An A*-algorithm for the Unordered Tree Edit Distance with Custom Costs[⋆]

Benjamin Paaßen[0000−0002−3899−2450]

Institute of Informatics
Humboldt-University of Berlin
Rudower Chaussee 25, 12489 Berlin, Germany
benjamin.paassen@hu-berlin.de

Abstract. The unordered tree edit distance is a natural metric to compute distances between trees without intrinsic child order, such as representations of chemical molecules. While the unordered tree edit distance is MAX SNP-hard in principle, it is feasible for small cases, e.g. via an A* algorithm. Unfortunately, current heuristics for the A* algorithm assume unit costs for deletions, insertions, and replacements, which limits our ability to inject domain knowledge. In this paper, we present three novel heuristics for the A* algorithm that work with custom cost functions. In experiments on two chemical data sets, we show that custom costs make the A* computation faster and improve the error of a 5-nearest neighbor regressor, predicting chemical properties. We also show that, on these data, polynomial edit distances can achieve similar results as the unordered tree edit distance.

Keywords: Unordered Tree Edit Distance · A* algorithm · Tree Edit Distance · Chemistry

1 Introduction

Tree structures occur whenever data follows a hierarchy or a branching pattern, like in chemical molecules [6,10], in RNA secondary structures [11], or in computer programs [9]. To perform similarity search on such data, we require a measure of distance over trees. A popular choice is the tree edit distance, which is defined as the cost of the cheapest sequence of deletions, insertions, and relabelings that transforms one tree to another [3,14,16]. Unfortunately, the edit distance becomes MAX SNP-hard for unordered trees, like tree representations of chemical molecules [15]. Still, for smaller trees, we can compute the unordered tree edit distance (UTED) exactly using strategies like A* algorithms [7,13]. Roughly speaking, an A* algorithm starts with an empty edit sequence and then successively extends the edit distance such that a heuristic lower bound for the cost of the edit sequence remains as low as possible. The

⋆ Funding by the German Research Foundation (DFG) under grant number PA 3460/2-1 is gratefully acknowledged.

tighter our lower bound h, the more we can prune the search and the faster the A* algorithm becomes. Horesh et al. have proposed a heuristic based on the histogram of node degrees [7] and Yoshino et al. have improved upon this heuristic by also considering label histograms and by re-using intermediate values via dynamic programming [13]. However, both approaches assume unit costs, i.e. that deletions, insertions, and relabelings all have a cost of 1, irrespective of the content that gets deleted, inserted, or relabeled. This is unfortunate because, in many domains, we have prior knowledge that suggests different costs or we may wish to learn costs from data [9]. Accordingly, most tree edit distance algorithms are general enough to support custom deletion, insertion, and replacement costs, as long as they conform to metric constraints [3,14,16].

In this paper, we develop three novel heuristics for the A* algorithm which all support custom costs. The three heuristics have linear, quadratic, and cubic complexity, respectively, where the slower heuristics provide tighter lower bounds. Based on these novel heuristics, we investigate three research questions:

RQ1: Which of the three heuristics is the fastest? And how do they compare against the state-of-the-art by Yoshino et al. [13]?
RQ2: Do custom edit costs actually contribute to similarity search?
RQ3: How does UTED compare to polynomial edit distances in similarity search?

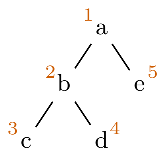
We investigate these research questions on two example data sets of chemical molecules, both represented as unordered trees. To answer RQ2 and RQ3, we consider a regression task where we try to predict the chemical properties of a molecule (boiling point and stability, respectively) via a nearest-neighbor regression. We begin our paper with more background and related work before we describe our proposed A* algorithm, present our experiments, and conclude.

2 Background and Related Work

Let Σ be an arbitrary set which we call *alphabet*. Then, we define a *tree* over Σ as an expression of the form $\hat{x} = x(\hat{y}_1, \ldots, \hat{y}_K)$, where $x \in \Sigma$ and where $\hat{y}_1, \ldots, \hat{y}_K$ is a list of trees over Σ, which we call the *children* of $\hat{x}$. If $K = 0$, we call $x()$ a *leaf*. We denote the set of all trees over Σ as $\mathcal{T}(\Sigma)$.

In this paper, we are concerned with similarity search on trees. In the literature, there are three general strategies to compute similarities on trees. First, we can construct a feature mapping $\phi : \mathcal{T}(\Sigma) \to \mathbb{R}^n$, which maps an input tree to a feature vector, and then compute a (dis-)similarity between features, e.g. via $d(\hat{x}, \hat{y}) = \|\phi(\hat{x}) - \phi(\hat{y})\|$. For example, we can represent trees by *pq*-grams [2], by counts of typical tree patterns [5], or by training a neural network [6,8]. The second strategy are tree kernels k, i.e. functions that directly compute inner products $k(\hat{x}, \hat{y}) = \phi(\hat{x})^T \cdot \phi(\hat{y})$ without the need to explicitly compute ϕ [1,5].

In this paper, we focus on a third option, namely tree edit distances [3]. Let Σ be an alphabet with $- \notin \Sigma$. Roughly speaking, a tree edit distance $d(\hat{x}, \hat{y})$ between two trees $\hat{x}$ and $\hat{y}$ from $\mathcal{T}(\Sigma)$ is the cost of the cheapest sequence



$M_1 = \{(1,1),(2,0),(3,4),(4,3),(5,2)\}$ $\quad d_1^{\mathrm{UTED}}(\hat{x},\hat{y}) = 1$

$M_2 = \{(1,1),(2,0),(3,4),(4,3),(5,0),(0,2)\}$ $\quad d_1^{\mathrm{CUTED}}(\hat{x},\hat{y}) = 3$

$M_3 = \{(1,1),(2,0),(3,2),(4,3),(5,4)\}$ $\quad d_1^{\mathrm{TED}}(\hat{x},\hat{y}) = 3$

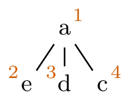


Fig. 1. An illustration of mappings according to the unordered tree edit distance [15] (top), the constrained unordered tree edit distance [14] (center), and the ordered tree edit distance [16] (bottom) between the same two trees. The distances assume unit costs. Numbers in superscript show the depth first order.

of deletions, insertions, and relabelings of nodes in $\hat{x}$ such that we obtain $\hat{y}$ [3,14,16]. More precisely, let $x_1, \ldots, x_m$ be the nodes of $\hat{x}$[1] and $y_1, \ldots, y_n$ be the nodes of $\hat{y}$ in depth-first-search order. Then, we define a *mapping* between $\hat{x}$ and $\hat{y}$ as a set of tuples $M \subset \{0, 1, \ldots, m\} \times \{0, 1, \ldots, n\}$ such that each $i \in \{1, \ldots, m\}$ occurs exactly once on the left and each $j \in \{1, \ldots, n\}$ occurs exactly once on the right. Figure 1 illustrates three example mappings between the trees $a(b(c, d), e)$ (left) and $a(e, d, c)$ (right), namely M_1, M_2, and M_3 (center left). Each mapping M can be translated into a sequence of edits by deleting all nodes x_i where $(i, 0) \in M$, by replacing nodes x_i with y_j where $(i, j) \in M$ and $x_i \neq y_j$, and by inserting all nodes y_j where $(0, j) \in M$. We denote the set of all mappings between $\hat{x}$ and $\hat{y}$ as $\mathcal{M}(\hat{x}, \hat{y})$. Next, we define a *cost function* as a metric $c : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \to \mathbb{R}$, and we define the cost of a mapping M as $c(M) = \sum_{(i,j) \in M} c(x_i, y_j)$ where $x_0 = y_0 = -$. A typical cost function is $c_1(x, y) = 1$ if $x \neq y$ and $c_1(x, y) = 0$ if $x = y$, which we call *unit costs*. Finally, we define the tree edit distance $d_c : \mathcal{T}(\Sigma) \times \mathcal{T}(\Sigma) \to \mathbb{R}$ according to cost function c as the minimum $d_c(\hat{x}, \hat{y}) = \min_{M \in \mathcal{M}(\hat{x}, \hat{y})} c(M)$.

We obtain different edit distances depending on the additional restrictions we apply on the set of mappings $\mathcal{M}(\hat{x}, \hat{y})$. The unordered tree edit distance (UTED) requires that mappings respect the ancestral ordering, i.e. if $(i, j) \in M$, then descendants of i can only be mapped to descendants of j [3]. A cheapest example mapping according to unit costs is M_1 (Figure 1, top). The constrained unordered tree edit distance (CUTED) [14] additionally requires that a deletion/insertion of a node implies either deleting/inserting all of its siblings or all of its children but one. This forbids M_1 and M_3, where b is deleted but both its sibling and more than one child are maintained. M_2 is a cheapest mapping according to CUTED with unit costs. The ordered tree edit distance (TED) [16] requires that the ancestral ordering and the depth-first ordering is maintained. Accordingly, neither M_1 nor M_2 are permitted because they swap the order of c and d. $M_3$is a cheapest mapping according to TED with unit costs. Note that UTED is MAX-SNP hard. However, CUTED and TED are both polynomial [14,16] via dynamic programming and we consider them as baselines in our experiments.

[1] Note that we use 'node' and 'label' interchangeably in this paper. To disambiguate between two nodes with the same label, we use the index.

3 Method

In this section, we explain our proposed A* algorithm for the unordered tree edit distance (UTED). We begin with the general scheme, which we adapt from Yoshino et al. [13], and then introduce three heuristics to plug into the A* algorithm.

Algorithm 1 The A* algorithm to compute the unordered tree edit distance $d_c^{\mathrm{UTED}}(\hat{x}, \hat{y})$ between two trees $\hat{x}$ and $\hat{y}$, depending on a cost function c and a heuristic h.

```
 1: function ASTAR_UTED(trees $\hat{x}$ and $\hat{y}$, cost function $c$, heuristic $h$)
 2:     Initialize a priority queue $Q$ with the partial mapping $M = \{(1,1)\}$
 3:         and value $c(x_1, y_1) + h(\{2,\ldots,m\}, \{2,\ldots,n\})$.
 4:     while $Q$ is not empty do
 5:         Poll partial mapping $M$ with lowest value $f$ from $Q$.
 6:         $i \leftarrow \min\{1,\ldots,m+1\} \setminus I_M$.
 7:         if $i = m+1$ then
 8:             return $c(M \cup \{(0,j) | 1 \leq j \leq n, j \notin J_M\})$.
 9:         end if
10:         Retrieve $(k,l) \in M$ with largest $k$ such that $x_k$ is ancestor of $x_i$ and $l > 0$.
11:         $h^p \leftarrow h\big(\{1,\ldots,m\} \setminus (\mathcal{X}_k \cup I_M), \{1,\ldots,m\} \setminus (\mathcal{Y}_l \cup J_M)\big)$.
12:         $M_0 \leftarrow M \cup \{(i,0)\}$
13:         $h_0 \leftarrow h\big(\mathcal{X}_k \setminus I_{M_0}, \mathcal{Y}_l \setminus J_{M_0}\big) + h^p$.
14:         for $j \in \mathcal{Y}_l \setminus J_M$ do
15:             Let $y'_0, \ldots, y'_t$ be the path from $y_l$ to $y_j$ in $\hat{y}$ with $y'_0 = y_l$ and $y'_t = y_j$.
16:             $M_j \leftarrow M \cup \{(i,j), (0,y'_1), \ldots, (0,y'_{t-1})\}$.
17:             $h_j \leftarrow h\big(\mathcal{X}_i \setminus I_{M_j}, \mathcal{Y}_j \setminus J_{M_j}\big) + h\big(\mathcal{X}_k \setminus (\mathcal{X}_i \cup I_{M_j}), \mathcal{Y}_l \setminus (\mathcal{Y}_j \cup J_{M_j})\big) + h^p$.
18:         end for
19:         Put $M_j$ with value $c(M_j) + h_j$ onto $Q$ for all $j \in \{0\} \cup (\mathcal{Y}_l \setminus J_M)$.
20:     end while
21: end function
```

A algorithm:* We first introduce a few auxiliary concepts that we require for the A* algorithm. First, let M be some subset of $\{0,\ldots,m\} \times \{0,\ldots,n\}$. Then, we denote with I_M the set $\{i > 0 | \exists j : (i,j) \in M\}$ and with J_M the set $\{j > 0 | \exists i : (i,j) \in M\}$, i.e. the set of left-hand-side and right-hand-side indices of M. Next, let $\hat{x}$ and $\hat{y}$ be trees with nodes $x_1, \ldots, x_m$ and $y_1, \ldots, y_n$, respectively. Then, we define $\mathcal{X}_i$ and $\mathcal{Y}_j$ as the index sets of all descendants of x_i and y_j, respectively. Finally, let c be a cost function. Then, we define a *heuristic* as a function $h : \mathcal{P}(\{1,\ldots,m\}) \times \mathcal{P}(\{1,\ldots,n\}) \to \mathbb{R}$, such that for any $I \subseteq \{1,\ldots,m\}$ and $J \subseteq \{1,\ldots,n\}$ it holds

$$h(I,J) \leq \min_{M \in \mathcal{M}^{\mathrm{UTED}}(\hat{x},\hat{y})} \sum_{(i,j) \in M : i \in I, j \in J} c(x_i, y_j). \qquad (1)$$

Algorithm 1 shows the pseudocode for the A* algorithm. We initialize a partial mapping $M = \{(1, 1)\}$ which maps the root of $\hat{x}$ to the root of $\hat{y}$. If this is undesired, all input trees can be augmented with a placeholder root node. Next, we initialize a priority queue Q with M and its lower bound. Now, we enter the main loop. In each step, we consider the current partial mapping M with the lowest lower bound f (line 5). If I_M already covers all nodes in $\hat{x}$, we complete M by inserting all remaining nodes of $\hat{y}$ and return the cost of the resulting mapping (lines 7-9)[2]. Otherwise, we extend M by mapping the smallest non-mapped index i either to zero (lines 12-13), or to j for some available node y_j from $\hat{y}$ (lines 14-18). In the latter case, we need to maintain the ancestral ordering of the tree $\hat{y}$. Accordingly, we first retrieve the lowest ancestor x_k of x_i such that $(k, l) \in M$ with $l > 0$ and only permit i to be mapped to descendants $\mathcal{Y}_l$. Note that a $(k, l) \in M$ with $l > 0$ must exist because we initialized M with $\{(1, 1)\}$. Further, if we map i to a non-direct descendant of y_l, we make sure to insert all nodes on the ancestral path $y'_0, \ldots, y'_t$, first. We generate lower bounds h_j for all extensions M_j and put them back onto the priority queue (line 19).

Note that the space complexity of this algorithm can be polynomially limited by representing the partial mappings in a tree structure. However, the worst-case time complexity remains exponential because the algorithm may need to explore combinatorially many possible mappings. Generally, though, the tighter the lower bound provided by h, the fewer partial mappings need to be explored before we find a complete mapping. To further cut down the time complexity, we tabulate the lower bounds h^p for the ancestor mappings (k, l) (line 11), as recommended by Yoshino et al. [13].

Heuristics: The final ingredient we need is the actual heuristic h. We define three heuristics in increasing relaxation and decreasing time complexity. First, $h_3(I, J) = \min_{M \subseteq \mathcal{M}(I,J)} \sum_{(i,j)\in M} c(x_i, y_j)$, where $\mathcal{M}(I, J)$ denotes the set of all mappings between I and J, irrespective of ancestral ordering. Accordingly, Inequality 1 is trivially fulfilled because any mapping that respects ancestral ordering is also in $\mathcal{M}(I, J)$. Importantly, this relaxation can be solved in $\mathcal{O}((m+n)^3)$ via the Hungarian algorithm [4]. While polynomial, this appears rather expensive for a heuristic. Therefore, we also consider further relaxations. Without loss of generality, let $|I| \geq |J|$, otherwise exchange the roles of $\hat{x}$, $\hat{y}$, I and J. Then, we define

$$h_2(I, J) = \min_{I' \subseteq I : |I'| = |I| - |J|} \Big(\sum_{i \in I'} c(x_i, -) \Big) + \Big(\sum_{i \in I \setminus I'} \min_{j \in J} c(x_i, y_j) \Big). \tag{2}$$

Note that this is a lower bound for $h_3(I, J)$ because we expand the class of permitted mappings M to one-to-many mappings, which is a proper superset of $\mathcal{M}(I, J)$. Further, h_2 can be solved in $\mathcal{O}(m \cdot n)$ because we can evaluate $c_i := \min_{j \in J} c(x_i, x_j)$ for all i in $|I| \cdot |J|$ steps and we can solve the outer minimization by finding the $|I| - |J|$ smallest terms according to $c(x_i, -) - c_i$ and using those

[2] Strictly speaking, this is only valid if the lower bound f is exact for insertions. This is the case for all heuristics considered in this paper.

as I', which is possible in $O(|I|)$. In case even $\mathcal{O}(m \cdot n)$ is too expensive, we relax further to $h_1(I, J) = \min_{I' \subseteq I : |I'| = |I| - |J|} \sum_{i \in I'} c(x_i, -)$. This is obviously a lower bound for h_2 and can be solved in $\mathcal{O}(\max\{m, n\})$.

4 Experiments

We evaluate our three research questions on two data sets from Chemistry, namely the Alkanes data set of 150 alkane molecules by Micheli et al. [6] and the hundred smallest molecules from the ZINC molecule data set of Kusner et al. [8]. In the former case, the molecules are directly represented as trees (with 8.87 nodes on average) with hydrogen counts as node labels. In the latter case, we use the syntax tree of the molecule's SMILES representation (with 13.82 nodes on average) [12], where nodes are labeled with syntactic blocks. Note that this is a lossy representation because we cut aromatic rings to obtain trees.

Regarding RQ1, we compute all pairwise UTED values using the three heuristics h_1, h_2, and h_3, both with unit costs and with custom costs. As custom cost function c, we use the difference in hydrogen count between two carbon atoms for the alkanes data set. For the ZINC data set, we use the difference in electron count. For further reference, we also compare to the heuristic of Yoshino et al. [13] for unit costs. We execute all computations in Python on a consumer desktop PC with Intel core i9-10900 CPU and 32 GB RAM, and measure time using Python's `time` function. All experimental code is available at `https://gitlab.com/bpaassen/uted`.

Table 1. The average runtime in milliseconds (top) and the number of partial mappings searched (bottom) per distance computation for each heuristic.

		unit				custom		
	data set	h_1	h_2	h_3	h_{yoshino}	h_1	h_2	h_3
runtime	alkanes	8.70	12.15	10.72	9.52	**7.34**	8.21	9.92
	ZINC	549.38	277.15	192.97	266.66	130.62	75.53	**68.12**
search size	alkanes	376	348	260	279	318	302	**246**
	ZINC	24586	9164	4158	6781	6643	2655	**1379**

Table 1 shows the average runtime in milliseconds (top) for each heuristic on both data sets. On alkanes, h_1 is fastest and on ZINC, h_3 is fastest. All heuristics get faster for custom costs. Surprisingly, h_{yoshino} is not the fastest for unit costs, even though it is optimized for this setting. This may just be due to an unfavourable constant factor, though: h_{yoshino} is successful in reducing the size of the search space almost to the same level as h_3 (see Table 1, bottom). Further, Figure 2 displays a linear regression for the runtime versus tree size in a log-log plot, indicating that h_{yoshino} and h_3 have the lowest slopes/best scaling behavior for large trees.

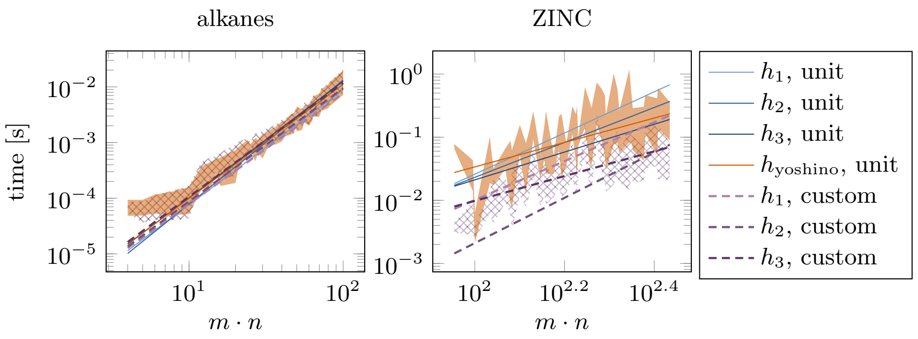


Fig. 2. A log-log regression of the runtime needed for computing UTED for all four heuristics (indicated by color) on the alkanes data (left) and the ZINC data (right). Shading indicates region between 25th and 75th percentile of the runtimes for h_{yoshino} (orange, solid), and h_3 with custom costs (purple, crosshatch), respectively.

Table 2. Average RMSE (± std.) of a 5-NN regressor across 15 crossvalidation folds for UTED, CUTED, and TED with unit and custom costs.

	unit			custom		
data set	UTED	CUTED	TED	UTED	CUTED	TED
alkanes	0.27 ± 0.24	0.27 ± 0.24	0.27 ± 0.24	**0.25 ± 0.24**	**0.25 ± 0.24**	**0.25 ± 0.24**
ZINC	1.33 ± 0.85	1.31 ± 0.86	1.36 ± 0.84	**1.24 ± 0.87**	1.26 ± 0.87	1.29 ± 0.86

Regarding RQ2 and RQ3, we perform a 5-nearest neighbor regression[3] to predict the boiling point of alkanes and the chemical stability measure of [8] for ZINC molecules, respectively. Table 2 shows the prediction error for both data sets in 15-fold crossvalidation. For reference, we do not only evaluate UTED with unit and custom costs, but also CUTED and TED. We observe that all methods perform better with custom costs compared to unit costs. For alkanes, there is no measurable difference between UTED, CUTED, and TED. For ZINC, TED performs worst, CUTED performs better than UTED for unit costs, and UTED performs better than CUTED for custom costs.

5 Conclusion

We proposed three novel heuristics to compute the unordered tree edit distance via an A* algorithm. In contrast to prior work, our heuristics can accommodate custom costs, not only unit costs. Our three heuristics provide different trade-offs of time complexity (linear, quadratic, cubic) versus how much they prune the A* search.

In our experiments on two chemical experiments, we observed that this trade-off works in favor of the linear heuristic for small trees but that the cubic heuristic

[3] We also tested lower K, which achieved worse results for all methods.

takes over for larger trees. Interestingly, the cubic heuristic compared favorably even to the current state-of-the-art heuristic. When applying custom costs, all our heuristics became faster thanks to the disambiguation provided by the custom cost function.

Regarding similarity search, we investigated the performance of a 5-nearest neighbor regressor, predicting chemical properties. We observed that custom costs lowered the regression error. However, we also saw that a similar performance can be achieved with a polynomial, restricted edit distance. Future work might investigate further tree data set to check whether these results generalize beyond chemistry.

References

1. Aiolli, F., Da San Martino, G., Sperduti, A.: An efficient topological distance-based tree kernel. IEEE TNNLS **26**(5), 1115–1120 (2015)
2. Augsten, N., Böhlen, M., Gamper, J.: The pq-gram distance between ordered labeled trees. ACM TDS **35**(1) (2008)
3. Bille, P.: A survey on tree edit distance and related problems. Theoretical Computer Science **337**(1), 217 – 239 (2005)
4. Bougleux, S., Brun, L., Carletti, V., Foggia, P., Gaüzère, B., Vento, M.: Graph edit distance as a quadratic assignment problem. Pat. Rec. Letters **87**, 38–46 (2017)
5. Collins, M., Duffy, N.: New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In: Proc. ACL. pp. 263–270 (2002)
6. Gallicchio, C., Micheli, A.: Tree echo state networks. Neurocomputing **101**, 319 – 337 (2013)
7. Horesh, Y., Mehr, R., Unger, R.: Designing an A* algorithm for calculating edit distance between rooted-unordered trees. J. Comp. Bio. **13**(6), 1165–1176 (2006)
8. Kusner, M.J., Paige, B., Hernández-Lobato, J.M.: Grammar variational autoencoder. In: Proc. ICML. pp. 1945–1954 (2017)
9. Paaßen, B., Gallicchio, C., Micheli, A., Hammer, B.: Tree Edit Distance Learning via Adaptive Symbol Embeddings. In: Proc. ICML. pp. 3973–3982 (2018)
10. Rarey, M., Dixon, J.S.: Feature trees: a new molecular similarity measure based on tree matching. J. comp.aided mol. design **12**(5), 471–490 (1998)
11. Shapiro, B.A., Zhang, K.: Comparing multiple RNA secondary structures using tree comparisons. Bioinformatics **6**(4), 309–318 (10 1990)
12. Weininger, D.: Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. J. Chem. Inf. and CS **28**(1), 31–36 (1988)
13. Yoshino, T., Higuchi, S., Hirata, K.: A dynamic programming A* algorithm for computing unordered tree edit distance. In: Proc. IIAI-AAI. pp. 135–140 (2013)
14. Zhang, K.: A constrained edit distance between unordered labeled trees. Algorithmica **15**(3), 205–222 (1996)
15. Zhang, K., Jiang, T.: Some max snp-hard results concerning unordered labeled trees. Inf. Proc. Letters **49**(5), 249–254 (1994)
16. Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. SIAM Computing **18**(6), 1245–1262 (1989)