[25] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Computing*, 18(6):1245–1262, Dec. 1989.

[26] K. Zhang, D. Shasha, and J. T. L. Wang. Approximate tree matching in the presence of variable length don't cares. Technical Report TR-328, Department of Computer Science, The University of Western Ontario, 1992.

[27] K. Zhang, D. Shasha, and J. T. L. Wang. Fast serial and parallel algorithms for approximate tree matching with VLDC's. In *Proc. 3rd Annual Symp. on Combinatorial Pattern Matching*, pages 148–158, Tucson, AZ, April 1992.

[8] S. R. Kosaraju. Efficient tree pattern matching. In *Proc. 30th Annual Symp. on Foundations of Computer Science*, pages 178–183, Oct. 1989.

[9] G. M. Landau and U. Vishkin. Fast parallel and serial approximate string matching. *J. Algorithms*, 10:157–169, 1989.

[10] J. Markowitz, T. Ahlswede, and M. Evans. Semantically significant patterns in dictionary definitions. In *Proc. Annual Meeting of the Association for Computational Linguistics*, pages 112–119, 1986.

[11] B. Moayer and K. S. Fu. A tree system approach for fingerprint pattern recognition. *IEEE Trans. Pattern Anal. Machine Intell.*, 8:376–387, May 1986.

[12] E. W. Myers and W. Miller. Approximate matching of regular expressions. *Bulletin of Mathematical Biology*, 51(1):5–37, 1989.

[13] M. Neff and B. K. Boguraev. Dictionaries, dictionary grammars and dictionary entry parsing. In *Proc. 27th Annual Meeting of the Association for Computational Linguistics*, June 1989.

[14] M. Neff, R. Byrd, and O. Rizk. Creating and querying hierarchical lexical data bases. In *Proc. 2nd Conf. Applied Natural Language Processing*, pages 84–93, 1988.

[15] H. Samet. Distance transform for images represented by quadtrees. *IEEE Trans. on Pattern Anal. Machine Intell.*, 4(3):298–303, May 1982.

[16] P. H. Sellers. The theory and computation of evolutionary distances. *J. Algorithms*, 1:359–373, 1980.

[17] B. A. Shapiro and K. Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Comput. Appl. Biosci.*, 6(4):309–318, 1990.

[18] D. Shasha and K. Zhang. Fast algorithms for the unit cost editing distance between trees. *J. Algorithms*, 11(4):581–621, 1990.

[19] T. F. Smith and W. A. Beyer. Some biological sequence metrics. *Adv. Maths*, 20:367–387, 1976.

[20] J. L. Sussman and S. H. Kim. Three dimensional structure of a transfer RNA in two crystal forms. *Science*, 192:853, 1976.

[21] K.-C. Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.

[22] E. Ukkonen. Finding approximate patterns in strings. *J. Algorithms*, 6:132–137, 1985.

[23] J. T. L. Wang, K. Jeong, K. Zhang, and D. Shasha. Reference manual for ATBE: A tool for approximate tree matching. Technical Report TR-551, Courant Institute of Mathematical Sciences, New York University, Mar. 1991.

[24] J. T. L. Wang, K. Zhang, K. Jeong, and D. Shasha. A system for approximate tree matching. *IEEE Trans. on Knowledge and Data Engineering*, in press.

7 Conclusion

Many application domains make use of trees. Whenever an application needs to compare trees, it is natural to ask about approximate matching. Variable length don't cares extend the power of approximate matching by allowing a query to suppress certain details about a tree.

We presented two different definitions for VLDC's that are natural generalizations of VLDC's in strings, and formulated the problems of approximate tree matching with the VLDC's. We then introduced a new suffix forest distance measure for solving the problems. Our algorithms differ in the particular semantics they give to the VLDC's, but share the same time complexity. In [26,27], we presented a parallel version of the algorithms. In our toolkit [23,24], we have found that these algorithms work very well in practice too. In fact, the toolkit also generates a best mapping having the distance between the pattern and the data trees, still preserving the time complexity.

8 Acknowledgements

The authors wish to thank members of the lexical systems project at IBM T. J. Watson Research Center, in particular B. Boguraev, R. Byrd, M. Chodorow, J. Klavans, M. Neff, and Y. Ravin for helpful discussions concerning this work. We also thank the anonymous referees, whose excellent comments helped to improve the paper.

References

[1] A. V. Aho, M. Ganapathi, and S. W. K. Tjiang. Code generation using tree matching and dynamic programming. *ACM Trans. Programming Languages and Systems*, 11(4):491–516, Oct. 1989.

[2] R. Byrd. LQL user notes: An informal guide to the lexical query language. Technical report, IBM T. J. Watson Research Center, New York, 1990.

[3] M. Chodorow and J. L. Klavans. Locating syntactic patterns in text corpora. Technical report, IBM T. J. Watson Research Center, New York, 1990.

[4] M. S. Chodorow, R. J. Byrd, and G. E. Heidorn. Extracting semantic hierarchies from a large on-line dictionary. In *Proc. Annual Meetings of the Association for Computational Linguistics*, pages 299–304, 1985.

[5] M. Dubiner, Z. Galil, and E. Magen. Faster tree pattern matching. In *Proc. 31st Annual Symp. on Foundations of Computer Science*, pages 145–150, Oct. 1990.

[6] Z. Galil and K. Park. An improved algorithm for approximate string matching. *SIAM J. Comput.*, 19(6):989–999, Dec. 1990.

[7] C. M. Hoffmann and M. J. O'Donnell. Pattern matching in trees. *J. ACM*, 29:68–95, 1982.

2. when $P[s] \in i$-path of P, $D[t] \in j$-path of D, and $P[s]$ is not a VLDC symbol, lines 18, 26 in the procedure **treedist**(i,j) (cf. Lemmas 4.3, 4.7) and line 13 in the procedure **treedist_cut**(i,j) (cf. Lemma 5.4) can be calculated in $O(1)$ time;

3. when $P[s] \in i$-path of P, $D[t] \in j$-path of D, and $P[s]$ is a VLDC symbol, lines 20, 22 in the procedure **treedist**(i,j) (cf. Lemmas 4.4, 4.5) and line 15 in the procedure **treedist_cut**(i,j) (cf. Lemma 5.5) can be calculated in $O(deg(t))$ time.

Lemma 6.4. *(i)* $\sum_{i \in LR_keyroots(P)} N_i =$ # of VLDC's in P; *(ii)* $\sum_{j \in LR_keyroots(D)} C_j = |D| - 1$.

Proof. For (i), observe that each node in P is on an i-path for some $i \in LR_keyroots(P)$. Further, i-path $\cap$ i'-path $= \emptyset$, for any i, $i' \in LR_keyroots(P)$ where $i \neq i'$. Therefore the i-paths corresponding to $LR_keyroots(P)$ partition P. So, $\sum_{i \in LR_keyroots(P)} N_i =$ # of VLDC's in P. Similarly, $\sum_{j \in LR_keyroots(D)} C_j = \sum_{j=1}^{|D|} deg(j) = |D| - 1$. □

Theorem 6.1. *Given the pattern P and data tree D where P may contain VLDC symbols, our algorithms compute the distance from P to D (with and without cut) in $O(|P| \times |D| \times min(depth(P), leaves(P)) \times min(depth(D), leaves(D)))$ time and use $O(|P| \times |D|)$ space.*

Proof. The preprocessing in the main framework in Section 3 takes linear time. We invoke the procedure **treedist**(i,j) (or **treedist_cut**(i,j)) for each $i \in LR_keyroots(P)$ and $j \in LR_keyroots(D)$. By Lemma 6.3, this takes time

$$O(\sum_{i \in LR_keyroots(P)} \sum_{j \in LR_keyroots(D)} Size(i) \times Size(j) + N_i \times C_j)$$

$$= O(\sum_{i \in LR_keyroots(P)} \sum_{j \in LR_keyroots(D)} Size(i) \times Size(j) + \sum_{i \in LR_keyroots(P)} \sum_{j \in LR_keyroots(D)} N_i \times C_j).$$

By Lemma 6.2, the first term equals

$$\sum_{i=1}^{|P|} LR_colldepth(i) \times \sum_{j=1}^{|D|} LR_colldepth(j),$$

which is less than

$$|P| \times |D| \times LR_colldepth(P) \times LR_colldepth(D).$$

By Lemma 6.1 and the definition of $LR_colldepth()$, we can see that, for a tree T, $LR_colldepth(i) \leq \min(depth(T), leaves(T))$ for $1 \leq i \leq |T|$. So, $LR_colldepth(P) \leq \min(depth(P), leaves(P))$ and $LR_colldepth(D) \leq \min(depth(D), leaves(D))$. Hence the first term is bounded by $O(|P| \times |D| \times \min(depth(P), leaves(P)) \times \min(depth(D), leaves(D)))$. By Lemma 6.4, the second term is bounded by (# of VLDC's in P) $\times$ $(|D| - 1)$, which is in turn bounded by $|P| \times |D|$. Therefore the time complexity is $O(|P| \times |D| \times \min(depth(P), leaves(P)) \times \min(depth(D), leaves(D)))$.

The space complexity is obtained by noting that the algorithms employ a global array for $treedist$, and at most two local arrays for $forestdist$ and sfd, each of which needs $O(|P| \times |D|)$ space. □

$l(s) = l(i)$ and $l(t) = l(j)$, and put them in the global array (line 16). Thus, no value is used before it is computed. Lemmas 5.3 to 5.5 show that the computed values are correct. It follows that our algorithm is also correct. □

6 Resource Analysis of the Algorithms

Lemma 6.1. $|LR_keyroots(T)| \leq |leaves(T)|$.

Proof. See [25, Lemma 6]. □

Since not all subtree-to-subtree distances need to be computed, the number of such calculations a node participates in is less than its depth. In fact, it is the node's *collapsed depth*:

$$LR_colldepth(i) = |anc(i) \cap LR_keyroots(T)|.$$

We define the collapsed depth of tree T as follows:

$$LR_colldepth(T) = \max_{1 \leq i \leq |T|} \{LR_colldepth(i)\}.$$

Lemma 6.2.

$$\sum_{i \in LR_keyroots(T)} Size(i) = \sum_{j=1}^{|T|} LR_colldepth(j).$$

Proof. See [25, Lemma 7]. □

Given a tree T, we define the *i-path* of T to be the path from $T[l(i)]$ to $T[i]$. Let N_i be the number of VLDC's on the i-path of P, and

$$C_j = \sum_{D[k] \in j\text{-path of } D} deg(k).$$

Lemma 6.3. *The time required for computing the procedures* **treedist**(i, j) *of Figure 10 and* **treedist_cut**(i, j) *of Figure 17 is* $O(Size(i) \times Size(j) + N_i \times C_j)$.

Proof. Suppose $s \in des(i)$ and $t \in des(j)$. We obtain the time complexity by observing the following:

1. when $P[s] \notin i$-path of P or $D[t] \notin j$-path of D, lines 15, 28 in the procedure **treedist**(i, j) (cf. Lemmas 4.2, 4.8) and line 10 in the procedure **treedist_cut**(i, j) (cf. Lemma 5.3) can be calculated in $O(1)$ time;

We are now ready to give the algorithm for the procedure **treedist_cut**(i,j) (see Figure 17). A local array forestdist_cut, which will hold distance values $forestdist_cut$, is allocated when invoking the procedure and is freed once exiting it.

Procedure treedist_cut(i,j)

```
1.  forestdist_cut(∅,∅) := 0;        /* Lemma 5.2 */
2.  for s := l(i) to i do
3.      forestdist_cut(P[l(i)..s],∅) := forestdist_cut(P[l(i)..s − 1],∅) + γ(P[s] → λ);
4.  for t := l(j) to j do
5.      forestdist_cut(∅, D[l(j)..t]) := 0;
6.  for s := l(i) to i do
7.      for t := l(j) to j do
8.          begin
9.              if l(s) ≠ l(i) or l(t) ≠ l(j) then        /* Lemma 5.3 */
10.                 forestdist_cut(l(i)..s, l(j)..t) := min {
                        forestdist_cut(l(i)..s, l(j)..l(t) − 1),
                        forestdist_cut(l(i)..s − 1, l(j)..t) + γ(P[s] → λ),
                        forestdist_cut(l(i)..s, l(j)..t − 1) + γ(λ → D[t]),
                        forestdist_cut(l(i)..l(s) − 1, l(j)..l(t) − 1) + treedist_cut(s,t)
                        }
11.             else begin        /* l(s) = l(i) and l(t) = l(j) */
12.                 if (P[s] ≠ |) then        /* Lemma 5.4 */
13.                     forestdist_cut(l(i)..s, l(j)..t) := min {
                            forestdist_cut(P[l(i)..s],∅),
                            forestdist_cut(l(i)..s − 1, l(j)..t) + γ(P[s] → λ),
                            forestdist_cut(l(i)..s, l(j)..t − 1) + γ(λ → D[t]),
                            forestdist_cut(l(i)..s − 1, l(j)..t − 1) + γ(P[s] → D[t])
                            }
14.                 if (P[s] = |) then (let t_1, t_2, ..., t_{n_t} be the children of t)        /* Lemma 5.5 */
15.                     forestdist_cut(l(i)..s, l(j)..t) := min {
                            forestdist_cut(P[l(i)..s],∅),
                            forestdist_cut(l(i)..s − 1, l(j)..t) + γ(P[s] → λ),
                            forestdist_cut(l(i)..s, l(j)..t − 1) + γ(λ → D[t]),
                            forestdist_cut(l(i)..s − 1, l(j)..t − 1) + γ(P[s] → D[t]),
                            min_{t_k} {treedist_cut(s,t_k) | 1 ≤ k ≤ n_t}        /* present only if t ≠ l(j). */
                            }
16.                 treedist_cut(s,t) := forestdist_cut(l(i)..s, l(j)..t)
17.             end (else)
18.         end (for);
```

Fig. 17. Algorithm for the procedure **treedist_cut**(i,j).

The following theorem shows the correctness of the algorithm and establishes the basic assumptions (for the "with cut" case) in Section 3.

Theorem 5.1. *The algorithm for the procedure* **treedist_cut**(i,j) *in Figure 17 is correct.*

Proof. By Lemma 5.2, the initialization of $forestdist_cut()$ is correct. In computing $forestdist_cut()$, we only use such $treedist_cut(s,t)$ values where $l(i) \le s \le i$ and $l(j) \le t \le j$ and either $l(s) \ne l(i)$ or $l(t) \ne l(j)$ (line 10). We compute the $treedist_cut(s,t)$ values where $l(i) \le s \le i$ and $l(j) \le t \le j$ and both

$$forestdist_cut(l(i)..s, l(j)..t) = \mathbf{min} \begin{cases} forestdist_cut(P[l(i)..s], \emptyset), \\ forestdist_cut(l(i)..s-1, l(j)..t) + \gamma(P[s] \rightarrow \lambda), \\ forestdist_cut(l(i)..s, l(j)..t-1) + \gamma(\lambda \rightarrow D[t]), \\ forestdist_cut(l(i)..s-1, l(j)..t-1) + \gamma(P[s] \rightarrow D[t]) \end{cases}$$

Proof. If $tree(t)$ is cut, then the distance should be $forestdist(P[l(i)..s], \emptyset)$. Otherwise, let M be a minimum-cost mapping between $P[l(i)..s]$ and $D[l(j)..t]$ after performing an optimal removal of subtrees of $D[l(j)..t]$. Depending on whether $P[s]$ or $D[t]$ is touched by a line in M, again we argue similarly as in case 1 of Lemma 4.2. □

Lemma 5.5. *Suppose $s \in des(i)$ and $t \in des(j)$. Suppose both $P[l(i)..s]$ and $D[l(j)..t]$ are trees (i.e., $l(s) = l(i)$ and $l(t) = l(j)$). If $P[s] = |$, then*

$$forestdist_cut(l(i)..s, l(j)..t) = \mathbf{min} \begin{cases} forestdist_cut(P[l(i)..s], \emptyset), \\ forestdist_cut(l(i)..s-1, l(j)..t) + \gamma(P[s] \rightarrow \lambda), \\ forestdist_cut(l(i)..s, l(j)..t-1) + \gamma(\lambda \rightarrow D[t]), \\ forestdist_cut(l(i)..s-1, l(j)..t-1) + \gamma(P[s] \rightarrow D[t]), \\ \min_{t_k}\{treedist_cut(s, t_k) | 1 \le k \le n_t\} \end{cases}$$

where $D[t_k]$, $1 \le k \le n_t$, are children of $D[t]$. (If $D[t]$ is a leaf, i.e., $t = l(j)$, then only the first four expressions are present.)

Proof. If $tree(t)$ is cut, then the distance should be $forestdist_cut(P[l(i)..s], \emptyset)$. Otherwise, let M be a minimum-cost mapping between $P[l(i)..s]$ and $D[l(j)..t]$ after performing an optimal removal of subtrees of $D[l(j)..t]$. There are three cases to examine: in the best substitution, (1) $P[s]$ is replaced by an empty tree, (2) $P[s]$ is replaced by a nonempty tree and $D[t]$ is not touched by a line in M, and (3) $P[s]$ is replaced by a nonempty tree and $D[t]$ is touched by a line in M.

Cases 1 and 2 are similar to cases 2(a) and 2(b) of Lemma 4.2, and hence we get the same formulae derived there. For case 3, if $t = l(j)$, then $P[s]$ must be replaced by $D[t]$. So, $forestdist_cut(l(i)..s, l(j)..t) = forestdist_cut(l(i)..s-1, l(j)..t-1) + \gamma(P[s] \rightarrow D[t])$. The distance is the minimum of the three corresponding costs. This completes the proof for $t = l(j)$.

If $t \neq l(j)$, then $P[s]$ must be replaced by a path of the tree rooted at $D[t]$. Let the path end at node $D[d]$. Let the children of $D[t]$ be, in left-to-right order, $D[t_1], D[t_2], \ldots, D[t_{n_t}]$. There are two subcases to examine:

(a) $d = t$. Thus, $P[s]$ is replaced by $D[t]$. So $forestdist_cut(l(i)..s, l(j)..t) = forestdist_cut(l(i)..s-1, l(j)..t-1) + \gamma(P[s] \rightarrow D[t])$.

(b) $d \neq t$. Let $D[t_k]$ be the child of $D[t]$ on the path from $D[t]$ to $D[d]$. We can cut all subtrees on the two sides of the path. So, $forestdist_cut(l(i)..s, l(j)..t) = treedist_cut(s, t_k)$. The value of k ranges from 1 to n_t. Hence, $forestdist_cut(l(i)..s, l(j)..t) = \min_{t_k} \{treedist_cut(s, t_k) \mid 1 \le k \le n_t\}$. □

Next, assume that N is labeled by $|$. Let the path of the data tree that N substitutes for be P. Then the subtrees on the two sides of P must be cut, since otherwise the distance would be higher. Now, we could replace N with a node labeled by $\wedge$ and let it match the path P as well as the subtrees on the two sides of P. The cost of the resulting mapping would be no greater than the original distance.

Therefore, with cut, there is no difference between $|$ and $\wedge$. □

Because of this lemma, we shall focus on pattern trees containing $|$'s only.

Lemma 5.2. *Suppose $s \in des(i)$ and $t \in des(j)$. Then*
(i) $forestdist_cut(\emptyset, \emptyset) = 0$;
(ii) $forestdist_cut(\emptyset, D[l(j)..t]) = 0$;
(iii) $forestdist_cut(P[l(i)..s], \emptyset) = forestdist_cut(P[l(i)..s-1], \emptyset) + \gamma(P[s] \rightarrow \lambda)$.

Proof. (i) requires no edit operation. (ii) is straightforward because we can cut all the subtrees in the forest $D[l(j)..t]$. For (iii), nothing can be cut from the data tree, and the case corresponds to deleting a node from the pattern tree. (As in (ii) of Lemma 4.1, adding $\gamma(P[s] \rightarrow \lambda)$ to the right hand side of the formula in (iii) does not change the cost when $P[s] = |$.) □

In deriving formulae in the following lemmas, we adopt the same strategy: try to find a best substitution for $|$'s in $P[l(i)..s]$, and then ask whether or not the subtree rooted at $D[t]$ (i.e., $tree(t)$) is cut.

Lemma 5.3. *Suppose $s \in des(i)$ and $t \in des(j)$. If $l(s) \neq l(i)$ or $l(t) \neq l(j)$, then*

$$forestdist_cut(l(i)..s, l(j)..t) = \mathbf{min} \begin{cases} forestdist_cut(l(i)..s, l(j)..l(t)-1), \\ forestdist_cut(l(i)..s-1, l(j)..t) + \gamma(P[s] \rightarrow \lambda), \\ forestdist_cut(l(i)..s, l(j)..t-1) + \gamma(\lambda \rightarrow D[t]), \\ forestdist_cut(l(i)..l(s)-1, l(j)..l(t)-1) + treedist_cut(s,t) \end{cases}$$

Proof. If $tree(t)$ is cut, then the distance should be $forestdist_cut(l(i)..s, l(j)..l(t)-1)$. Otherwise, consider a minimum-cost mapping M between $P[l(i)..s]$ and $D[l(j)..t]$ after performing an optimal removal of subtrees of $D[l(j)..t]$. As in case 1 of Lemma 4.2, the distance is the minimum of the following three cases: (1) $P[s]$ is not touched by a line in M (this includes the case where $P[s] = |$ is replaced by an empty tree), (2) $D[t]$ is not touched by a line in M, and (3) $P[s]$ and $D[t]$ are both touched by lines in M (this includes the case where $P[s] = |$ is replaced by a path of nodes in the data tree). □

Lemma 5.4. *Suppose $s \in des(i)$ and $t \in des(j)$. Suppose both $P[l(i)..s]$ and $D[l(j)..t]$ are trees (i.e., $l(s) = l(i)$ and $l(t) = l(j)$). If $P[s] \neq |$, then*

where $l(i) \leq s \leq i$ and $l(j) \leq t \leq j$ and either $l(s) \neq l(i)$ or $l(t) \neq l(j)$, and the procedure **treedist**(i,j) yields $treedist(s,t)$ where $l(i) \leq s \leq i$ and $l(j) \leq t \leq j$ and both $l(s) = l(i)$ and $l(t) = l(j)$ (cf. the basic assumptions in Section 3).

Theorem 4.1. *The algorithm for the procedure* **treedist**(i,j) *in Figure 10 is correct.*

Proof. By Lemma 4.1, the initialization of $forestdist()$ is correct. In initializing $sfd()$, we use the initial values of $forestdist()$ (cf. Lemma 4.6 and line 10), which are available because we initialize $forestdist()$ first (cf. lines 1-9). So, the initialization of $sfd()$ is also correct.

Next, in calculating $forestdist(l(i)..s, l(j)..t)$, we need $sfd(l(i)..s-1, l(j)..t_k)$, $1 \leq k \leq n_t$, where $D[t_k]$ is a child of $D[t]$ (cf. Lemma 4.5 and line 22). These distance values are available, because we have computed $forestdist(l(i)..s', l(j)..t')$ and $sfd(l(i)..s', l(j)..t')$ where $s' \leq s$, $t' \leq t$, and at least one inequality must hold (cf. Lemmas 4.6, 4.7, 4.8 and lines 10, 26, 28). On the other hand, in calculating $sfd(l(i)..s, l(j)..t)$, we need $forestdist(l(i)..s, l(j)..t)$ (cf. Lemma 4.7 and line 26). These distance values are again available, because we have also computed them.

Now, during the computation of $forestdist()$ and $sfd()$, we only use such $treedist(s,t)$ values where $l(i) \leq s \leq i$ and $l(j) \leq t \leq j$ and either $l(s) \neq l(i)$ or $l(t) \neq l(j)$ (cf. Lemmas 4.2, 4.8 and lines 15, 28). We compute the $treedist(s,t)$ values where $l(i) \leq s \leq i$ and $l(j) \leq t \leq j$ and both $l(s) = l(i)$ and $l(t) = l(j)$, and put them in the global array (cf. line 23). Therefore, no value is used before it is computed, establishing the basic assumption of Section 3. The lemmas show that the computed values are correct. It follows that our algorithm is also correct. □

5 Algorithm for treedist_cut(i,j)

As it happens, the "with cut" case is the easiest, because no auxiliary distance measures such as $sfd()$ are needed. So this section is formally independent of the previous one. We present five lemmas and then give the algorithm. (Note that, as in the procedure **treedist**(i,j) of Figure 10, $\gamma(P[s] \rightarrow \lambda) = 0$ and $\gamma(P[s] \rightarrow D[t]) = 0$ when $P[s]$ is either | or $\wedge$.)

Lemma 5.1. *A path-VLDC can be substituted for an umbrella-VLDC or vice versa without changing the mapping or the distance value when we allow subtrees to be cut freely from the data tree.*

Proof. Consider a pattern tree having both |'s and $\wedge$'s. We try to find a best substitution for the VLDC's, and then consider a minimum-cost mapping from the pattern tree to the data tree after performing an optimal removal of subtrees of the data tree.

Now, concentrate on an (arbitrary) VLDC node N in the pattern tree. Assume that N is labeled by $\wedge$. Let the portion of the data tree that N substitutes for be U. Then, we could replace N with a node labeled by |, let it match the path in U and cut all other subtrees not on the path. The cost of the resulting mapping would be no greater than the original distance.

We try to find a best substitution for |'s and $\wedge$'s in $P[l(i)..s]$ and then consider a minimum-cost mapping M from $P[l(i)..s]$ to $D[l(j)..t]$ after performing an optimal removal of a left subforest F (consisting of consecutive complete trees with the same parent) of forest $D[l(j)..t]$. There are two cases to examine:

Case 1: $l(par(t)) = l(j)$ (cf. Figure 15). There are four subcases to be considered:

(a) The subtree rooted at $D[t]$ is not removed and $P[s]$ is not touched by a line in M. So, $sfd(l(i)..s, l(j)..t) = sfd(l(i)..s-1, l(j)..t) + \gamma(P[s] \rightarrow \lambda)$.

(b) The subtree rooted at $D[t]$ is not removed and $D[t]$ is not touched by a line in M.

Claim: The best left subforest removed from $D[l(j)..t]$ (i.e., F) is the best for $D[l(j)..t-1]$. Moreover, the best mapping between $P[l(i)..s]$ and $D[l(j)..t] - F$ is also the best mapping between $P[l(i)..s]$ and $D[l(j)..t-1] - F$.

Proof of Claim: Suppose not; consider the best left subforest removed in $D[l(j)..t-1]$ (call it F_{t-1}) and the corresponding best mapping M_{t-1}. Since the removed subtrees in F_{t-1} have $D[par(t)]$ as their parent, they do not include any subtree rooted at a child of $D[t]$. Hence, we could remove F_{t-1} from $D[l(j)..t]$ and apply M_{t-1} to map nodes in $P[l(i)..s]$ to those in $D[l(j)..t] - F_{t-1}$; the resulting cost would be smaller than that of using M between $P[l(i)..s]$ and $D[l(j)..t] - F$, a contradiction. □

Therefore, $sfd(l(i)..s, l(j)..t) = sfd(l(i)..s, l(j)..t-1) + \gamma(\lambda \rightarrow D[t])$.

(c) The subtree rooted at $D[t]$ is not removed and $P[s]$ and $D[t]$ are both touched by lines in M. So, $sfd(l(i)..s, l(j)..t) = sfd(l(i)..l(s)-1, l(j)..l(t)-1) + treedist(s,t)$.

(d) The entire forest $D[l(j)..t]$ is removed. So, $sfd(l(i)..s, l(j)..t) = forestdist(P[l(i)..s], \emptyset) = forestdist(P[l(i)..s-1], \emptyset) + \gamma(P[s] \rightarrow \lambda)$.

Case 2: $l(par(t)) \neq l(j)$ (cf. Figure 16). There are three subcases to be considered (the subcase 1(d) doesn't apply here since we can not remove the whole forest!):

(a) $P[s]$ is not touched by a line in M. So, $sfd(l(i)..s, l(j)..t) = sfd(l(i)..s-1, l(j)..t) + \gamma(P[s] \rightarrow \lambda)$.

(b) $D[t]$ is not touched by a line in M. Since the subtree rooted at $D[t]$ can not be removed, $sfd(l(i)..s, l(j)..t) = sfd(l(i)..s, l(j)..t-1) + \gamma(\lambda \rightarrow D[t])$. Note that since $D[l(j)..t]$ is a forest and $l(par(t)) \neq l(j)$, a collection of consecutive complete subtrees from the left of $D[l(j)..t-1]$ with the same parent will not include any subtree rooted at a child of $D[t]$ (the reasons are similar to those in case 2 of (iii) in Lemma 4.6 and case 1(b) above). So, we can use the recursive formula.

(c) Both $P[s]$ and $D[t]$ are touched by lines in M. So, $sfd(l(i)..s, l(j)..t) = sfd(l(i)..l(s)-1, l(j)..l(t)-1) + treedist(s,t)$.

Now try to combine the above two cases. In case 1, we do not need subcase (d).[9] Thus, the formulae of the two cases are exactly the same. Furthermore, we can combine these formulae with that obtained from the case where $P[s] =$ | or $P[s] = \wedge$. This gives the formula asserted by the lemma. □

Since in the computation of $forestdist()$ ($sfd()$, respectively), we use the values of $sfd()$ ($forestdist()$, respectively), we still need to prove that whenever we need a distance value, it is indeed available. Also, we need to prove that in the execution of the procedure **treedist**(i,j) in Figure 10, we only use $treedist(s,t)$

[9] Subcase 1(d) is covered in subcase 1(a), since $sfd(l(i)..s-1, l(j)..t) \leq forestdist(P[l(i)..s-1], \emptyset)$. (The latter represents a particular case where the whole forest is removed whereas the former includes not only that particular case but other cases where only a certain subforest in $D[l(j)..t]$ is removed.) So, subcase 1(d) can be eliminated.

Case 2: $l(par(t)) \neq l(j)$. (See Figure 16). Thus, a collection of consecutive complete subtrees from the left of $D[l(j)..t-1]$ with the same parent will not include any subtree rooted at a child of $D[t]$ (because otherwise these subtrees would have $D[t]$ as their parent, which would imply $l(par(t)) = l(j)$). This means that the best collection of subtrees to be removed in the forest $D[l(j)..t-1]$ is the same as that in $D[l(j)..t]$. Therefore, $sfd(\emptyset, D[l(j)..t]) = sfd(\emptyset, D[l(j)..t-1]) + \gamma(\lambda \rightarrow D[t])$. □

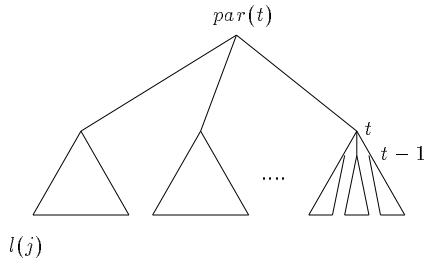


Fig. 15. Case 1 of (iii) in Lemma 4.6.

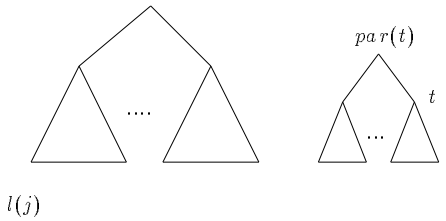


Fig. 16. Case 2 of (iii) in Lemma 4.6.

Lemma 4.7. (Computation of suffix_dist_for_tree(t)) *Suppose $s \in des(i)$ and $t \in des(j)$. If $l(t) = l(j)$, then*

$$sfd(l(i)..s, l(j)..t) = \mathbf{min} \begin{cases} forestdist(P[l(i)..s], \emptyset), \\ forestdist(l(i)..s, l(j)..t) \end{cases}$$

Proof. By assumption, $D[l(j)..t]$ is a tree. Thus, we have two choices:

Case 1: It is best to remove the tree. So, $sfd(l(i)..s, l(j)..t) = forestdist(P[l(i)..s], \emptyset)$.

Case 2: It is best not to remove the tree. So, $sfd(l(i)..s, l(j)..t) = forestdist(l(i)..s, l(j)..t)$.

□

Lemma 4.8. (Computation of suffix_dist_for_forest($l(j)..t$)) *Suppose $s \in des(i)$ and $t \in des(j)$. If $l(t) \neq l(j)$, then*

$$sfd(l(i)..s, l(j)..t) = \mathbf{min} \begin{cases} sfd(l(i)..s-1, l(j)..t) + \gamma(P[s] \rightarrow \lambda), \\ sfd(l(i)..s, l(j)..t-1) + \gamma(\lambda \rightarrow D[t]), \\ sfd(l(i)..l(s)-1, l(j)..l(t)-1) + treedist(s,t) \end{cases}$$

Proof. We will only consider the situation where $P[s] \neq |$ and $P[s] \neq \wedge$. The case in which $P[s] = |$ or $P[s] = \wedge$ can be proved similarly as case 2 of Lemma 4.2.

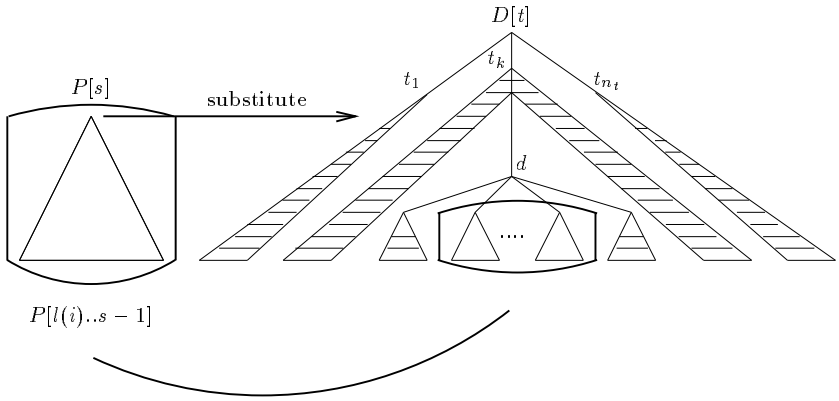


Fig. 13. Case 3(a) of Lemma 4.5. The shaded subtrees are used for substitution; the unshaded subtrees map to $P[l(i)..s-1]$.

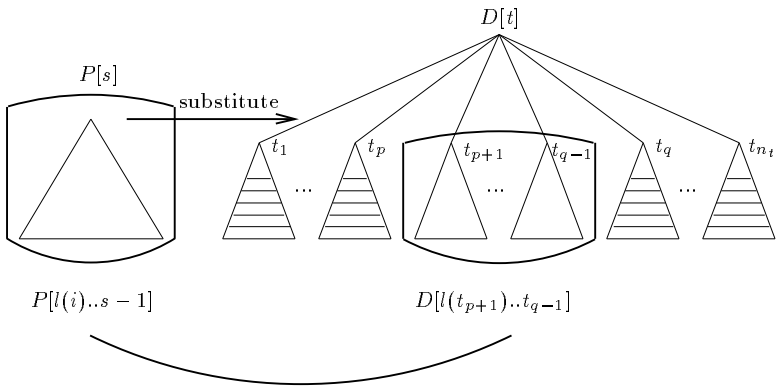


Fig. 14. Case 3(b) of Lemma 4.5. The shaded subtrees are used for substitution; the unshaded subtrees in the forest $D[l(t_{p+1})..t_{q-1}]$ map to $P[l(i)..s-1]$.

The following lemmas show the computation of the suffix forest distance.

Lemma 4.6. (Initialization of $sfd()$) *Suppose $s \in des(i)$ and $t \in des(j)$. Then*
(i) $sfd(\emptyset, \emptyset) = 0$;
(ii) $sfd(P[l(i)..s], \emptyset) = forestdist(P[l(i)..s], \emptyset)$;
(iii) If $l(t) = l(j)$ *or* $l(par(t)) = l(j)$ *then* $sfd(\emptyset, D[l(j)..t]) = 0$ *else* $sfd(\emptyset, D[l(j)..t]) = sfd(\emptyset, D[l(j)..t-1]) + \gamma(\lambda \rightarrow D[t])$.

Proof. (i) is trivial. (ii) follows by observing that nothing can be removed from $\emptyset$, and therefore the sfd must be equivalent to the $forestdist$. For (iii), if $l(t) = l(j)$, $D[l(j)..t]$ is a tree. It would be best to remove the entire tree. So, $sfd(\emptyset, D[l(j)..t]) = 0$. If $l(t) \neq l(j)$, $D[l(j)..t]$ is a forest. Then at least one of the following two cases must hold:

Case 1: $l(par(t)) = l(j)$. Then, the subtrees in the forest $D[l(j)..t]$ have the same parent, namely $D[par(t)]$ (Figure 15). Thus, we can remove the whole forest. So, $sfd(\emptyset, D[l(j)..t]) = 0$.

$$forestdist(l(i)..s, l(j)..t) = \mathbf{min} \begin{cases} forestdist(l(i)..s-1, l(j)..t) + \gamma(P[s] \rightarrow \lambda), \\ forestdist(l(i)..s, l(j)..t-1) + \gamma(\lambda \rightarrow D[t]), \\ forestdist(l(i)..s-1, l(j)..t-1) + \gamma(P[s] \rightarrow D[t]), \\ \min_{t_k}\{treedist(s, t_k) | 1 \leq k \leq n_t\}, \\ \min_{t_k}\{sfd(l(i)..s-1, l(j)..t_k) | 1 \leq k \leq n_t\} \end{cases}$$

where $D[t_k]$, $1 \leq k \leq n_t$, are children of $D[t]$. (If $D[t]$ is a leaf, i.e., $t = l(j)$, then only the first three expressions are present.)

Proof. First, consider the simple case where $s = l(i)$. In this case, we have a single node $P[s]$ in $P[l(i)..s]$. Thus, the best substitution is to replace $P[s]$ by the entire subtree rooted at $D[t]$, resulting in a cost of zero.

Next, consider the the more interesting case where $s \neq l(i)$. We first try to find a best substitution for |'s and $\wedge$'s in the tree $P[l(i)..s]$, and consider a minimum-cost mapping M between $P[l(i)..s]$ and $D[l(j)..t]$. Then, at least one of the following three cases must hold: in the best substitution, (1) $P[s]$ is replaced by an empty tree, (2) $P[s]$ is replaced by a nonempty tree and $D[t]$ is not touched by a line in M, and (3) $P[s]$ is replaced by a nonempty tree and $D[t]$ is touched by a line in M.

Cases 1 and 2 are similar to cases 2(a) and 2(b) in Lemma 4.2, and hence we get the same formulae derived there. For case 3, if $t = l(j)$, then $P[s]$ must be replaced by $D[t]$. So, $forestdist(l(i)..s, l(j)..t) = forestdist(l(i)..s-1, l(j)..t-1) + \gamma(P[s] \rightarrow D[t])$. The distance is the minimum of the three corresponding costs. This completes the proof for $t = l(j)$.

If $t \neq l(j)$, then $P[s]$ must be replaced by an umbrella pattern of nodes rooted at $D[t]$. Recall that $P[l(i)..s-1]$ must be mapped to a set of consecutive subtrees of a node in the tree rooted at $D[t]$. Let this node be $D[d]$. Let the children of $D[t]$ be, in left-to-right order, $D[t_1], D[t_2], \ldots D[t_{n_t}]$. There are two subcases to examine (as in the previous lemma):

(a) $d \neq t$. Let $D[t_k]$ be the child of $D[t]$ on the path from $D[t]$ to $D[d]$ (Figure 13). So, $forestdist(l(i)..s, l(j)..t) = treedist(s, d)$. But now $treedist(s, t_k) = treedist(s, d)$. Therefore, $forestdist(l(i)..s, l(j)..t) = treedist(s, t_k)$. The value of k ranges from 1 to n_t. Hence, $forestdist(l(i)..s, l(j)..t) = \min_{t_k} \{treedist(s, t_k) \mid 1 \leq k \leq n_t\}$.

(b) $d = t$. Then, $P[s]$ is replaced by $D[t]$ and the subtrees rooted at $D[t_1] \ldots D[t_p]$ and $D[t_q] \ldots D[t_{n_t}]$ (Figure 14). So, $forestdist(l(i)..s, l(j)..t) = forestdist(l(i)..s-1, l(t_{p+1})..t_{q-1}) = sfd(l(i)..s-1, l(j)..t_{q-1})$. The value of q ranges from 1 to n_t. Hence, $forestdist(l(i)..s, l(j)..t) = \min_{t_q} \{sfd(l(i)..s-1, l(j)..t_q) \mid 1 \leq q \leq n_t\}$.

Note that this formula also covers the case where $P[s]$ substitutes for the whole tree rooted at $D[t]$, since in that case, $forestdist(l(i)..s, l(j)..t) = forestdist(l(i)..s-1, \emptyset) \geq sfd(l(i)..s-1, l(j)..t_k)$ for any k between 1 and n_t.

Note also that when $d = t$, the expression $forestdist(l(i)..s-1, l(j)..t-1) + \gamma(P[s] \rightarrow D[t]) \geq \min_{t_q}\{sfd(l(i)..s-1, l(j)..t_q) \mid 1 \leq q \leq n_t\}$. Thus, adding this additional expression to the resulting formula does not change the value.

The proof of the lemma is complete. □

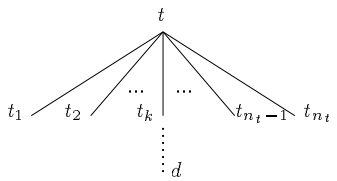


Fig. 11. Case 3(a) of Lemma 4.4.

As mentioned in the previous section, in order to compute umbrella_tree_to_tree_dist(s,t) we need an auxiliary distance measure $sfd()$. (Recall that this subroutine calculates the distance between trees $P[l(i)..s]$ and $D[l(j)..t]$ where $P[s] = \wedge$.) Here is why.

Suppose that, in the computation of umbrella_tree_to_tree_dist(s,t), $P[s]$ maps to $D[t]$ and its leftmost subtree. Thus, we need to calculate the distance from $P[l(i)..s-1]$ to $D[l(t_2)..t-1]$, where $D[t_2]$ is the root of $D[t]$'s second-to-leftmost subtree.[8] (In general, we may need to calculate the distance from $P[l(i)..s-1]$ to $D[l(t_i)..t-1]$, where $D[t_i]$ is the root of $D[t]$'s ith-to-leftmost subtree.) To find the best substitution for $P[s]$, a naive way would require calculating all such distance values and then taking the minimum. This motivates us to introduce the suffix forest distance measure.

Let F_P and F_D be the forests in the pattern P and the data tree D, respectively. A subtree of F_D is *complete* if it is not a subtree of any other tree in F_D. We define the suffix forest distance between F_P and F_D, denoted $sfd(F_P, F_D)$, as the distance between F_P and F'_D, where F'_D is a subforest of F_D with some consecutive complete subtrees removed from the left all having the same parent, i.e.,

$$sfd(F_P, F_D) = \min_{F'_D} forestdist(F_P, F'_D)$$

Figure 12 illustrates this definition. Intuitively, the removed subtrees will be used to replace the umbrella-VLDC.

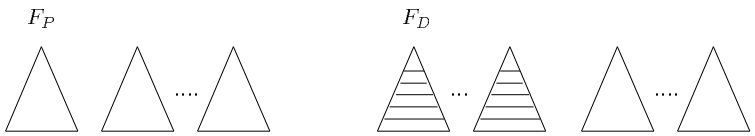


Fig. 12. The suffix distance between forests F_P and F_D. The removed subtrees are shaded. Note that these removed subtrees must (i) be complete, (ii) be consecutive, (iii) be leftmost, and (iv) have the same parent. The unshaded subtrees of the data tree need not have the same parent.

With the suffix forest distance, we are able to show the following.

Lemma 4.5. (Computation of umbrella_tree_to_tree_dist(s,t)) *Suppose $s \in des(i)$ and $t \in des(j)$. Suppose both $P[l(i)..s]$ and $D[l(j)..t]$ are trees (i.e., $l(s) = l(i)$ and $l(t) = l(j)$). If $P[s] = \wedge$, then*

[8] Note that $forestdist(l(i)..s-1, l(j)..t-1)$ does not offer such a distance value, because in calculating $forestdist(l(i)..s-1, l(j)..t-1)$, we always count the forests $P[l(i)..s-1]$ and $D[l(j)..t-1]$ as starting with their leftmost leaves (i.e., $P[l(i)]$ and $D[l(j)]$, respectively).

4.1 Algorithms for the Subroutines

In the following lemma, we show the computation of path_tree_to_tree_dist(s,t) and prove its correctness.[6]

Lemma 4.4. (Computation of path_tree_to_tree_dist(s,t)) *Suppose $s \in des(i)$ and $t \in des(j)$. Suppose both $P[l(i)..s]$ and $D[l(j)..t]$ are trees (i.e., $l(s) = l(i)$ and $l(t) = l(j)$). If $P[s] = |$, then*

$$forestdist(l(i)..s, l(j)..t) = \mathbf{min} \begin{cases} forestdist(l(i)..s-1, l(j)..t) + \gamma(P[s] \rightarrow \lambda), \\ forestdist(l(i)..s, l(j)..t-1) + \gamma(\lambda \rightarrow D[t]), \\ forestdist(l(i)..s-1, l(j)..t-1) + \gamma(P[s] \rightarrow D[t]), \\ forestdist(\emptyset, D[l(j)..t-1]) + \min_{t_k}\{treedist(s,t_k) - \\ treedist(\emptyset,t_k) | 1 \le k \le n_t\} \end{cases}$$

where $D[t_k]$, $1 \le k \le n_t$, are children of $D[t]$. (If $D[t]$ is a leaf, i.e., $t = l(j)$, then only the first three expressions are present.)

Proof. We try to find a best substitution for |'s and $\wedge$'s in the tree $P[l(i)..s]$,[7] and then consider a minimum-cost mapping M between $P[l(i)..s]$ and $D[l(j)..t]$. There are three cases to examine: in the best substitution, (1) $P[s]$ is replaced by an empty tree, (2) $P[s]$ is replaced by a nonempty tree and $D[t]$ is not touched by a line in M, and (3) $P[s]$ is replaced by a nonempty tree and $D[t]$ is touched by a line in M.

Cases 1 and 2 are similar to cases 2(a) and 2(b) in Lemma 4.2, and hence we get the same formulae derived there. For case 3, if $t = l(j)$, then $P[s]$ must be replaced by $D[t]$. So, $forestdist(l(i)..s, l(j)..t) = forestdist(l(i)..s-1, l(j)..t-1) + \gamma(P[s] \rightarrow D[t])$. The distance is the minimum of the three corresponding costs. This completes the proof for $t = l(j)$.

If $t \neq l(j)$, then $P[s]$ must be replaced by a path of the tree rooted at $D[t]$. Let the path end at node $D[d]$. Let the children of $D[t]$ be, in left-to-right order, $D[t_1], D[t_2], \ldots, D[t_{n_t}]$. There are two subcases to be considered:

(a) $d \neq t$. Let $D[t_k]$ be the child of $D[t]$ on the path from $D[t]$ to $D[d]$ (see Figure 11). Thus, $forestdist(l(i)..s, l(j)..t) = treedist(s,t_k) + treedist(\emptyset,t_1) + \ldots + treedist(\emptyset,t_{k-1}) + treedist(\emptyset,t_{k+1}) + \ldots + treedist(\emptyset,t_{n_t})$. By Lemma 4.1, $forestdist(\emptyset, D[l(j)..t-1]) = \sum_{i=1}^{n_t} treedist(\emptyset,t_i)$. Hence, we can rewrite the expression on the right hand side of the formula as $forestdist(\emptyset, D[l(j)..t-1]) + treedist(s,t_k) - treedist(\emptyset,t_k)$. The value of k ranges from 1 to n_t. Therefore, the distance should be the minimum of these corresponding costs, i.e., $forestdist(l(i)..s, l(j)..t) = forestdist(\emptyset, D[l(j)..t-1]) + \min_{t_k}\{treedist(s,t_k) - treedist(\emptyset,t_k) | 1 \le k \le n_t\}$.

(b) $d = t$. Then $P[s]$ is replaced by $D[t]$. So, $forestdist(l(i)..s, l(j)..t) = forestdist(l(i)..s-1, l(j)..t-1) + \gamma(P[s] \rightarrow D[t])$.

The above cases exhaust all possible mappings yielding $forestdist(l(i)..s, l(j)..t)$, and hence we take the minimum of all the corresponding costs.

The proof of the lemma is complete. □

[6] The formulae derived for the subroutines discussed in this subsection are used to replace directly their subroutine titles in the procedure **treedist**(i,j) of Figure 10.

[7] Note that even though $P[s] = |$, the other part of the tree may contain $\wedge$'s.

Procedure treedist(i, j)

```
1.  forestdist(∅,∅) := 0;        /* Lemma 4.1 */
2.  for s := l(i) to i do
3.      forestdist(P[l(i)..s],∅) := forestdist(P[l(i)..s − 1],∅) + γ(P[s] → λ);
4.  for t := l(j) to j do
5.      begin
6.          forestdist(∅, D[l(j)..t]) := forestdist(∅, D[l(j)..t − 1]) + γ(λ → D[t]);
7.          if l(t) = l(j) then
8.              treedist(∅, t) := forestdist(∅, D[l(j)..t])
9.      end;
10. initialize the suffix forest distances;
11. for s := l(i) to i do
12.     for t := l(j) to j do
13.         begin
14.             if (l(s) ≠ l(i) or l(t) ≠ l(j)) then        /* Lemma 4.2 */
15.                 forestdist(l(i)..s, l(j)..t) := min {
                        forestdist(l(i)..s − 1, l(j)..t) + γ(P[s] → λ),
                        forestdist(l(i)..s, l(j)..t − 1) + γ(λ → D[t]),
                        forestdist(l(i)..l(s) − 1, l(j)..l(t) − 1) + treedist(s, t)
                        }
16.             else begin        /* l(s) = l(i) and l(t) = l(j) */
17.                 if (P[s] ≠ | and P[s] ≠ ∧) then        /* Lemma 4.3 */
18.                     forestdist(l(i)..s, l(j)..t) := min {
                            forestdist(l(i)..s − 1, l(j)..t) + γ(P[s] → λ),
                            forestdist(l(i)..s, l(j)..t − 1) + γ(λ → D[t]),
                            forestdist(l(i)..s − 1, l(j)..t − 1) + γ(P[s] → D[t])
                            }
19.                 else if (P[s] = |) then
20.                     compute path_tree_to_tree_dist(s, t)
                        (i.e., compute the forestdist(l(i)..s, l(j)..t) where P[s] = |);
21.                 else if (P[s] = ∧) then
22.                     compute umbrella_tree_to_tree_dist(s, t)
                        (i.e., compute the forestdist(l(i)..s, l(j)..t) where P[s] = ∧);
23.                 treedist(s, t) := forestdist(l(i)..s, l(j)..t);        /* forestdist yields treedist */
24.             end (else);
                /* calculate the suffix forest distance */
25.             if l(t) = l(j) then
26.                 compute suffix_distance_for_tree(t)
27.             else
28.                 compute suffix_distance_for_forest(l(j)..t)
                /* end of calculating the suffix forest distance */
29.         end (for);
```

Fig. 10. Algorithm for the procedure **treedist**(i, j).

- Subroutine umbrella_tree_to_tree_dist(s,t): compute the distance between trees $P[l(i)..s]$ and $D[l(j)..t]$ where $P[s] = \wedge$.

It will become clear that in order to compute umbrella_tree_to_tree_dist(s,t), we need an auxiliary distance measure, called *suffix forest distance* (represented by $sfd()$). Informally speaking, $sfd(l(i)..s, l(j)..t)$ is the distance between two forests $P[l(i)..s]$ and $D[l(j)..t]$, allowing a subforest of $D[l(j)..t]$ to be removed from $D[l(j)..t]$. (The subtrees in the removed subforest must be consecutive, leftmost, and have the same parent.) Depending on whether $D[l(j)..t]$ is a tree, we need two additional subroutines to compute the $sfd()$.

- Subroutine suffix_distance_for_tree(t): compute the suffix forest distance between $P[l(i)..s]$ and $D[l(j)..t]$ where $D[l(j)..t]$ is a tree (i.e., $l(t) = l(j)$);
- Subroutine suffix_distance_for_forest$(l(j)..t)$: compute the suffix forest distance between $P[l(i)..s]$ and $D[l(j)..t]$ where $D[l(j)..t]$ is a forest (i.e., $l(t) \neq l(j)$).

With these subroutines, we are now able to give the algorithm for the procedure **treedist**(i,j) (see Figure 10). Two local arrays forestdist and sfd, which hold distance values $forestdist$ and sfd, are allocated when calling **treedist**(i,j) and are freed once exiting the procedure.

$forestdist(l(i)..s-1, l(j)..t)$. Since $\gamma(| \rightarrow \lambda) = 0$ and $\gamma(\wedge \rightarrow \lambda) = 0$, the expression on the right hand side can be rewritten as $forestdist(l(i)..s-1, l(j)..t) + \gamma(P[s] \rightarrow \lambda)$.

(b) In the best substitution, $P[s]$ is replaced by a nonempty tree and $D[t]$ is not touched by a line in M. Then, $forestdist(l(i)..s, l(j)..t) = forestdist(l(i)..s, l(j)..t-1) + \gamma(\lambda \rightarrow D[t])$.

(c) In the best substitution, $P[s]$ is replaced by a nonempty tree and $D[t]$ is touched by a line in M. Let the root of the tree substituting for $P[s]$ be N. So, N must be mapped to $D[t]$ by mapping condition arguments similar to case 1(c). This means that the subtree rooted at N must be mapped to the subtree rooted at $D[t]$. So, $forestdist(l(i)..s, l(j)..t) = forestdist(l(i)..l(s)-1, l(j)..l(t)-1) + treedist(s,t)$.[5] (See Figure 9.)

Thus, we end up with the the same formula for both cases. □

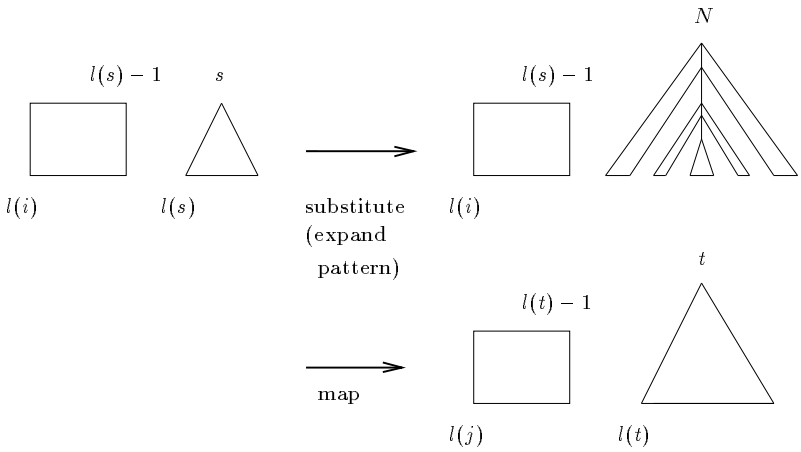


Fig. 9. Case 2(c) of Lemma 4.2.

Lemma 4.3. *Suppose $s \in des(i)$ and $t \in des(j)$. Suppose both $P[l(i)..s]$ and $D[l(j)..t]$ are trees (i.e., $l(s) = l(i)$ and $l(t) = l(j)$). If ($P[s] \neq |$ and $P[s] \neq \wedge$), then*

$$forestdist(l(i)..s, l(j)..t) = \mathbf{min} \begin{cases} forestdist(l(i)..s-1, l(j)..t) + \gamma(P[s] \rightarrow \lambda), \\ forestdist(l(i)..s, l(j)..t-1) + \gamma(\lambda \rightarrow D[t]), \\ forestdist(l(i)..s-1, l(j)..t-1) + \gamma(P[s] \rightarrow D[t]) \end{cases}$$

Proof. Similar to the proof for case 1 in Lemma 4.2. □

Two notes follow. First, the formulae obtained so far are somewhat reminiscent of those in [25, Lemma 4, Lemma 5], though the arguments establishing them are different. Second, we have purposely left out the cases where both $P[l(i)..s]$ and $D[l(j)..t]$ are trees and $P[s]$ is either | or $\wedge$. The two cases are quite involved; we defer their details to later. For now, let us assume the following two subroutines are available.

- Subroutine path_tree_to_tree_dist(s,t): compute the distance between trees $P[l(i)..s]$ and $D[l(j)..t]$ where $P[s] = |$;

[5] It is worth noting that this case also includes the situation where $P[s]$ is a leaf and labeled by a VLDC symbol.

Lemma 4.2. *Suppose $s \in des(i)$ and $t \in des(j)$. If ($l(s) \neq l(i)$ or $l(t) \neq l(j)$), then*

$$forestdist(l(i)..s, l(j)..t) = \mathbf{min} \begin{cases} forestdist(l(i)..s-1, l(j)..t) + \gamma(P[s] \rightarrow \lambda), \\ forestdist(l(i)..s, l(j)..t-1) + \gamma(\lambda \rightarrow D[t]), \\ forestdist(l(i)..l(s)-1, l(j)..l(t)-1) + treedist(s,t) \end{cases}$$

Proof. We first try to find a best substitution for |'s and $\wedge$'s in $P[l(i)..s]$, and then consider a minimum-cost mapping M between $P[l(i)..s]$ and $D[l(j)..t]$. There are two cases to examine:

Case 1: $P[s] \neq |$ and $P[s] \neq \wedge$. We can extend M to $P[s]$ and $D[t]$ in three ways.

(a) $P[s]$ is not touched by a line in M. Then, $forestdist(l(i)..s, l(j)..t) = forestdist(l(i)..s-1, l(j)..t) + \gamma(P[s] \rightarrow \lambda)$.

(b) $D[t]$ is not touched by a line in M. Then, $forestdist(l(i)..s, l(j)..t) = forestdist(l(i)..s, l(j)..t-1) + \gamma(\lambda \rightarrow D[t])$.

(c) $P[s]$ and $D[t]$ are both touched by lines in M (see Figure 8). By the ancestor and sibling conditions on mappings, (s,t) must be in M. By the ancestor condition on mappings, any node in the subtree rooted at $P[s]$ can be touched only by a node in the subtree rooted at $D[t]$. Hence, $forestdist(l(i)..s, l(j)..t) = forestdist(l(i)..l(s)-1, l(j)..l(t)-1) + forestdist(l(s)..s-1, l(t)..t-1) + \gamma(P[s] \rightarrow D[t])$.

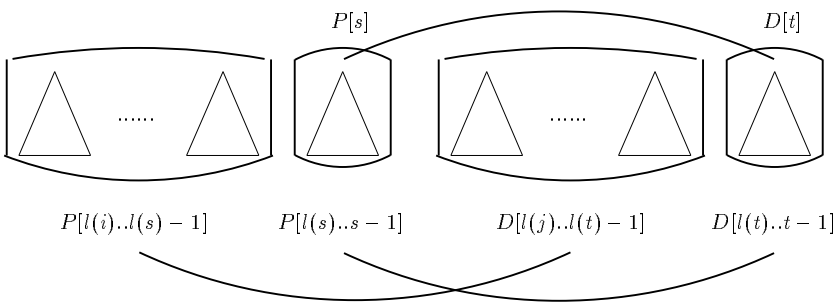


Fig. 8. Case 1(c) of Lemma 4.2.

Since these three cases exhaust all possible mappings yielding $forestdist(l(i)..s, l(j)..t)$, we take the minimum of the costs, i.e.,

$$forestdist(l(i)..s, l(j)..t) = \mathbf{min} \begin{cases} forestdist(l(i)..s-1, l(j)..t) + \gamma(P[s] \rightarrow \lambda), \\ forestdist(l(i)..s, l(j)..t-1) + \gamma(\lambda \rightarrow D[t]), \\ forestdist(l(i)..l(s)-1, l(j)..l(t)-1) \\ + forestdist(l(s)..s-1, l(t)..t-1) \\ + \gamma(P[s] \rightarrow D[t]) \end{cases}$$

Note that $forestdist(l(i)..s, l(j)..t) \leq forestdist(l(i)..l(s)-1, l(j)..l(t)-1) + treedist(s,t)$, since the distance is the cost of a minimum cost mapping, and the latter expression stands for a particular (and therefore possibly suboptimal) mapping from $P[l(i)..s]$ to $D[l(j)..t]$. For the same reason, $treedist(s,t) \leq forestdist(l(s)..s-1, l(t)..t-1) + \gamma(P[s] \rightarrow D[t])$. Hence, we can rewrite the third expression above to give the formula asserted by the lemma.

Case 2: $P[s] = |$ or $P[s] = \wedge$. The distance is the minimum of the following three subcases:

(a) In the best substitution, $P[s]$ is replaced by an empty tree. So, $forestdist(l(i)..s, l(j)..t) =$

(1) Immediately before invoking the procedure **treedist**(i,j), all distances $treedist(s,t)$, where $l(i) \leq s \leq i$ and $l(j) \leq t \leq j$ and either $l(s) \neq l(i)$ or $l(t) \neq l(j)$, are available. In other words, $treedist(s,t)$ is available if $P[s]$ is in the subtree rooted at $P[i]$ but not in the path from $P[l(i)]$ to $P[i]$ or $D[t]$ is in the subtree rooted at $D[j]$ but not in the path from $D[l(j)]$ to $D[j]$.

(2) Immediately after the execution of the procedure **treedist**(i,j), all distances $treedist(s,t)$, where $l(i) \leq s \leq i$ and $l(j) \leq t \leq j$, are available.

We first show that if (1) is true then (2) is true. From the basic assumptions, we know that the available distances are precisely the required ones. We compute each $treedist(s,t)$, where $l(s) = l(i)$ and $l(t) = l(j)$ in the procedure **treedist**(i,j) and add it to the global array. So, (2) holds.

Let us show that (1) always holds. Suppose $l(s) \neq l(i)$. Let $P[s']$ be the lowest ancestor of $P[s]$ such that $s' \in LR_keyroots(P)$. Since $l(s') = l(s) \neq l(i)$, $s' \neq i$. Since $i \in LR_keyroots(P)$, $s' \leq i$. So, $s' < i$. Let $D[t']$ be the lowest ancestor of $D[t]$ such that $t' \in LR_keyroots(D)$. Since $j \in LR_keyroots(D)$, $t' \leq j$. This means that the procedure **treedist**(s',t') will have already been executed before invoking the procedure **treedist**(i,j), since in the main loop $LR_keyroots(P)$ and $LR_keyroots(D)$ are in increasing order. The result follows immediately by noting that $treedist(s,t)$ is available after the execution of the procedure **treedist**(s',t'). □

4 Algorithm for treedist(i,j)

We use dynamic programming to compute **treedist**(i,j). The procedure considers forest-to-forest distances between the pattern P and the data tree D.

We begin with three lemmas and then give our algorithm. Note that, in the algorithm, both deleting and relabeling nodes with VLDC symbols cost zero (i.e., $\gamma(P[s] \rightarrow \lambda) = 0$ and $\gamma(P[s] \rightarrow D[t]) = 0$ when $P[s]$ is either | or $\wedge$).

Lemma 4.1. *Suppose $s \in des(i)$ and $t \in des(j)$. Then*
(i) $forestdist(\emptyset,\emptyset) = 0$;
(ii) $forestdist(P[l(i)..s],\emptyset) = forestdist(P[l(i)..s-1],\emptyset) + \gamma(P[s] \rightarrow \lambda)$;
(iii) $forestdist(\emptyset, D[l(j)..t]) = forestdist(\emptyset, D[l(j)..t-1]) + \gamma(\lambda \rightarrow D[t])$.

Proof. (i) requires no edit operation. In (ii) and (iii), the distances correspond to the cost of deleting or inserting the nodes in $P[l(i)..s]$ and $D[l(j)..t]$, respectively. Note that for (ii), when $P[s]$ is a VLDC symbol, $forestdist(P[l(i)..s],\emptyset) = forestdist(P[l(i)..s-1],\emptyset)$. In that case, since both $\gamma(| \rightarrow \lambda) = 0$ and $\gamma(\wedge \rightarrow \lambda) = 0$, we can add the term $\gamma(P[s] \rightarrow \lambda)$ to the right hand side of the above formula without changing the value. □

Output: $treedist(i,j)$ (or $treedist_cut(i,j)$ for the "with cut" case), where $1 \leq i \leq |P|$ and $1 \leq j \leq |D|$.

Preprocessing:

-- Compute $l()$, $LR_keyroots(P)$ and $LR_keyroots(D)$;

Main loop:
for $i' := 1$ **to** $|LR_keyroots(P)|$ **do**
    for $j' := 1$ **to** $|LR_keyroots(D)|$ **do**
        begin
            $i :=$ LR_keyrootsP[i'];
            $j :=$ LR_keyrootsD[j'];
            Invoke the procedure **treedist**(i,j) (or **treedist_cut**(i,j) for the "with cut" case)
        end;

We defer the details of the procedures **treedist**(i,j) and **treedist_cut**(i,j) to the next two sections. For now, assume that they somehow store the distance values they compute in the global array treedist (or treedist_cut).

To show the correctness of the framework, we need the following:

Definitions: Suppose that $s \in des(i)$ and $t \in des(j)$. If an intermediate result of the execution of the procedure **treedist**(i,j) is the value of $treedist(s,t)$, then we say that **treedist**(i,j) *yields* $treedist(s,t)$. If the execution of the procedure **treedist**(i,j) does not yield $treedist(s,t)$, but requires its value, then we say that **treedist**(i,j) *requires* $treedist(s,t)$. (The definitions are analogous in the "with cut" case.)

In essence, the above definitions show the concrete order in which our main algorithms (to be discussed later) fill their arrays. Note that one might fill the arrays row by row or column by column, though that would increase the time complexity of the algorithms. The definitions also show what computation can be avoided.

Basic Assumptions: Suppose $s \in des(i)$ and $t \in des(j)$. If either $l(s) \neq l(i)$ or $l(t) \neq l(j)$, then the procedure **treedist**(i,j) requires $treedist(s,t)$; otherwise, the procedure **treedist**(i,j) yields $treedist(s,t)$. (The assumption is analogous for the "with cut" case.)

The assumptions will be established in Section 4 for the "without cut" case and Section 5 for the "with cut" case. With the basic assumptions, we are able to prove the following:

Theorem 3.1. *The algorithmic framework is correct. That is, if the procedure* **treedist**(i,j) *(or* **treedist_cut**(i,j)*) is invoked, then at the moment its execution begins, its required subtree-to-subtree distances are available.*

Proof. We prove the "without cut" case only. The "with cut" case can be proved similarly. We will show that for any pair (i,j) such that $i \in LR_keyroots(P)$ and $j \in LR_keyroots(D)$, the following invariants hold.

numbers for descendants (ancestors, respectively) of $T[i]$ is $des(i)$ ($anc(i)$, respectively). The number of children of node $D[i]$ is $deg(i)$.

In the postorder numbering of the nodes in a tree T, $T[1..i]$ will be forests as in Figure 7. (The edges are those in the subgraph of the tree induced by the vertices.) In general, we use $T[i..j]$ to represent the ordered subforest of T induced by the nodes numbered i to j inclusive. If $i > j$, then $T[i..j] = \emptyset$. $T[l(i)..i]$ will be referred to as $tree(i)$. The number of nodes in $tree(i)$ is $Size(i)$.

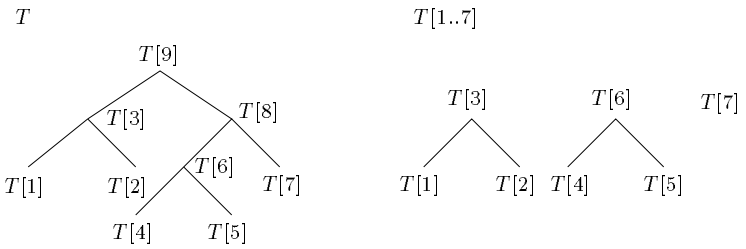


Fig. 7.

As in [25], our algorithms consider, in their intermediate steps, the distance between two ordered forests. (The definition of mapping for ordered forests is the same as that for trees.) We denote the distance between $P[i'..i]$ and $D[j'..j]$ as $forestdist(P[i'..i], D[j'..j])$ or $forestdist(i'..i, j'..j)$ if the context is clear. The distance between the subtree rooted at $T[i]$ and the subtree rooted at $T[j]$ is sometimes denoted $treedist(i, j)$. The definitions are analogous in the "with cut" case.

3 An Algorithmic Framework for Tree Matching

In computing the distance between the pattern and data tree, we consider only certain nodes in the trees. Define the set $LR_keyroots$ of tree T as follows:

$$LR_keyroots(T) = \{k \mid \text{there exists no } k' > k \text{ such that } l(k) = l(k')\}.$$

Intuitively, if $k \in LR_keyroots(T)$ then either $T[k]$ is the root of T or $l(k) \neq l(par(k))$, i.e., $T[k]$ has a left sibling. (For example, the $LR_keyroots$ set for the tree in Figure 7 is $\{2, 5, 7, 8, 9\}$.)

It is easy to see that there is a linear time algorithm to compute the function $l()$ and the sets $LR_keyroots$ for the pattern P and data tree D.[4] We assume that the results are in arrays l, LR_keyrootsP and LR_keyrootsD, respectively. Furthermore, the elements in arrays LR_keyroots are in increasing order.

We allocate a global array treedist (or treedist_cut for the "with cut" case) consisting of $|P| \times |D|$ cells. The cell (i, j) will hold $treedist(i, j)$ (or $treedist_cut(i, j)$).

The Main Framework:

Input: Pattern tree P and data tree D where P may contain both |'s and $\wedge$'s.

[4]Note also that, given the postorder numbers of nodes and the function $l()$ for tree T, one can construct in linear time a data structure which helps enumerate, for a node $T[i]$, the postorder numbers of all its children in $O(deg(i))$ time.

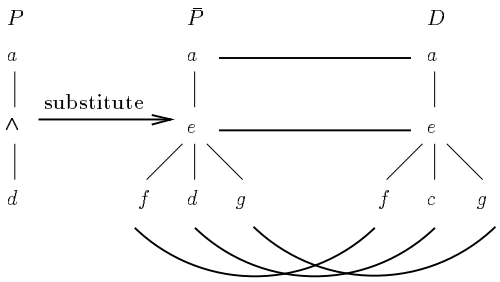


Fig. 5. An example VLDC-substitution (the node labeled $\wedge$ substitutes for nodes labeled e, f, g). Also shown is a mapping from the resulting pattern tree to the data tree. Note that the mapping must map the substituting nodes to themselves.

2.4 Cut Operations

One generalization of approximate string matching is to allow the pattern tree to match only a part of the data tree, i.e., subtrees can be freely cut from the data tree (cf. Section 1).

Formally, the operation *cutting at node* $D[i]$ means removing the subtree rooted at $D[i]$. (See Figure 6.) Let S be a set of nodes. We define S to be a set of *consistent subtree cuts* if

1. $D[i] \in S$ implies that $1 \leq i \leq |D|$;
2. $D[i], D[j] \in S$ implies that neither is an ancestor of the other.

We use $Cut(D, S)$ to represent the tree D with subtree removals at all nodes in S. Let $Subtrees(D)$ be the set of all possible sets of consistent subtree cuts. The term "matching with cut" (cf. Section 1) is defined as calculating

$$treedist_cut(P, D) = \min_{S \in Subtrees(D)} \{treedist(P, Cut(D, S))\}.$$

Our goal is an algorithmic framework that can handle tree matching with and without cut, where the pattern tree P may contain both path-VLDC's and umbrella-VLDC's.

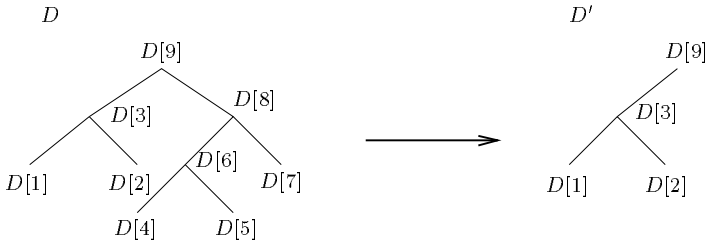


Fig. 6. Cutting at node $D[8]$.

2.5 Notation

We use $l(i)$ to represent the postorder number of the leftmost leaf descendant of the subtree rooted at $T[i]$. When $T[i]$ is a leaf, $l(i) = i$. The postorder number for the parent of $T[i]$ is $par(i)$; the set of postorder

We use a postorder numbering of the nodes in the trees. Let $T[i]$ represent the node of T whose position in the postorder for nodes of T is i. When there is no confusion, we also use $T[i]$ to represent the label of node $T[i]$. Formally, a mapping from T to T' is a triple (M, T, T') (or simply M if the context is clear), where M is any set of pairs of integers (i, j) satisfying:

(**Map Conditions**)

1. $1 \leq i \leq |T|$, $1 \leq j \leq |T'|$;
2. For any pair of (i_1, j_1) and (i_2, j_2) in M,
   (a) $i_1 = i_2$ if and only if $j_1 = j_2$ (one-to-one);
   (b) $T[i_1]$ is to the left of $T[i_2]$ if and only if $T'[j_1]$ is to the left of $T'[j_2]$ (sibling order preserved);
   (c) $T[i_1]$ is an ancestor of $T[i_2]$ if and only if $T'[j_1]$ is an ancestor of $T'[j_2]$ (ancestor order preserved).

The cost of M, denoted $\gamma(M)$, is the cost of deleting nodes of T not touched by a mapping line plus the cost of inserting nodes of T' not touched by a mapping line plus the cost of relabeling nodes in those pairs related by mapping lines with different labels.

The following lemma, which was proved in [25], establishes the relationship between the best mapping and the best sequence of edit operations. Intuitively, the lemma says that the sequence can be formed from the deletes induced by the mapping followed by the relabelings and inserts.

Lemma 2.1. *Given S, a sequence $s_1, s_2, \ldots, s_k$ of edit operations from T to T', there exists a mapping M from T to T' such that $\gamma(M) \leq \gamma(S)$. Conversely, for any mapping M, there exists a sequence of edit operations S such that $\gamma(S) = \gamma(M)$.*

Hence, $treedist(T, T') = \min \{\gamma(M) \mid M \text{ is a mapping from } T \text{ to } T'\}$.

The equivalence between mappings and editing sequences simplifies later proofs by making the definition of distances constructive rather than operational.

2.3 Substitution of VLDC's

Thus far, we have concentrated on trees without VLDC's. Let P be a pattern tree that contains both path-VLDC's (|'s) and umbrella-VLDC's ($\wedge$'s), and let D be a data tree. (| and $\wedge$ are two special symbols not in the alphabet Σ.) A VLDC-substitution s on P replaces each node labeled | in P by a path of nodes in the data tree and replaces each node labeled $\wedge$ in P by an umbrella pattern of nodes in the data tree (cf. Section 1). We require that any mapping from the resulting (VLDC-free) pattern tree $\bar{P}$ to D map the substituting nodes to themselves (Figure 5). (Thus, no cost is induced by VLDC-substitutions.) Define the distance between P and D with respect to s, denoted $treedist(P, D, s)$, as the cost of the best mapping from $\bar{P}$ to D. Then, $treedist(P, D) = \min_{s \in \mathcal{S}} \{treedist(P, D, s)\}$ where $\mathcal{S}$ is the set of all possible VLDC-substitutions.

editing distance, or simply the *distance*, from tree T to tree T', denoted $treedist(T, T')$, is the cost of the minimum cost sequence of edit operations which transform T to T'.

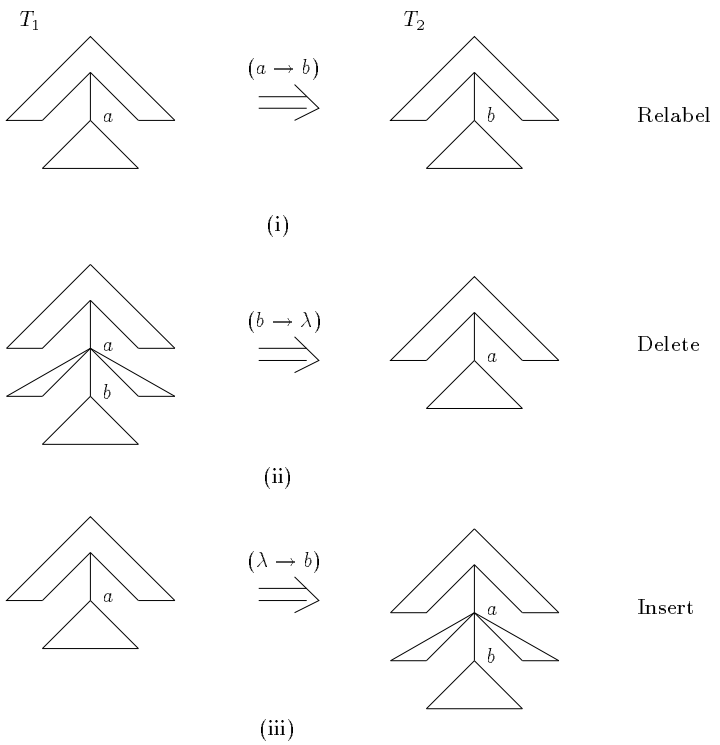


Fig. 3. (i) Relabeling: To change one node label (a) to another (b). (ii) Delete: To delete a node; all children of the deleted node (labeled b) become children of the parent (labeled a). (iii) Insert: To insert a node; a consecutive sequence of siblings among the children of the node labeled a become the children of the node labeled b.

2.2 Mappings

Often, it is convenient to describe the distance between two trees through the concept of *mappings*. A mapping is a graphical specification of what edit operations apply to each node in the two trees. For example, the mapping in Figure 4 shows a way to transform T to T'. The transformation includes deleting node labeled b in T and inserting node labeled f in T'.

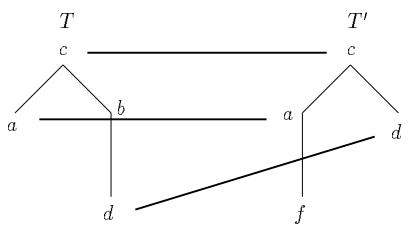
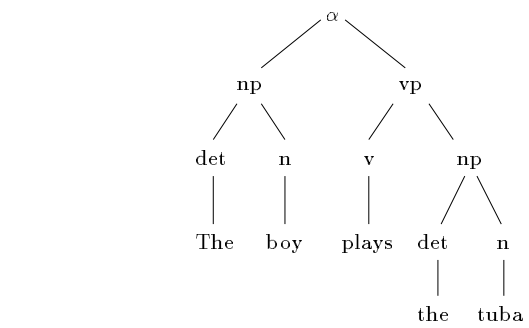
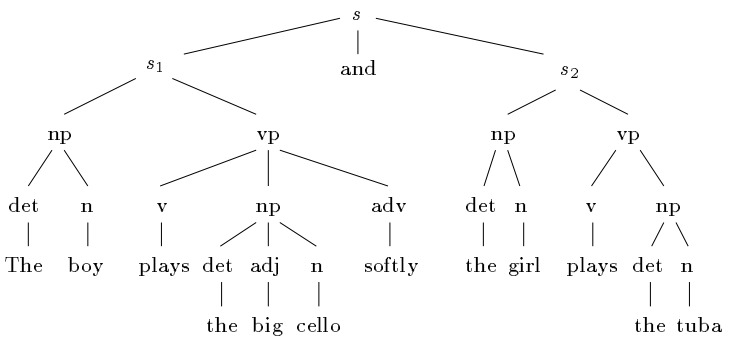


Fig. 4. A mapping from T to T'. A bold line from a node labeled u in T to a node labeled v in T' indicates that u should be changed to v if $u \neq v$, or that u remains unchanged if $u = v$. The nodes of T not touched by a bold line are to be deleted and the nodes of T' not touched are to be inserted. The mapping shows a way to transform T to T'.



(a)



(b)

Fig. 2. (a) a pattern tree; (b) a data tree.

2 Preliminaries

2.1 Edit Operations and Editing Distance between Trees

We use the editing distance to measure the difference between two trees. There are three types of edit operations: *relabeling*, *delete*, and *insert*. Relabeling node n means changing the label on n. Deleting a node n means making the children of n become the children of the parent of n and removing n. Insert is the inverse of delete. Inserting node n as the child of node n' makes n the parent of a consecutive subsequence of the current children of n'. Figure 3 illustrates these edit operations.

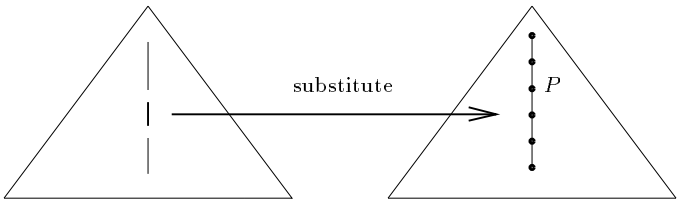
Suppose each node label is a symbol chosen from an alphabet Σ.[3] Let λ, a unique symbol not in Σ, denote the null symbol. We represent an edit operation as a pair $u \rightarrow v$, where each u is either λ or a label of a node in tree T_1 and v is either λ or a label of a node in tree T_2. We call $u \rightarrow v$ a relabeling operation if $u \neq \lambda$ and $v \neq \lambda$, a delete operation if $v = \lambda$, and an insert operation if $u = \lambda$. Let T_2 be the tree that results from the application of an edit operation $u \rightarrow v$ to tree T_1; this is written $T_1 \rightarrow T_2$ via $u \rightarrow v$.

Let S be a sequence $s_1, s_2, \ldots s_k$ of edit operations. S transforms tree T to tree T' if there is a sequence of trees $T_0, T_1, \ldots, T_k$ such that $T = T_0$, $T' = T_k$ and $T_{i-1} \rightarrow T_i$ via s_i for $1 \leq i \leq k$.

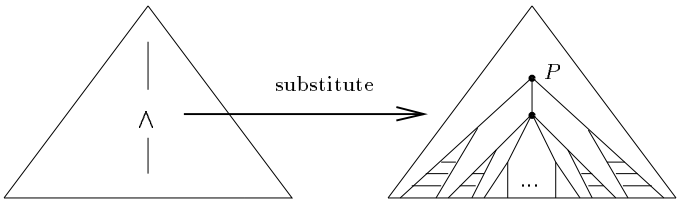
Let γ be a cost function that assigns to each edit operation $u \rightarrow v$ a nonnegative real number $\gamma(u \rightarrow v)$. By extension, the cost of a sequence is simply the sum of the costs of constituent edit operations. The

[3] A symbol could be a character, an (empty) string, etc.

in sequences. When placed at leaves in a pattern, the umbrella-VLDC's can be considered as variables that can extract information from a database, perhaps from dictionary definitions [2] or parsed text [3]. The path-VLDC (with subtrees removed being allowed) enables one to extract information from complicated sentences in the database [3,24]. Our motivation for introducing these definitions of VLDC matching comes from applications in natural language processing [2,3] and molecular biology [17]. Thus, these seemed to us to be the two most reasonable generalizations of sequence-based VLDC's.



(a)



(b)

Fig. 1. Substitution of VLDC's. (a) | substitutes for the nodes (black dots) on the path P; (b) $\wedge$ substitutes for the nodes on P and the shaded subtrees.

Example. Consider the pattern and the data tree in Figure 2. If α were |, then this would not be a close match, because the subtrees rooted at "and" and the s_2 would be counted as part of the distance.

If α were $\wedge$, then the α would match the root, the s_2, and the two left subtrees and then an approximate match would be done between the rest of the pattern and the clause concerning the girl. The cost would be 2, because "the" would have to be substituted for "The" and "girl" would have to be substituted for "boy".

If α were either symbol but subtree removals were free, then α would match the path starting at the root and ending at the s_1. The subtrees rooted at "and" and the s_2 would be removed as would the subtrees rooted at "adj" and "adv". This would give a match within distance 1 (because tuba and cello differ).

In this paper, we present algorithms to compute the distance between a pattern P and data tree D (with and without cut), where the pattern may contain both path-VLDC's and umbrella-VLDC's. Our algorithms run in $O(|P| \times |D| \times \min(depth(P), leaves(P)) \times \min(depth(D), leaves(D)))$ time, the same as for the best known algorithms for approximate tree matching without VLDC's [25]. We have implemented these algorithms in a tree toolkit [23]. Applications of the toolkit to natural language processing and molecular biology can be found in [23,24].

This problem has been studied by Sellers [16] and Ukkonen [22]. Landau and Vishkin [9] and Galil and Park [6] improved their algorithms and gave fast parallel algorithms.

Myers and Miller [12] investigated similar problems in the context of regular expression matching. Specifically, they develop algorithms to find (1) the sequence in a pattern that is most similar to the data string, and (2) substrings of a data string that strongly align with a sequence in the pattern. One interesting variation on these problems is to match a pattern that contains variable-length don't cares (VLDC's) against the data string, where the VLDC's may substitute for zero or more characters in the data string. (For example, in matching the pattern "com$*$er" with the data string "computer", $*$ would substitute for the substring "put" in the data string and the resulting distance is 0. On the other hand, matching "com$*$er" with "counter" yields the distance 1 (representing the cost of removing the "m" from "com$*$er" and having the "$*$" substitute for "unt").) Such matching is particularly useful when part of the pattern allows many variations.

1.3 Approximate Tree Matching with VLDC's

Generalizing this idea to trees gives several possible definitions of VLDC matching. The first two concern the case in which the pattern tree must match the entire data tree. The third one allows the pattern tree to match only a part of the data tree, i.e. subtrees can be freely removed from the data tree. (We believe the third to be particularly useful.)

1. The VLDC substitutes for part of a path from the root to a leaf of the data tree. We represent such a substitution by a vertical bar "|" and call it a path-VLDC (Figure 1(a)).

2. The VLDC matches part of such a path and all the subtrees emanating from the nodes of that path, except possibly at the lowest node of that path. At the lowest node, the VLDC symbol can substitute for a set of leftmost subtrees and a set of rightmost subtrees. Because the substitution part has an umbrella pattern, we call this VLDC an umbrella-VLDC and represent it by a circumflex "$\wedge$" (Figure 1(b)). Formally, let the lowest node be n and let the children of n be $c_1, \ldots, c_k$ in left-to-right order. Let i, j be such that $0 \leq i < j \leq k+1$. An umbrella can substitute for the trees rooted at $c_1, \ldots, c_i$ and $c_j, \ldots c_k$ in addition to the node n,[2] ancestors along a path starting at n, and the subtrees of those proper ancestors of n.

3. Either type of VLDC with the additional possibility that subtrees can be freely removed from the data tree for the purpose of determining a match. We denote this form of matching by the phrase "with cut".

These definitions constitute two possible generalizations of VLDC matching in sequence pattern recognition. The generalizations differ in whether the VLDC matches only a path or a path plus the subtrees emanating from the path. If we think of a sequence as a tree, each of whose interior nodes has a single child, then the two generalizations reduce to the same definition, which is consistent with VLDC matching

[2]Note that this also includes the case where the umbrella may substitute for the whole tree rooted at n.

1 Introduction

This paper is a generalization of two, heretofore independent, lines of work:

1. approximate comparison of ordered labeled trees;
2. approximate matching in strings containing variable-length don't cares.

1.1 Approximate Tree Matching

Ordered labeled trees are trees whose nodes are labeled and in which the left to right order among siblings is significant.[1] Such trees have many applications in vision, molecular biology, programming compilation and natural language processing, including the representation of images [15], patterns [11], intermediate code [1], grammar parses [3], dictionary definitions [2,13] and secondary structures of RNA [17]. They are frequently used in other disciplines as well.

Many of the above applications involve comparing ordered trees. For example, in natural language processing, computational linguists store dictionary definitions in a lexical database. The definitions are represented syntactically as trees. The syntactic head of each definition is often the genus term (superordinate) of the word being defined [4,10]. Thus, by performing syntactic analysis of the dictionary definitions (which entails matching them against a template using unification), linguists are able to extract semantic information from the definitions, thereby constructing semantic taxonomies [3,14]. (We refer the reader to [5,7,8] for algorithms dealing with exact tree matching.)

Often, researchers are interested in not only exact, but also *approximate* matches. As an example, biologists collect RNA secondary structures (trees) whose features have been analyzed. To gain information about a newly sequenced RNA, they compare the RNA's secondary structure against those in the database, searching for ones with "close" topologies. From such topological similarities, it is often possible to infer similarities in the functions of the related RNAs [17,20].

In measuring the similarity of two trees, one commonly used technique is to find a minimum-cost set of deletion, insertion and relabeling operations that converts one tree to the other [19]. In [21], Tai presented an algorithm to solve this problem in time $O(|T_1| \times |T_2| \times (depth(T_1))^2 \times (depth(T_2))^2)$, where $|T_1|$ and $|T_2|$ are the number of nodes of trees T_1 and T_2 respectively. More recently, Zhang and Shasha [25] developed a faster algorithm that computes the distance between two trees in time $O(|T_1| \times |T_2| \times \min(depth(T_1), leaves(T_1)) \times \min(depth(T_2), leaves(T_2)))$ and space $O(|T_1| \times |T_2|)$. Using suffix trees, they developed a fast parallel algorithm for the unit cost distance case [18].

1.2 Approximate String Matching with VLDC's

The second line of work that is closely related to ours is approximate string matching. Given a pattern $SPAT$ and a data string $SDATA$, the problem of approximate string matching is to compute, for each i, the distance between $SPAT[1..|SPAT|]$ and $SDATA[1..i]$ where any prefix can be removed from $SDATA[1..i]$.

[1] Throughout the paper, we shall refer to ordered trees simply as trees when no ambiguity occurs.

Approximate Tree Matching in the Presence of Variable Length Don't Cares*

Kaizhong Zhang† Dennis Shasha‡ Jason T. L. Wang§

January 18, 1993

Abstract

Ordered labeled trees are trees in which the sibling order matters. This paper presents algorithms for three problems having to do with approximate matching for such trees with variable-length don't cares (VLDC's).

In strings, a VLDC symbol in the pattern may substitute for zero or more symbols in the data string. For example, if "com∗er" is the pattern, then the "∗" would substitute for the substring "put" when matching the data string "computer". Approximate VLDC matching in strings means that after the best possible substitution, the pattern still need not be the same as the data string for a match to be allowed. For example, "com∗er" matches "counter" within distance 1 (representing the cost of removing the "m" from "com∗er" and having the "∗" substitute for "unt").

We generalize approximate VLDC string matching to three algorithms for approximate VLDC matching on trees. The time complexity of our algorithms is $O(|P| \times |D| \times \min(depth(P), leaves(P)) \times \min(depth(D), leaves(D)))$ (where $|P|$ and $|D|$ are the number of nodes respectively of the pattern P and the data tree D), the same as for the best approximate tree matching algorithm without VLDC's previously reported in [25].

*This work was supported in part by the National Science Foundation under Grants IRI-8901699 and CCR-9103953, by the Office of Naval Research under Grants N00014-90-J-1110 and N00014-91-J-1472, by the Natural Sciences and Engineering Research Council of Canada under Grant OGP0046373, by the New Jersey Institute of Technology under Grant No. 421280, and by a grant from AT&T Foundation.

†Department of Computer Science, The University of Western Ontario, London, Ontario, Canada N6A 5B7 (kzhang@csd.uwo.ca).

‡Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, New York 10012 (shasha@cs.nyu.edu).

§Department of Computer and Information Science, New Jersey Institute of Technology, University Heights, Newark, New Jersey 07102 (jason@vienna.njit.edu).