1 Flexible Tree Matching

One problem of the standard tree edit distance is the strict requirement regarding the tree hierarchy and ordering among siblings. In some domains, the most appropriate matching may not follow these requirements. One of these domains is the DOM (Document Object Model) of a website. Take a look at figure 1.1. There you see an example website. Furthermore figure 1.2 contains two websites, where only small changes were made. In figure 1.2a the button was moved from the bottom of the page to the top, in figure 1.2b the order of the social media icons were jumbled. These changes are minor in both cases and should not affect the distance measurement substantially. Nevertheless, depending on the costs of the edit operations, it may be very expensive to perform the necessary edit operations. High costs would lead to a large edit distance, although the websites are eminently similar.

This issue is not a rare one among the problem of comparing trees. That

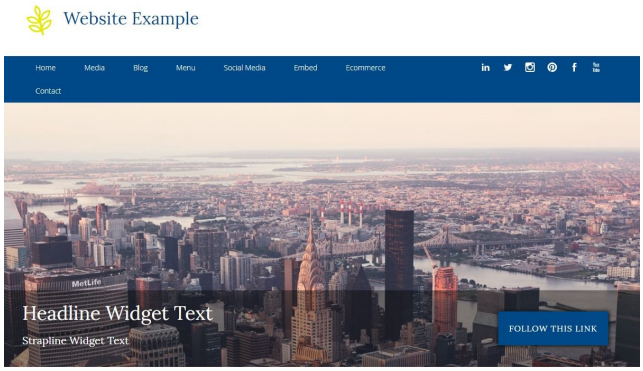
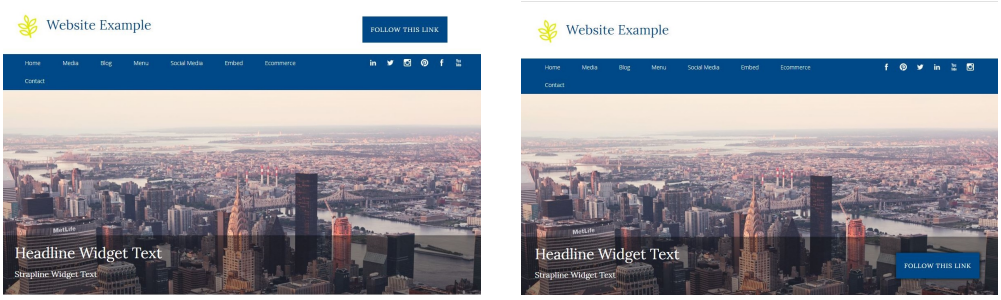


Figure 1.1: The example website.



(a) Only the button has a different location.

(b) The ordering of the social media links where changed.

Figure 1.2: Two example of similar DOMs like the example website

lead to the introduction of the flexible tree matching. Instead of requiring strict left-to-right ordering and hierarchy conditions, one may relax them by introducing costs for all occurrences, where these conditions are not fulfilled. Kumar et al. [] developed an algorithm that matches nodes with similar labels and penalizes edges that break up sibling groups or violate the hierachy. Finding the minimum flexible tree edit distance can be reduced to finding a minimum cost matching. Therefore one can show, that finding the flexible tree edit distance is strongly $\mathcal{NP}$-complete. This implies, that there are no efficient algorithms to compute the minimum flexible tree edit distance. However, there are some approximating heuristics.

1.1 The Model for the Flexible Tree Edit Distance

Let T_1 and T_2 be the two trees for which the flexible tree edit distances shall be computed. We construct a complete bipartite graph G with $G := (\{V(T_1) \cup \otimes_1\} \times \{V(T_2) \cup \otimes_2\}, E)$. Hence the edge set E is defined as the set $E := \{\{v_1, v_2\} \mid v_i \in \{V(T_i) \cup \otimes_i\} \text{ for } i = 1, 2\}$. The nodes $\otimes_i$ are so called *no-match* nodes. Every edge $e = \{v_1, v_2\} \in E$ represents matching a node v_1 to a node v_2. If $v_2 = \otimes_2$, the edge e would represent the deletion of v_1, since v_1 was not matched to any node from the tree T_2. Analogously, the same holds for an edge $\{\otimes_1, v_2\}$. Every edge $e \in T_1 \times T_2$ is assigned

some cost function $c(e) = c_r(e) + c_a(e) + c_s(e)$. The exact definitions follow later. In short terms, these three summands represent the costs that we mentioned earlier in this chapter: c_r represents the costs of relabelling a node, c_a penalizes violations of ancestry relationships and c_s punishes broken up sibling groups. Edges connecting tree nodes with no-match nodes have a fixed constant cost w_n, only depending on the number of nodes in the trees. The goal is to find a minimum cost flexible matching $M \subset E$ such that every node of $T_1 \cup T_2$ is covered exactly once while the no-match nodes may be covered multiple times. Suppose that $v \in V(T_1)$ and $w \in V(T_2)$ and let $e := \{v, w\} \in E$. The costs for relabelling e, i.e. $c_r(e)$, only depend on the nodes v and w themselves. They are fixed for every edge and are known before starting to match nodes. $c_a(e)$ and $c_s(e)$ on the other hand may depend on the choice of the flexible matching M. To be more precise the costs $c_a(\{v, w\})$ are linearly dependent on the number of children of v that do not get mapped onto children of w. Define $M(v) \in \{T_2 \cup \otimes_2\}$ to be the node that v is mapped onto according to the flexible matching M and suppose that $M(v) \neq \otimes_2$. Moreover, define $C(v) \subset T_1$ to be the set of children of node v. Then we can define $V(v)$ to be the set of children of v that violate the ancestry condition:

$$V(v) := \{v' \in C(v) \mid M(v') \in T_2 \setminus C(M(v))\}$$

As stated previously, the cost function $c_a(e)$ is linearly dependent on the sizes of the sets $V(v)$ and $V(w)$ with some constant factor ω_a:

$$c_a(\{v, w\}) := \omega_a(|V(v)| + |V(w)|)$$

We need further tree related concepts before we can define the cost function $c_s(e)$ in a straight forward way. Therefore $P(v) \in T_1$ is defined as the parent of the node v and $S(v) := C(P(v))$ is the sibling group of v. If v is the root of the tree, then $P(v)$ does not exist and $S(v) := \{v\}$. Note that the sibling group $S(v)$ always contains the node v itself and thus is not empty. For a mapping M, we define the sibling-invariant subset of v, $I_M(v)$, to be the

siblings of v which are mapped into the same sibling group as v:

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

Accordingly the sibling-divergent subset of v, $D_M(v)$, are the siblings of v which are mapped to a node in $T_2 \setminus M(v)$:

$$\begin{aligned} D_M(v) :&= \{v' \in S(v) \mid M(v') \in T_2 \setminus S(M(v))\} \\ &= \{v' \in S(v) \setminus I_M(v) \mid M(v') \neq \otimes_2\} \end{aligned}$$

Finally, we define the set of distinct sibling families to be the set of all sibling groups, that the siblings of v map into:

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

Now we can define the costs for sibling group violations depending on a constant ω_s:

$$c_s(\{v, w\}, M) := \omega_s\left(\frac{|D_M(v)|}{|I_M(v)||F_M(v)|} + \frac{|D_M(w)|}{|I_M(w)||F_M(w)|}\right)$$

One can show, that the costs $c_s(\{v, w\}, M)$ increase, if a node in the sibling group of v or w gets reassigned to some node outside of the corresponding sibling group.

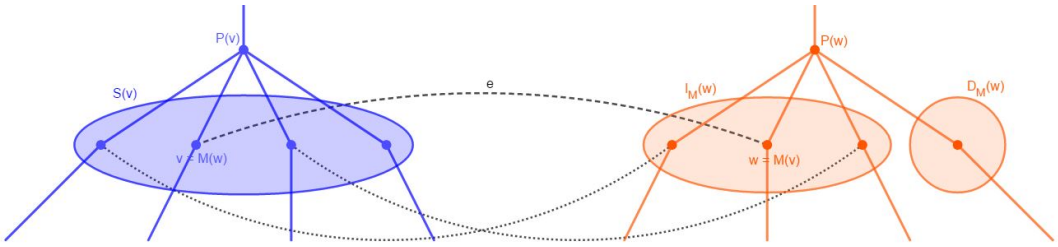


Figure 1.3: A visual representation of the above described tree concepts.

1.2 Approximation and Conclusion

As spoiled in the introduction of this section, computing the flexible tree edit distance is $\mathcal{NP}$-hard. There is a short and elegant proof, that presents a reduction of the 3-partition problem to the flexible tree matching problem []. Hence, one needs to rely on stochastic optimization algorithms to get an approximation of the flexible tree edit distance. Kumar et al. [] present a Monte Carlo algorithm where they fix edges one after another, prune all other incident edges to its endpoints and update the bounds for all other nodes. Naturally the more edges are fixed, the better the bounds get. For a more detailed description of the actual algorithm, please take a look at the cited paper. They also described how to adapt the cost factors ω_r, ω_s, ω_a and ω_n step by step to improve the results.
Depending on the application, the flexible tree matching can have huge advantages, especially in fields where hierarchy is suggestive rather than definitive. As mentioned in the introduction, you may have different significance on sibling group violations or re-parenting nodes. The cost model for the flexible tree matching allows you to adapt the weights to get the results you want. If you have a database of exemplar matchings you may even implement a learning cost model that improves the cost factors to follow your needs. Nevertheless, there can not be an efficient algorithm. So all results are more suggestive than definitive, just like the problem itself.