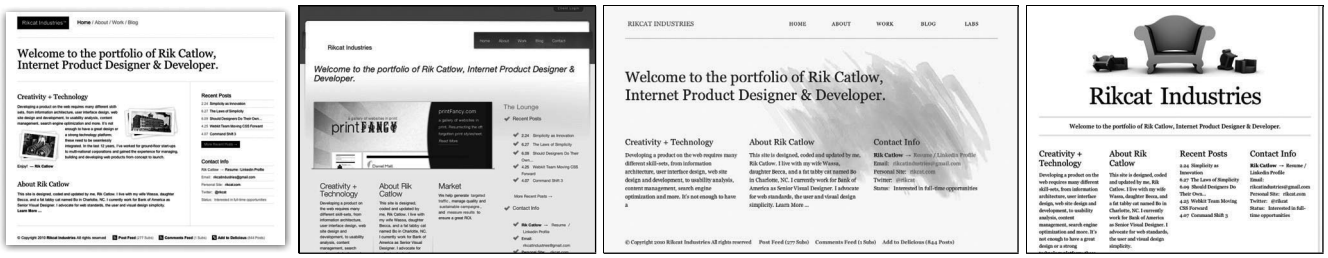

Figure 8: Flexible tree matching used to rapidly prototype many alternatives. *Left:* the original Web page. *Rest:* the page automatically retargeted to three other layouts and styles.

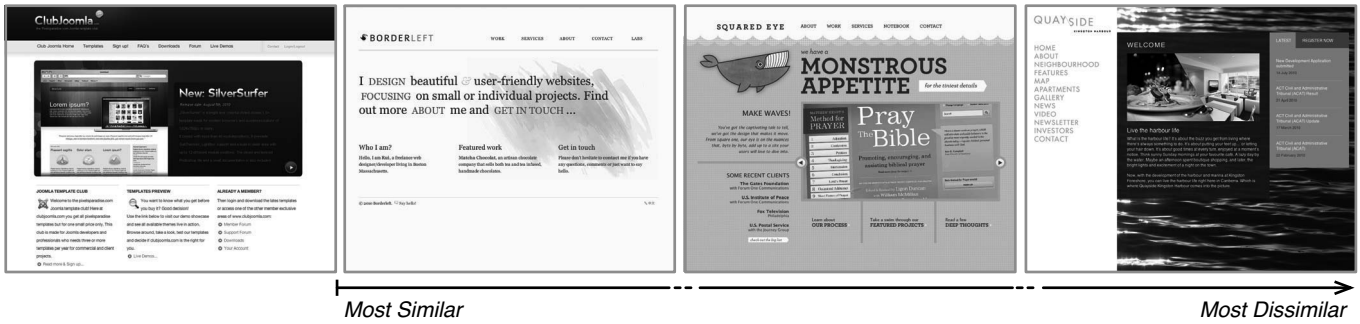

Figure 9: The flexible tree matching algorithm can be used to induce a distance metric on the space of Web designs. Comparing the cost of matching the source page onto each of the targets can differentiate similar and dissimilar designs.

Once the cost model has been learned, the matchings it produces can be used to guide the transfer of content and design between pages (Figure 7-8). In addition, the algorithm provides an approximate distance metric on the space of page designs (Figure 9). See Kumar *et al.* [2011] for details.

10 Conclusion

This paper describes a tunable algorithm for flexible tree matching, with parameters that can be learned via structured prediction techniques. This algorithm may be useful for matching in many domains in which hierarchy is suggestive rather than definitive. We have demonstrated its efficacy on mapping between Web designs; future work could apply the technique to other design domains with hierarchical structure, such as 3D models or graphic design.

References

[Bille, 2005] Philip Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337:217–239, 2005.

[Chawathe and Garcia-Molina, 1997] Sudarshan Chawathe and Hector Garcia-Molina. Meaningful change detection in structured data. In *Proc. SIGMOD*, pages 26–37. ACM, 1997.

[Collins, 2002] Michael Collins. Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In *Proc. EMNLP*. ACL, 2002.

[Demaine *et al.*, 2009] Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An optimal decomposition algorithm for tree edit distance. *Transactions on Algorithms*, 6(1):2:1–2:19, December 2009.

[Garey and Johnson, 1975] Michael R. Garey and David S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975.

[Kumar *et al.*, 2011] Ranjitha Kumar, Jerry O. Talton, Salman Ahmad, and Scott R Klemmer. Bricolage: Example-based retargeting for web design. In *Proc. CHI*. ACM, 2011.

[Press *et al.*, 2007] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3rd edition, 2007.

[Shasha *et al.*, 1994] Dennis Shasha, Jason Tsong-Li Wang, Kaizhong Zhang, and Frank Y. Shih. Exact and approximate algorithms for unordered tree matching. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(4):668–678, 1994.

[Tai, 1979] Kuo-Chung Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3):422–433, July 1979.

[Zhang *et al.*, 1992] Kaizhong Zhang, Rick Statman, and Dennis Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, 42(3):133–139, May 1992.