On the Maximum Common Embedded Subtree Problem for Ordered Trees

Gabriel Valiente

Department of Software, Technical University of Catalonia, E-08034 Barcelona, Spain, Fax 00 34 934 017 014, valiente@lsi.upc.es

Abstract. The maximum common embedded subtree problem, which generalizes the subtree homeomorphism problem, is reduced for ordered trees to a variant of the longest common subsequence problem, called the longest common balanced sequence problem. While the maximum common embedded subtree problem is known to be APX-hard for unordered trees, an exact solution for ordered trees can be found in polynomial time. A dynamic programming algorithm is presented that solves the longest common balanced sequence problem, and thus the maximum common embedded subtree problem, in $O(m^2n^2)$ time, where m and n are the number of edges in the trees.

Keywords. combinatorial problems, graph algorithms, trees, string algorithms, pattern matching, subtree isomorphism, subtree homeomorphism, maximum common embedded subtree, longest common balanced sequence

1 Introduction

An important generalization of tree and subtree isomorphism, known as *subtree homeomorphism*, is the problem of determining whether a tree is isomorphic to an embedded subtree of another tree, where an embedded subtree of a tree is obtained by contracting some of the edges in the tree. A further generalization of subtree homeomorphism, known as *maximum common embedded subtree*, is the problem of finding or determining the size of a largest common embedded subtree of two trees.

Both subtree homeomorphism and maximum common embedded subtree are fundamental problems on trees, which find application in various areas of computer science. Subtree homeomorphism is also known as the *tree inclusion* problem [3, 12], which is often formulated as a node deletion rather than an edge contraction problem. One of the main application areas of the subtree homeomorphism problem is information retrieval, where queries are formulated as trees and answers are homeomorphic subtrees in a semi-structured database represented as a collection of trees. The subtree homeomorphism problem can be solved on unordered trees with m and n edges in $O(m^{3/2}n)$ time [5], and the subtree isomorphism algorithm of [16] can be changed to solve the subtree homeomorphism problem in $O((m^{3/2}/\log m)/n)$ time. However, none of these algorithms can be extended to solve the maximum common embedded subtree problem, because they compute homeomorphic subtrees in bottom-up order and knowing the sizes of maximum common embedded subtrees between all children nodes of one tree and all children nodes of the other tree, is not sufficient to know the size of a maximum common embedded subtree between the trees rooted at their parent nodes. The maximum common embedded subtree problem is known to be NP-hard [12] and APX-hard, but approximable within $\log^2 n$, for unordered trees [1, 10, 18], where n is the number of nodes in the trees.

In this paper, the maximum common embedded subtree problem on ordered trees is reduced to a variant of the longest common subsequence problem, for which a dynamic programming algorithm is presented which solves the longest common balanced sequence problem, and thus the maximum common embedded subtree problem, in $O(m^2n^2)$ time, where m and n are the number of edges in the trees.

Similar reductions of pattern matching problems on ordered trees to string matching problems are known. The restricted subtree isomorphism problem (where a subtree is a whole tree rooted at a node) was reduced to a string edit problem in [6, 8, 9, 15], and the tree edit distance algorithm of [19] was reformulated as a string edit problem in [13]. The maximum common subtree problem can also be reduced to a tree edit distance problem, as done for general graphs in [2], and the tree edit problem on ordered trees with m and n edges can be solved in $O(m^{5/2}n)$ time [4]. These are just a few examples of the algorithm design technique of reducing problems on trees to problems on strings and sequences.

All trees considered in this paper will be (rooted) ordered trees, unless explicitly mentioned. The rest of the paper is organized as follows. A particular form of well-formed parenthesis strings, which will be used to describe trees, are introduced and their properties are studied in Section 2. The relationship between embedded subtrees and balanced sequences in further established in Section 3, where the longest common balanced sequence problem is introduced. A dynamic programming algorithm for solving the longest common balanced sequence problem, and thus the maximum common embedded subtree problem, is presented in Section 4. The algorithm is extended in Section 5 to deal with labeled trees. Finally, some conclusions are outlined in Section 6.

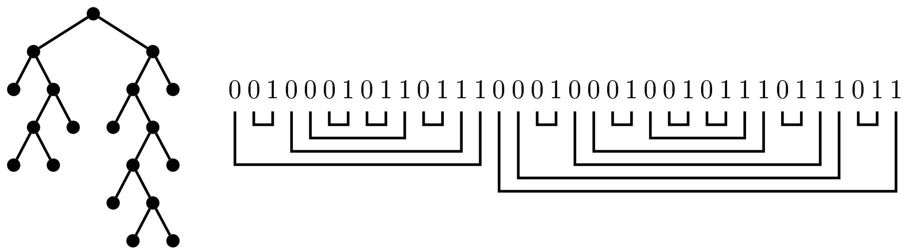
2 Balanced Sequences

Trees will be described by means of a particular form of well-formed parenthesis strings, called *balanced sequences.*

Definition 1. *Let T be a tree with m edges. The balanced sequence of T, denoted by t, is a sequence over $\{0,1\}$ of $2m$ symbols defined as follows. The balanced sequence of a leaf node is the empty sequence, and the balanced sequence associated to a nonleaf node is obtained by concatenating the balanced sequences of* *the children of the node, each of them preceded by an additional* 0 *and followed by an additional* 1*. The balanced sequence of* T *is the balanced sequence of the root of* T*.*

The length of a balanced sequence x, denoted by $|x|$, is the number of edges in the tree described by x, that is, the number of 0 characters or the number of 1 characters in x. The empty balanced sequence, which describes the tree with no nodes and no edges, is denoted by λ. Concatenation is indicated by juxtaposition.

Example 1. The sequence shown to the right is the balanced sequence of the tree shown to the left of the following picture. The balanced sequence is annotated with a nested structure, corresponding to the edges of the tree.



Fact 1 *There is a one-to-one correspondence between edges in a tree and edge annotations in the balanced sequence of the tree.*

The previous definition yields a recursive algorithm for obtaining the balanced sequence of a tree.

Fact 2 *The balanced sequence of a tree can be obtained in time linear in the size of the tree.*

Remark 1. The balanced sequence of a tree can also be obtained by performing a preorder traversal of the tree, adding a 0 each time an edge is first traversed and adding a 1 each time an edge is traversed in the opposite direction. That is, by performing a leftmost depth-first traversal of the bidirected graph underlying the tree, starting at the root, what is equivalent to finding an Euler trail through the tree [17, Sect. 5.1.2].

A balanced sequence is contained in another balanced sequence if it can be obtained from the latter by deleting edge annotations (character pairs), that is, if the tree represented by the former sequence can be embedded in the tree represented by the latter sequence.

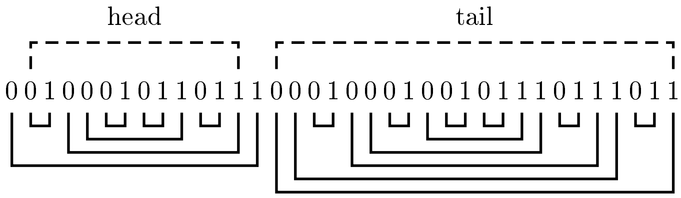
Definition 2. *A balanced sequence* s *in said to be contained in a balanced sequence* t *if there is are (not necessarily balanced) sequences* $s_1, \ldots, s_k$ *for some* $k \geqslant 1$ *such that* $s = s_1 \ldots s_k$ *and* $t = \{01\}^* \, 0^i \, s_1 \, 1^i \ldots \{01\}^* \, 0^j \, s_k \, 1^j \, \{01\}^*$ *for some* $i, \ldots, j \geqslant 0$*. A longest common balanced sequence of* s *and* t *is a balanced sequence of largest length among all balanced sequences that are contained in both* s *and* t*.*

Example 2. The balanced sequence 00100100101111 contains the following balanced sequences:

00100100101111	00010111	000111	0011	01	λ
	00100111	001011	0101		
	00101011	010011			
	01001011	010101			

Definition 3. *Let s be a nonempty balanced sequence. The head and the tail of s, denoted respectively by $head(s)$ and $tail(s)$, are the unique balanced sequences such that $s = 0\ head(s)\, 1\ tail(s)$.*

Example 3. The balanced sequence 001000101101110001000100101110111011 of the tree from Exmp. 4 can be partitioned into the head 010001011011 and the tail 00010001001011101110 11, both of which are balanced sequences.



Remark 2. The partition of the balanced sequence of a tree into a head and a tail is isomorphic to the partition of a tree into the tree rooted at the first child and the forest whose first tree is rooted at the next sibling of the first child.

Definition 4. *Let s be a balanced sequence. The decomposition of s, denoted by $decomp(s)$, is the set of balanced sequences defined as follows:*

- *$s \in decomp(s)$,*
- *for all nonempty balanced sequences $t \in decomp(s)$,*
  - *$head(t) \in decomp(s)$,*
  - *$tail(t) \in decomp(s)$,*
  - *$head(t)\, tail(t) \in decomp(s)$,*
- *no other balanced sequence belongs to $decomp(s)$.*

The decomposition of a balanced sequence is thus described by the following recurrence:

$$\begin{aligned} D[\lambda] &= \{\lambda\} \\ D[0x1y] &= \{0x1y\} \cup D[x] \cup D[y] \cup D[xy] \end{aligned}$$

Now, every sequence in the decomposition of a balanced sequence which is a suffix of another balanced sequence, belongs to the decomposition of the latter sequence.

Lemma 1. *$D[y] \subseteq D[xy]$ for all balanced sequences x and y.*

Proof. By induction on $|x|$. Let x and y be balanced sequences. If $|x| = 0$, $D[y] = D[xy]$. Otherwise, let $x = 0x'1y'$. Then,

$$\begin{aligned} D[y] &\subseteq D[y'y] && \text{(by induction hypothesis)} \\ &\subseteq D[xy] && \text{(by definition of } D\text{)} \end{aligned}$$

□

Now, the recurrence

$$\begin{aligned} R[\lambda] &= \{\lambda\} \\ R[0x1y] &= \{0x1y\} \cup R[xy] \end{aligned}$$

will also be used in the proof of the next lemma below. Let $\|X\|$ denote the cardinality of a set X, in order to distinguish it from the length $|x|$ of a balanced sequence x.

Proposition 1. *The cardinality of $R[x]$ is linear in $|x|$, for any balanced sequence x.*

Proof. By induction on $|x|$. Let x be a balanced sequence. If $|x| = 0$, $\|R[x]\| = \|\{\lambda\}\| = 1 = |x| + 1$. Otherwise, let $x = 0x'1y'$. Then, $\|R[x]\| = \|\{0x'1y'\}\| + \|R[x'y']\| = 1 + (|x'y'| + 1) = |x| + 1$. Therefore, $\|R[x]\|$ is linear in $|x|$. □

Lemma 2. $D[x] \subseteq D[xy] \cup R[x]$ *for all balanced sequences x and y.*

Proof. By induction on $|x|$. Let x and y be balanced sequences. If $|x| = 0$, $D[x] = \{\lambda\} \subseteq D[xy] \cup R[x]$. Otherwise, let $x = 0x'1y'$. Note first that

$$\begin{aligned} D[x] &= \{x\} \cup D[x'] \cup D[y'] \cup D[x'y'] && \text{(by definition of } D\text{)} \\ &= \{x\} \cup D[x'] \cup D[x'y'] && \text{(by induction hypothesis)} \end{aligned}$$

Moreover, $\{x\} \in R[x]$ by definition of R, $D[x'] \subseteq D[xy]$ by definition of D and, further,

$$\begin{aligned} D[x'y'] &\subseteq D[x'y'y] \cup R[x'y'] && \text{(by induction hypothesis)} \\ &\subseteq D[x'y'y] \cup R[x] && \text{(by definition of } R\text{)} \\ &\subseteq D[xy] \cup R[x] && \text{(by definition of } D\text{)} \end{aligned}$$

Therefore, it holds that $D[x] \subseteq D[xy] \cup R[x]$. □

The previous lemmata combine into the following result.

Corollary 1. $D[0x1y] \subseteq \{0x1y\} \cup D[xy] \cup R[x]$ *for all balanced sequences x and y.*

Proof. Let x and y be balanced sequences. Then,

$$\begin{aligned} D[0x1y] &= \{0x1y\} \cup D[x] \cup D[y] \cup D[xy] && \text{(by definition of } D\text{)} \\ &= \{0x1y\} \cup D[x] \cup D[xy] && \text{(by Lemma 1)} \\ &\subseteq \{0x1y\} \cup D[xy] \cup R[x] && \text{(by Lemma 2)} \end{aligned}$$

□

The main result in this section is an upper bound on the number of sequences in the decomposition of a balanced sequence.

Theorem 1. *The cardinality of the decomposition of a balanced sequence is quadratic in the length of the sequence.*

Proof. It will be shown by induction on $|xy|$ that $\|D[0x1y]\| \leqslant |0x1y|^2 + 1$. Let x and y be balanced sequences. If $|xy| = 0$, $\|D[0x1y]\| = \|\{\lambda, 01\}\| \leqslant |01|^2 + 1$. Otherwise, let $xy = 0x'1y'y$. Then,

$$\begin{aligned}
\|D[0x1y]\| &\leqslant \|D[0x'1y'y]\| + \|R[0x'1y']\| + 1 && \text{(by Corollary 1)}\\
&\leqslant \|D[0x'1y'y]\| + |0x'1y'| + 2 && \text{(by Proposition 1)}\\
&\leqslant |0x'1y'y|^2 + |0x'1y'| + 3 && \text{(by induction hypothesis)}\\
&\leqslant |0x'1y'y|^2 + 2\,|0x'1y'| + 2 &&\\
&= (|0x'1y'y| + 1)^2 + 1 &&\\
&= |0x1y|^2 + 1 &&
\end{aligned}$$

Therefore, $\|D[0x1y]\|$ is quadratic in $|0x1y|$. □

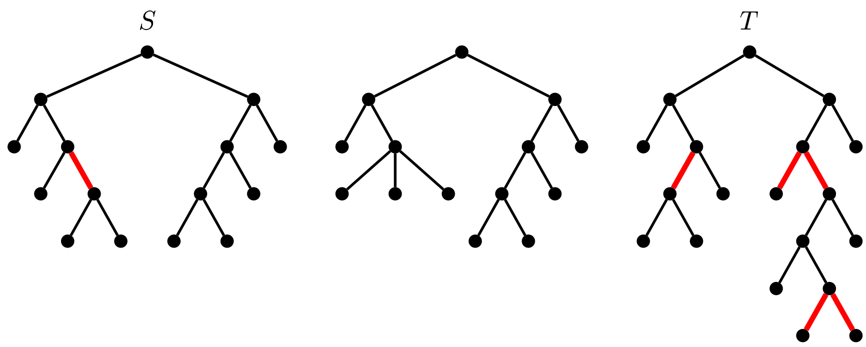
Remark 3. Note that the previous upper bound on the cardinality of the decomposition of a balanced sequence is tight, because it is achieved by an infinite number of sequences. As a matter of fact, the decomposition of a balanced sequence that describes the leftist full binary tree with m edges, for all even values of m, contains $m^2/8 + m/4 + 1$ sequences.

3 Embedded Subtrees and Balanced Sequences

Maximum common embedded subtree [1, GT48] is the problem of finding a tree of largest size that can be embedded into two given trees.

Definition 5. *Let S and T be trees. S is an embedded subtree of T if it can be obtained from T by a series of edge contractions. A common embedded subtree of S and T is a tree which is embedded in both S and T. A maximum common embedded subtree of S and T is a tree of largest size among all common embedded subtrees of S and T.*

Example 4. A maximum common embedded subtree of trees S and T can be obtained by contracting the edges highlighted in red in the following picture of the trees. The resulting tree, shown in the middle of the picture, has $14 - 1 = 18 - 5 = 13$ edges.

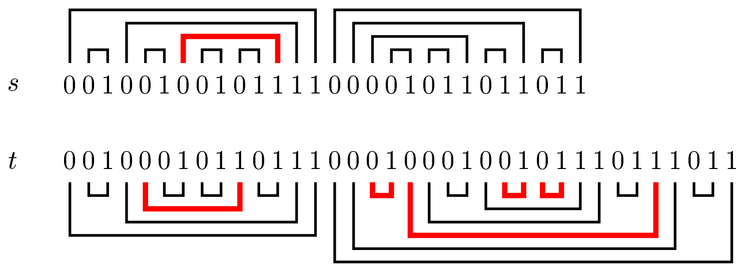


Now, the relationship between embedded subtrees and balanced sequences can be established as follows.

Theorem 2. *A longest common balanced sequence of the balanced sequences of two trees is the balanced sequence of a maximum common embedded subtree of the trees.*

Proof. Contraction of an edge in a tree corresponds to deletion of an edge annotation (one character pair) in the balanced sequence of the tree. A common embedded subtree with the largest number of edges corresponds to a longest common balanced sequence of the balanced sequences of the trees. □

Example 5. A longest common balanced sequence of the balanced sequences for the trees S and T in Exmp. 4 can be obtained by deleting the edge annotations highlighted in red, together with their adjacent characters. The resulting balanced sequence has $2(14-1) = 2(18-5) = 2 \cdot 13 = 26$ characters.



Remark 4. The longest common balanced sequence problem is related to the longest common subsequence problem for sequences with nested edge annotations, which is useful in the comparison of RNA secondary structures [11]. Although the latter problem is NP-hard [14], it consists in finding a longest common subsequence that preserves all induced edges, and the former problem is the particular case in which all characters are paired by an edge with some other character.

4 Computing Maximum Common Embedded Subtrees

The recursive decomposition of balanced sequences, studied in Section 2, into head, tail, and concatenation of head and tail, is motivated by a natural way in which the maximum common embedded subtree problem can be divided in smaller subproblems.

Lemma 3. *The size of a longest common balanced sequence* $lcs(s,t)$ *of two trees* s *and* t *is described by the following recurrence:*

$$
\begin{aligned}
&lcs(s,\lambda) = 0\\
&lcs(\lambda,t) = 0\\
&lcs(s,t) = \max \begin{cases} lcs(head(s), head(t)) + lcs(tail(s), tail(t)) + 1,\\ lcs(head(s)\, tail(s), t),\\ lcs(s, head(t)\, tail(t)) \end{cases}
\end{aligned}
$$

Proof. The only common sequence of a balanced sequence and the empty sequence is the empty sequence, which has zero length. Given two nonempty balanced sequences, if the first edge annotation of both sequences belongs to their longest common balanced sequence, then its size is one plus the sum of the sizes of the longest common balanced sequence of the heads and of the longest common balanced sequence of the tails of the two sequences. Otherwise, their longest common balanced sequence is the longest balanced sequence of one of the sequences and the result of deleting the first edge annotation in the other sequence (that is, contracting the first edge in the other tree). □

Now, in order to turn the previous recurrence into an efficient dynamic programming algorithm, a method is needed to map balanced sequences to array positions. Actually, only those sequences resulting from the decomposition of a given sequence need to be taken care of.

Recall from Theorem 1 that the cardinality of the decomposition of a balanced sequence is quadratic in the length of the sequence. Definition 4 yields a recursive algorithm for enumerating all sequences in the decomposition of the balanced sequence of a tree, which can be assigned unique numbers by exploiting the idea of [7] that a procedure for dynamically maintaining a global dictionary of unique identifiers, allows partitioning a tree into equivalence classes of restricted subtree isomorphism in expected time linear in the size of the tree. The equivalent problem of assigning unique identifiers to balanced sequences can thus be solved in expected time linear in the cardinality of the decomposition, meaning expected time quadratic in the size of the tree.

Algorithm 1 *The following algorithm assigns to each sequence* t *in the decomposition of a balanced sequence* s *a unique number* $code(t) \in \{1,\ldots,|s|^2\}$*, with the help of a dictionary* d *of integers indexed by boolean strings.*

procedure *encode* (s,d)
  set *count* to 0
  for t the sequence $head(s)$, $tail(s)$, and $head(s)\, tail(s)$ **do**
    if $t \neq \lambda$ **then**
      lookup t in dictionary d
      if not found **then**
        add 1 to *count*
        insert $\langle t, count\rangle$ in dictionary d
        set $code(t])$ to *count*

```
      encode(t, d)
    end if
  end if
 end for
end procedure
```

Lemma 4. *The algorithm for assigning unique numbers to the sequences in the decomposition of a balanced sequence is correct.*

Proof. Given a balanced sequence s, the unique number $code(t)$ for each sequence $t \in decomp(s)$ is set to the next non-assigned number for the decomposition, also updating the dictionary of assigned numbers. □

Lemma 5. *The algorithm for assigning unique numbers to the sequences in the decomposition of a balanced sequence takes expected $O(m^2)$ time, on a sequence of length m.*

Proof. For each sequence $t \in decomp(s)$, one unsuccessful dictionary lookup of t and one insertion of $\langle t, code(t) \rangle$ in the dictionary are made. With standard hashing techniques, each such operation takes expected $O(1)$ time and, by Theorem 1, the number of sequences in the decomposition of a balanced sequence of length m is $O(m^2)$. □

Now, the previous encoding is used to map balanced sequences to array positions, leading to a simple and efficient dynamic programming algorithm with memoization.

Algorithm 2 *The following dynamic programming algorithm finds the size of a longest common balanced sequence $lcs(s,t)$ of two sequences s and t, with memoization realized by storing the solution to a subproblem on sequences x and y at entry $a[code(x), code(y)]$ in an integer array a.*

```
procedure init lcs (s, t, a)
  for i from 1 to |s| do
    for j from 1 to |t| do
      set a[i, j] to ∞
    end for
  end for
end procedure
```

```
function lcs (s, t, a)
  if s = λ or t = λ then
    return 0
  else
    if a[code(s), code(t)] < ∞ then
      return a[code(s), code(t)]
    else
      set m to lcs(head(s), head(t)) + lcs(tail(s), tail(t)) + 1
      set m to max(m, lcs(head(s) tail(s), t))
      set m to max(m, lcs(s, head(t) tail(t)))
      set a[code(s), code(t)] to m
      return m
    end if
  end if
end function
```

Lemma 6. *The dynamic programming algorithm for the longest common balanced sequence problem is correct.*

Proof. The size of a longest common balanced sequence $lcs(s,t)$ of two sequences s and t is either looked up at array entry $a[code(s), code(t)]$, or computed according to Lemma 3 and stored in that array entry. □

Lemma 7. *The dynamic programming algorithm for the longest common balanced sequence problem takes $O(m^2n^2)$ time, on sequences of length m and n.*

Proof. There are, by Theorem 1, $O(m^2)$ and $O(n^2)$ sequences in the decomposition of two balanced sequences of length m and n and, for each pair of sequences, four array accesses and one array update are made, each taking $O(1)$ time. □

Corollary 2. *The dynamic programming algorithm for the maximum common embedded subtree problem takes $O(m^2n^2)$ time, on trees with m and n edges.*

Proof. The encoding of two trees with m and n edges takes, by Lemma 5, expected $O(m^2 + n^2)$ time upon which, by Lemma 7, solving the longest common balanced sequence problem takes $O(m^2n^2)$ time. □

The dynamic programming algorithm for the maximum common embedded subtree problem computes the size $lcs(s,t)$ of a largest common balanced sequence of the sequences s and t that describe two trees S and T. The sets of edge contractions in S and T that construct an embedding of the maximum common embedded subtree found into S and into T can be obtained by a straightforward extension of the algorithm, as follows.

There are three cases to consider. When computing the size $lcs(s,t)$ of a largest common balanced sequence:

1. If the largest size comes from $lcs(head(s), head(t)) + lcs(tail(s), tail(t)) + 1$, take note of the mapping of the first edge annotation in s to the first edge annotation in t.

2. If the largest size comes from $lcs(head(s)\, tail(s), t)$, take note of the contraction of the first edge annotation in s.
3. If the largest size comes from $lcs(s, head(t)\, tail(t))$, take note of the contraction of first edge annotation in t.

Note that there is redundant information in the previous procedure, because it actually suffices to keep track of the mapping of edges of S to edges of T in a largest common balanced sequence of s and t. Those edges of S that are not mapped to any edge of T are contracted in the embedding of the corresponding maximum common embedded subtree into S, and those edges of T which no edge of S is mapped to are contracted in the embedding into T.

5 Maximum Common Embedded Subtrees in Labeled Trees

Maximum common labeled subtrees of labeled trees can be found by a straightforward extension of the dynamic programming algorithm. Assume, without loss of generality, that trees have labels on edges. Trees with labels on nodes can be dealt with by shifting node labels to the edge joining the parent with the node, for all nonroot nodes.

Now, for each edge annotation in the balanced sequence of a tree, the edge label can be associated with the corresponding 0 character in the balanced sequence. Then, when computing the size $lcs(s, t)$ of a largest common balanced sequence of sequences s and t, or when mapping the first edge annotation in s to the first edge annotation in t, the case $lcs(head(s), head(t)) + lcs(tail(s), tail(t)) + 1$ of the recurrence in Lemma 3 applies only if the edge labels associated to these first edge annotations are identical, or if they satisfy some predefined criteria, depending on the intended application.

6 Conclusions

The maximum common embedded subtree problem, a generalization of the subtree homeomorphism problem, is studied in this paper. While the maximum common embedded subtree problem is known to be APX-hard for unordered trees, an exact solution is presented for ordered trees that is based on a reduction to a variant of the longest common subsequence problem, called the longest common balanced sequence problem. A dynamic programming algorithm is presented that solves the longest common balanced sequence problem, and thus the maximum common embedded subtree problem, in $O(m^2n^2)$ time, where m and n are the number of edges in the trees.

Acknowledgment

The author would like to acknowledge detailed comments by Rafel Casas and Antoni Lozano on a preliminary version of this paper.

References

1. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kahn, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and their Approximability Properties.* Springer-Verlag, 1999.
2. H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689–694, 1997.
3. W. Chen. More efficient algorithm for ordered tree inclusion. *Journal of Algorithms*, 26(2):370–385, 1998.
4. W. Chen. New algorithm for ordered tree-to-tree correction problem. *Journal of Algorithms*, 40(2):135–158, 2001.
5. M.-J. Chung. $O(n^{2.5})$ time algorithms for the subgraph homeomorphism problem on trees. *Journal of Algorithms*, 8(1):106–112, 1987.
6. P. Dublish. Some comments on the subtree isomorphism problem for ordered trees. *Information Processing Letters*, 36(5):273–275, 1990.
7. P. Flajolet, P. Sipala, and J.-M. Steyaert. Analytic variations on the common subexpression problem. In *Proc. 17th Int. Colloquium on Automata, Languages, and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 220–234. Springer-Verlag, 1990.
8. R. Grossi. Further comments on the subtree isomorphism for ordered trees. *Information Processing Letters*, 40(5):255–256, 1991.
9. R. Grossi. A note on the subtree isomorphism for ordered trees and related problems. *Information Processing Letters*, 39(2):81–84, 1991.
10. M. M. Halldórsson and K. Tanaka. Approximation and special cases of common subtrees and editing distance. In *Proc. 7th Ann. Int. Symp. on Algorithms and Computation*, volume 1178 of *Lecture Notes in Computer Science*, pages 75–84. Springer-Verlag, 1996.
11. T. Jiang, G.-H. Lin, B. Ma, and K. Zhang. The longest common subsequence problem for arc-annotated sequences. In *Proc. 11th Annual Symp. on Combinatorial Pattern Matching*, volume 1848 of *Lecture Notes in Computer Science*, pages 154–165. Springer-Verlag, 2000.
12. P. Kilpeläinen and H. Mannila. Ordered and unordered tree inclusion. *SIAM Journal on Computing*, 24(2):340–356, 1995.
13. P. N. Klein. Computing the edit-distance between unrooted ordered trees. In *Proc. 6th Annual European Symp. on Algorithms*, volume 1461 of *Lecture Notes in Computer Science*, pages 91–102, Berlin Heidelberg, 1998. Springer-Verlag.
14. G.-H. Lin, Z.-Z. Chen, T. Jiang, and J. Wen. The longest common subsequence problem for sequences with nested arc annotations. In *Proc. 28th Int. Colloquium on Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Science*, pages 444–455, Berlin Heidelberg, 2001. Springer-Verlag.
15. E. Mäkinen. On the subtree isomorphism problem for ordered trees. *Information Processing Letters*, 32(5):271–273, 1989.
16. R. Shamir and D. Tsur. Faster subtree isomorphism. *Journal of Algorithms*, 33(2):267–280, 1999.
17. G. Valiente. *Algorithms on Trees and Graphs.* Springer-Verlag, Berlin, 2002.
18. K. Zhang and T. Jiang. Some MAX SNP-hard results concerning unordered labeled trees. *Information Processing Letters*, 49(5):249–254, 1994.
19. K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.