Approximate Pattern Matching in Shared-Forest

M. Vilares, F.J. Ribadas, and V.M. Darriba

Computer Science Department, Campus de Elviña s/n, 15071 A Coruña, Spain
vilares@udc.es ribadas@mail2.udc.es darriba@mail2.udc.es
WWW home page: http://www.dc.fi.udc.es/~vilares

Abstract. We present a proposal intended to demonstrate the applicability of tabulation techniques to pattern recognition problems, when dealing with structures sharing some common parts. This work is motivated by the study of information retrieval for textual databases, using pattern matching as a basis for querying data.

1 Introduction

One critical aspect of an information system that determines its effectiveness is indexing, that is, the representation of concepts to get a well formed data structure for search. Once this has been done, the system must define a strategy in order to translate a query statement to the database and determine the information to be returned to the user.

Until recently, indexing was accomplished by creating a bibliographic citation in a structured file that references the original text. This approach allows a reduction in time and space bounds, although the ability to find information on a particular subject is limited by the system which creates index terms for that subject. At present, the significant reduction in cost processing has propitiated the notion of total document indexing, for which all words in a document are potential index descriptors. This reduction saves the indexer from entering index terms that are identical to words in the document, but does not facilitate the finding of relevant information for the user.

In effect, the words used in a query do not always reflect the value of the concepts being presented. It is the combination of these words and their semantic implications that contain the value of these concepts which leads us to more sophisticated index representations, such as context-free grammars [1] [2]. This information is inherent in the document and query. So, matching becomes a possible mechanism for extracting a common pattern from multiple data and we could use it to locate information of linguistic interest in natural language processing. Some related work about using syntax structures in IR can be found in Smeaton et al. [3] [4] [5].

However, the language intended to represent the document can often only be approximately defined, and therefore ambiguity arises. Since it is desirable to consider all possible parses for semantic processing, it is convenient to merge parse trees as much as possible into a single structure that allows them to share common parts. Although in the case of the query, language ambiguity could

probably be eliminated, queries could vary widely from indexes and an approximate matching strategy becomes necessary. At this point, our aim is to exploit structural sharing during the matching process in order to improve performances.

2 The Editing Distance

Given trees, T_1 and T_2, we define an *edit operation* as a pair $a \rightarrow b$, $a \in \text{labels}(T_1) \cup \{\varepsilon\}$, $b \in \text{labels}(T_2) \cup \{\varepsilon\}$, $(a,b) \neq (\varepsilon,\varepsilon)$, where ε represents the empty string. We can delete a node $(a \rightarrow \varepsilon)$, insert a node $(\varepsilon \rightarrow b)$, and change a node $(a \rightarrow b)$. Each edit operation has an associated cost, $\gamma(a \rightarrow b)$, that we extend to a sequence S of edit operations $s_1,\ s_2,\ \ldots,\ s_n$ in the form $\gamma(S) = \sum_{i=1}^{|S|}(\gamma(s_i))$. The distance between T_1 and T_2 is defined by the metric:

$$\delta(T_1,T_2) = \min\{\gamma(S),\ S \text{ editing sequence taking } T_1 \textit{ to } T_2\}$$

Given a postorder traversal, as shown in Fig. 2, to name each node i of a tree T by $T[i]$, a *mapping* from T_1 to T_2 is a triple (M,T_1,T_2), where M is a set of integer pairs (i,j) satisfying, for each $1 \leq i_1, i_2 \leq \mid T_1 \mid$ and $1 \leq j_1, j_2 \leq \mid T_2 \mid$:

$i_1 = i_2$	**iff** $j_1 = j_2$
$T_1[i_1]$ is to the left of $T_1[i_2]$	**iff** $T_2[j_1]$ is to the left of $T_2[j_2]$
$T_1[i_1]$ is an ancestor of $T_1[i_2]$	**iff** $T_2[j_1]$ is an ancestor of $T_2[j_2]$

which corresponds, in each case, to one-to-one assignation, sibling order preservation and ancestor order preservation. The cost, $\gamma(M)$, of a mapping (M,T_1,T_2) is computed from relabeling, deleting and inserting operations, as follows:

$$\gamma(M) = \sum_{(i,j)\in M} \gamma(T_1[i] \rightarrow T_2[j]) + \sum_{i\in\mathcal{D}} \gamma(T_1[i] \rightarrow \varepsilon) + \sum_{j\in\mathcal{I}} \gamma(\varepsilon \rightarrow T_2[j])$$

where $\mathcal{D}$ and $\mathcal{I}$ are, respectively, the nodes in T_1 and T_2 not touched by any line in M. Tai [6] proves, given trees T_1 and T_2, that

$$\delta(T_1,T_2) = min\{\gamma(M),\ M \text{ mapping from } T_1 \text{ to } T_2\}$$

which allows us to focus on edit sequences that are a mapping. We show, in the leftmost diagram of Fig. 1, an example of mapping between two trees. The rightmost diagram includes a sequence of edit operations not constituting a mapping.

3 The Zhang and Shasha's Algorithm

We have based our work in the Zhang and Shasha's tree pattern matching algorithm, introduced in [8] and extended with advanced matching features in [9]. A major characteristic of this algorithm is its bottom-up oriented approach. Given the *l_keyroots*(T), the set of all nodes in T which have a left sibling plus the root,

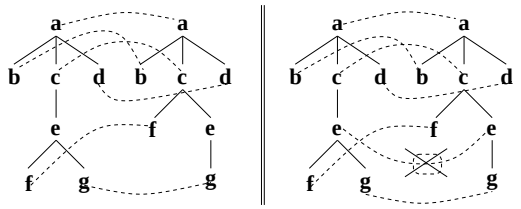

Fig. 1. An example on mappings

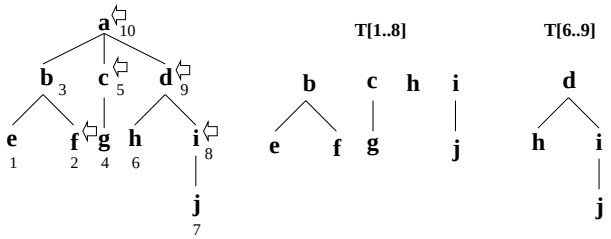

Fig. 2. The forest distance using a postorder numbering

$root(T)$, of T; the algorithm proceeds through the nodes determining mappings from all leaf l_keyroots first, then all l_keyroots at the next higher level, and so on to the root.

We introduce $l(i)$ (resp. $anc(i)$) as the leftmost leaf descendant of the subtree rooted at T_i (resp. the ancestors of T_i) in a tree T, and $T[i..j]$ as the ordered sub-forest of T induced by the nodes numbered i to j inclusive. In particular, we have $T[l(i)..i]$ which is the tree rooted at $T[i]$, as shown in Fig. 2, where the set of *l_keyroots* is indicated by arrows. We also define the *forest edition distance* as a generalization of δ, in the form

$$\text{f_d}(T_1[i_1..i_2], T_2[j_1..j_2]) = \delta(T_1[i_1..i_2], T_2[j_1..j_2])$$

that we shall denote $f_d(i_1..i_2, j_1..j_2)$ when the context is clear. Intuitively, this new concept computes the distance between two nodes, $T_1[i_2]$ and $T_2[j_2]$, in the context of their left siblings in the corresponding trees, while the corresponding tree distance, $\delta(T_1[i_2], T_2[j_2])$, is computed only from their descendants.

Formally, we compute $t_d(T_1, T_2)$ applying the formulas that follow, for nodes $i_1 \in anc(i)$ and $j_1 \in anc(j)$, as illustrated in Fig. 3, taking into account the different cases:

$$\text{f_d}(l(i_1)..i, l(j_1)..j) = \begin{cases} \min \left\{ \begin{array}{ll} \text{f_d}(l(i_1)..i-1,\ l(j_1)..j) & +\gamma(T_1[i] \to \varepsilon), \\ \text{f_d}(l(i_1)..i,\ l(j_1)..j-1) & +\gamma(\varepsilon \to T_2[j]), \\ \text{f_d}(l(i_1)..i-1,\ l(j_1)..j-1) & +\gamma(T_1[i] \to T_2[j]) \end{array} \right\} \\ \textbf{iff } l(i) = l(i_1) \textbf{ and } l(j) = l(j_1) \\ \\ \min \left\{ \begin{array}{ll} \text{f_d}(l(i_1)..i-1,\ l(j_1)..j) & +\gamma(T_1[i] \to \varepsilon), \\ \text{f_d}(l(i_1)..i,\ l(j_1)..j-1) & +\gamma(\varepsilon \to T_2[j]), \\ \text{f_d}(l(i_1)..l(i)-1,\ l(j_1)..l(j)-1) & +\text{t_d}(i,j) \end{array} \right\} \\ \textbf{otherwise} \end{cases}$$

Now, to compute the distance between T_1 and T_2, it will be sufficient to take into account that

$$\text{t_d}(T_1, T_2) = \text{f_d}(l(\text{root}(T_1))..\text{root}(T_1), l(\text{root}(T_2))..\text{root}(T_2))$$

The time complexity of this algorithm is, in the worst case:

$$\mathcal{O}(\mid T_1 \mid\mid T_2 \mid \min(\text{depth}(T_1), \text{leaves}(T_1))\min(\text{depth}(T_1), \text{leaves}(T_2)))$$

where $\mid T_1 \mid$ (resp. $\mid T_2 \mid$) is the number of nodes in the pattern tree T_1 (resp. in the data tree T_2), leaves(T_1) (resp. leaves(T_2)) is the number of leaves in T_1 (resp. in T_2), and depth(T_1) (resp. depth(T_2)) is the depth of T_1 (resp. of T_2).

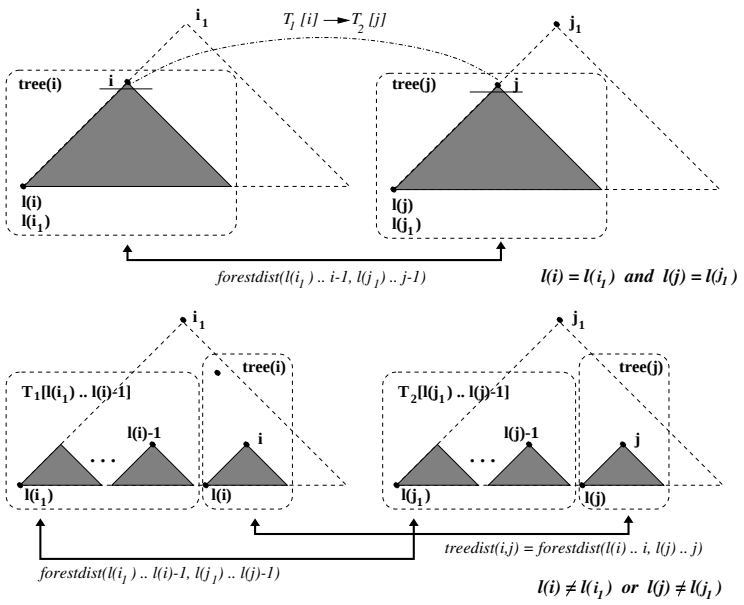

Fig. 3. The forest distance in Zhang and Shasha's algorithm

4 Relating Parsing and Approximate Tree Matching

The major question of Zhang and Shasha's algorithm [8], is the tree distance algorithm itself. However, parsing and tree-to-tree correction are topologically related and it is necessary to understand the mechanisms that cause the phenomenon of tree duplication to get the best performance.

A major factor to take into account is the syntactic representation used. We chose to work in the parsing context described for ICE [7]. Here, authors represent a parse as the chain of the context-free rules used in a leftmost reduction of the input sentence, rather than as a tree. When the sentence has distinct parses, the set of all possible parse chains is represented in finite shared form by a context-free grammar that generates that possibly infinite set.

This difference with most other parsers is only apparent, since context-free grammars can be represented by AND-OR graphs that in our case are precisely the shared-forest graph. In this graph, AND-nodes correspond to the usual parse-tree nodes, while OR-nodes correspond to ambiguities. Sharing of structures is represented by nodes accessed by more than one other node, and it may correspond to sharing of a complete subtree, but also to sharing of a part of the descendants of a given node.

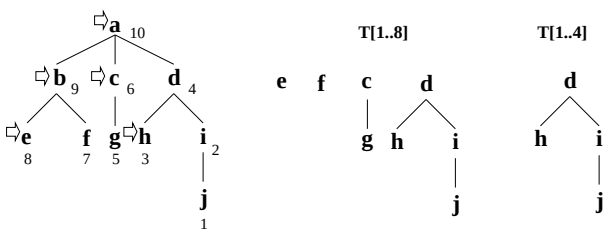

Fig. 4. The forest distance using an inverse postorder numbering

In this context, sharing of a tail of sons in a node of the resulting forest is possible. More exactly, bottom-up parsing may share only the rightmost constituents, while top-down parsing may only share the leftmost ones. This relates to the type of search used to build the forest. Breadth first search results in bottom-up constructions and depth first search results in top-down ones, as is shown in Fig. 5.

At this level, one major observation we noted is that Zhang and Shasha consider a postorder traversal, computing the forest distance by left-recursion on this search. As a consequence, we would need to consider a top-down parsing architecture to avoid redundant computations. However, top-down parsers are not computationally efficient, and a bottom-up approach, as is the case of ICE, requires a rightmost search of tree constituents. This implies redefining the architecture of the original matching strategy.

To accomplish this change, nodes in a tree T will be first numbered considering an inverse postorder traversal, as is shown in Fig. 4. We also introduce

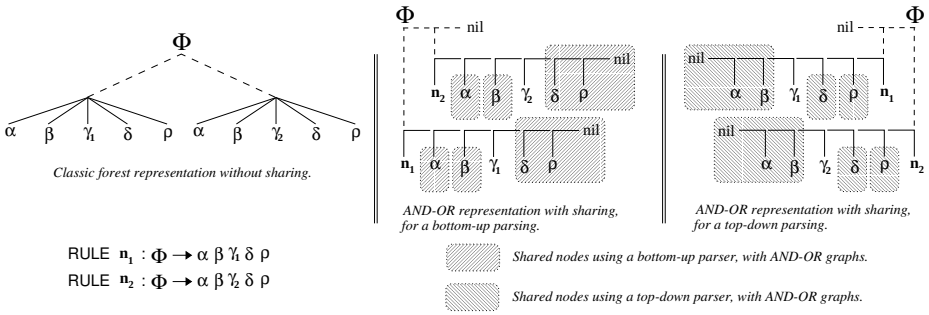

Fig. 5. How shared forest are built using an AND-OR formalism

r_keyroots(T), indicated by arrows in Fig. 4, as the set of all nodes in a tree T which have a right sibling plus the root, *root*(T), of T. And also we define $r(i)$ as the rightmost leaf descendant of the subtree rooted at T_i in a tree T.

From here, the alternative construction for the forest edition distance is analogous to the original algorithm, as shown in Fig. 6. For a better understanding we shall present the computations used with the inverse postorder. Given trees T_1 and T_2, and nodes $i_1 \in anc(i)$ and $j_1 \in anc(j)$, we have then that:

$$\text{f_d}(r(i_1)..i, r(j_1)..j) = \begin{cases} \min \left\{ \begin{array}{ll} \text{f_d}(r(i_1)..i-1,\ r(j_1)..j) & +\gamma(T_1[i] \to \varepsilon), \\ \text{f_d}(r(i_1)..i,\ r(j_1)..j-1) & +\gamma(\varepsilon \to T_2[j]), \\ \text{f_d}(r(i_1)..i-1,\ r(j_1)..j-1) & +\gamma(T_1[i] \to T_2[j]) \end{array} \right\} \\ \textbf{iff } r(i) = r(i_1) \textbf{ and } r(j) = r(j_1) \\ \\ \min \left\{ \begin{array}{ll} \text{f_d}(r(i_1)..i-1,\ r(j_1)..j) & +\gamma(T_1[i] \to \varepsilon), \\ \text{f_d}(r(i_1)..i,\ r(j_1)..j-1) & +\gamma(\varepsilon \to T_2[j]), \\ \text{f_d}(r(i_1)..r(i)-1,\ r(j_1)..r(j)-1) & +\text{t_d}(i,j) \end{array} \right\} \\ \textbf{otherwise} \end{cases}$$

To compute $t_d(T_1, T_2)$ it will be sufficient to take into account that

$$\text{t_d}(T_1, T_2) = \text{f_d}(\text{root}(T_1)..r(\text{root}(T_1)), \text{root}(T_2)..r(\text{root}(T_2)))$$

Lastly, time and space bounds are the same as in the classic Zhang and Shasha's algorithm.

5 Approximate Matching in Shared Forest

We now offer a simple explanation of how both environments, parsing and approximate tree matching, can be efficiently integrated in practice.

To start with, let T_1 be a labeled ordered tree, and T_2 an AND-OR graph, both of them built using our parsing frame. We shall identify T_1 with a query and T_2 with a part of the syntactic representation for a textual database with a certain degree of ambiguity. The presence of OR nodes in T_2 has two main

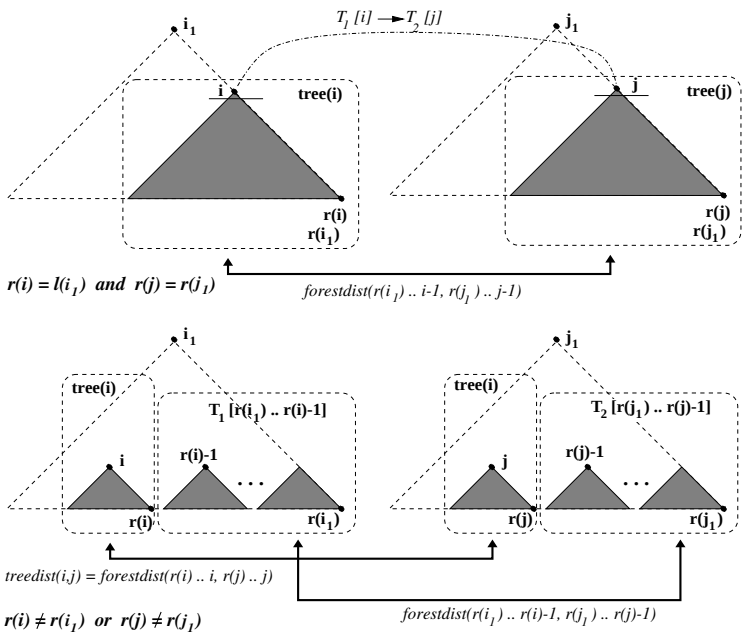

Fig. 6. The forest distance in our proposal

implications in our work: Firstly, there will exist situations where we must handle simultaneous values for some forest distances and, secondly, the parser may share some structures among the descendants of the different branches in an OR node. We shall now present the manner in which we calculate the distance between a pattern tree and the set of trees that are represented within the AND-OR graph, and how to take advantage of the shared structures created by the parser. The time complexity of this algorithm will be, in the worst case:

$$\mathcal{O}(\mid T_1 \mid\mid T_2 \mid \min(\text{depth}(T_1), \text{leaves}(T_1))\min(\text{depth}(T_2), \text{leaves}(T_2)))$$

where now $\mid T_2 \mid$ is the maximum number of nodes for a tree in T_2, leaves(T_2) is the maximum number of leaves for a tree in T_2, and depth(T_2) is the maximum depth for a tree in T_2.

Let $T_1[i]$ be the current node in the inverse postorder for T_1 and $i_1 \in \text{anc}(i)$ a r_keyroot. Given an OR node $T_2[k]$ we can distinguish two situations, depending on the situation of this OR node and the situation of the r_keyroots of T_2.

5.1 Sharing into a same r_keyroot

Let $T_2[j']$ and $T_2[j'']$ be the nodes we are dealing with in parallel for two branches labeled $T_2[k']$ and $T_2[k'']$ of the OR node $T_2[k]$. We have that $j_1 \in anc(j') \cap anc(j'')$, that is, the tree rooted at the r_keyroot $T_2[j_1]$ includes the OR alternatives $T_2[k']$ and $T_2[k'']$.

Such a situation is shown in Fig. 7 using a classic representation and the AND-OR graphs. Here, lightly shaded part refers to nodes whose distance have been computed in the inverse postorder before the OR node $T_2[k]$. The heavily shaded part represents a shared structure. The notation "• • •" in figures representing AND-OR graphs, expresses the fact that we descend along the rightmost branch of the corresponding tree.

We shall assume that nodes $T_2[r(j')-1]$ and $T_2[r(j'')-1]$ are the same, that is, their corresponding subtrees are shared. So, $T_2[r(j')]$ (resp. $T_2[r(j'')]$) is the following node in $T_2[k']$ (resp. $T_2[k'']$) to be dealt with once the distance for the shared structure has been computed.

At this point, our aim is to compute the value for $f_d(r(i_1)..i,\ r(j_1)..\hat{\jmath})$, $\hat{\jmath} \in \{j',\ j''\}$, proving that we can translate parse sharing into sharing on computations for these distances.

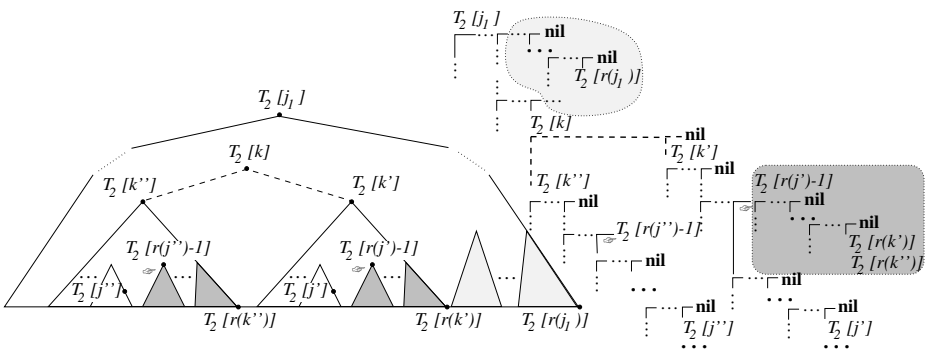

Fig. 7. Sharing into a same r_keyroot

Formally, the values for $f_d(r(i_1)..i,\ r(j_1)..\hat{\jmath})$, $\hat{\jmath} \in \{j',\ j''\}$ are given by:

$$\text{f_d}(r(i_1)..i, r(j_1)..\hat{\jmath}) = \begin{cases} \min \left\{ \begin{array}{ll} \text{f_d}(r(i_1)..i-1,\ r(j_1)..\hat{\jmath}) & +\ \gamma(T_1[i] \to \varepsilon), \\ \text{f_d}(r(i_1)..i,\ r(j_1)..\hat{\jmath}-1) & +\ \gamma(\varepsilon \to T_2[\hat{\jmath}]), \\ \text{f_d}(r(i_1)..i-1,\ r(j_1)..\hat{\jmath}-1) & +\ \gamma(T_1[i] \to T_2[\hat{\jmath}]) \end{array} \right\} \\ \textbf{iff } r(i) = r(i_1) \textbf{ and } r(\hat{\jmath}) = r(j_1) \\ \\ \min \left\{ \begin{array}{ll} \text{f_d}(r(i_1)..i-1,\ r(j_1)..\hat{\jmath}) & +\ \gamma(T_1[i] \to \varepsilon), \\ \text{f_d}(r(i_1)..i,\ r(j_1)..\hat{\jmath}-1) & +\ \gamma(\varepsilon \to T_2[\hat{\jmath}]), \\ \text{f_d}(r(i_1)..r(i)-1,\ r(j_1)..r(\hat{\jmath})-1) & +\ \text{t_d}(i, \hat{\jmath}) \end{array} \right\} \\ \textbf{otherwise} \end{cases}$$

where $\hat{\jmath} \in \{j',\ j''\}$. Here, $r(j_1) \neq r(\hat{\jmath})$, since we have assumed there is a shared structure between $T_2[r(\hat{\jmath})]$ and $T_2[r(j_1)]$. So, we can focus on the alternative computation, where:

1. The values for $f_d(r(i_1)..i-1,\ r(j_1)..\hat{\jmath})$, $\hat{\jmath} \in \{j',\ j''\}$ have been computed by the approximate matching algorithm in a previous step. So, in this case, parse sharing has not consequences on the natural computation for the distances.

2. Two cases are possible in relation to the nature of nodes $T_2[\hat{\jmath}]$, $\hat{\jmath} \in \{j',\ j''\}$, these are:
   - If both nodes are leaves, then $r(\hat{\jmath}) = \hat{\jmath}$. As a consequence, we have that

$$T_2[j'-1] = T_2[r(j')-1] = T_2[r(j'')-1] = T_2[j''-1]$$

     and the values $f_d(r(i_1)..i,\ r(j_1)..\hat{\jmath}-1)$, $\hat{\jmath} \in \{j',\ j''\}$ are also the same.
   - Otherwise, following the inverse postorder, we would arrive at the rightmost leaves of $T_2[j']$ and $T_2[j'']$, where we could apply the reasoning considered in the previous case.
3. Values for the distances $f_d(r(i_1)..r(i)-1,\ r(j_1)..r(\hat{\jmath})-1)$, $\hat{\jmath} \in \{j',\ j''\}$ are identical, given that nodes $T_2[r(\hat{\jmath})-1]$, $\hat{\jmath} \in \{j',\ j''\}$ are shared by the parser.

5.2 Sharing between different r_keyroots

We have that $j_1' \in anc(j')$ and $j_1'' \in anc(j'')$, with $j_1' \neq j_1''$, are two r_keyroots. We also have an OR node $T_2[k]$ being a common ancestor of these two nodes. We suppose that the r_keyroots are in different branches, that is, there exists a r_keyroot, $T_2[j_1']$ (resp. $T_2[j_1'']$), in the branch labeled $T_2[k']$ (resp. $T_2[k'']$).

Our aim now is to compute the value for distances $f_d(r(i_1)..i,\ r(\hat{\jmath}_1)..\hat{\jmath})$, where pairs $(\hat{\jmath}_1, \hat{\jmath})$ are in $\{(j_1',\ j'),\ (j_1'', j''\ )\}$. Formally, we have that these values are given by:

$$\text{f_d}(r(i_1)..i, r(\hat{\jmath}_1)..\hat{\jmath}) = \begin{cases} \min \left\{ \begin{array}{ll} \text{f_d}(r(i_1)..i-1,\ r(\hat{\jmath}_1)..\hat{\jmath}) & +\gamma(T_1[i] \to \varepsilon), \\ \text{f_d}(r(i_1)..i,\ r(\hat{\jmath}_1)..\hat{\jmath}-1) & +\gamma(\varepsilon \to T_2[\hat{\jmath}]), \\ \text{f_d}(r(i_1)..i-1,\ r(\hat{\jmath}_1)..\hat{\jmath}-1) & +\gamma(T_1[i] \to T_2[\hat{\jmath}]) \end{array} \right\} \\ \textbf{iff } r(i) = r(i_1) \textbf{ and } r(\hat{\jmath}) = r(\hat{\jmath}_1) \\ \\ \min \left\{ \begin{array}{ll} \text{f_d}(r(i_1)..i-1,\ r(\hat{\jmath}_1)..\hat{\jmath}) & +\gamma(T_1[i] \to \varepsilon), \\ \text{f_d}(r(i_1)..i,\ r(\hat{\jmath}_1)..\hat{\jmath}-1) & +\gamma(\varepsilon \to T_2[\hat{\jmath}]), \\ \text{f_d}(r(i_1)..r(i)-1,\ r(\hat{\jmath}_1)..r(\hat{\jmath})-1) & +\text{t_d}(i, \hat{\jmath}) \end{array} \right\} \\ \textbf{otherwise} \end{cases}$$

The situation, shown in Fig. 8, makes possible $r(i) = r(i_1)$ and $r(\hat{\jmath}) = r(\hat{\jmath}_1)$. In this first case, we can assume that a tail of sons is shared by nodes $T[\hat{\jmath}]$, $\hat{\jmath} \in \{j',\ j''\}$. We can also assume that this tail is proper given that, otherwise, our parser guarantees that the nodes $T_2[\hat{\jmath}]$, $\hat{\jmath} \in \{j',\ j''\}$ are also shared.

Taking into account our parsing strategy, which identifies syntactic structures and computations, we conclude that the distances $f_d(r(i_1)..i,\ r(\hat{\jmath}_1)..\hat{\jmath})$, with $(\hat{\jmath}_1, \hat{\jmath}) \in \{(j_1',\ j'), ; (j_1'', j''\ )\}$ do not depend on previous computations over the shared tail, such as is shown in the left-hand-side of Fig. 8. So, this sharing has no consequences on the calculus, although it will have effects on the computation of distances for nodes in the rightmost branch of the tree immediately to the left of the shared tail of sons, which is denoted by a double pointed line in Fig. 8, as we shall show immediately.

We consider now the second case, that is, the computation of the forest distance when $r(\hat{\jmath}_1) \neq r(\hat{\jmath})$, as is shown in Fig. 9. Here, in relation to each of the three alternative values to compute the minimum, we have that:

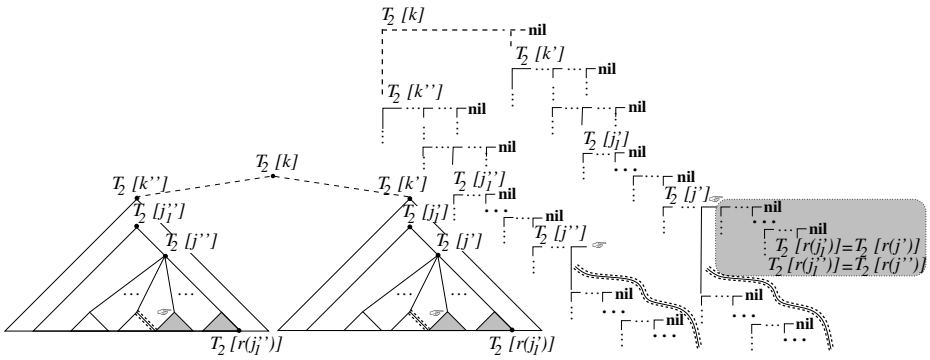

Fig. 8. Sharing between different r_keyroots (first case)

1. The values for $f_d(r(i_1)..i-1,\ r(\hat{\jmath}_1)..\hat{\jmath}),\ (\hat{\jmath}_1,\hat{\jmath}) \in \{(j'_1,\ j'),\ (j''_1, j''\ )\}$ have been computed by the approximate matching algorithm in a previous step and parse sharing does not affect the computation for distances.
2. We distinguish two cases in relation to the nature of nodes $T_2[\hat{\jmath}],\ \hat{\jmath} \in \{(j',\ j'')$. We shall apply the same reasoning considered when we had an only r_keyroot:
   - If both nodes are leaves, then $r(\hat{\jmath}) = \hat{\jmath}$. As a consequence, we have that

     $$T_2[j'-1] = T_2[r(j')-1] = T_2[r(j'')-1] = T_2[j''-1]$$

     and therefore the values for distances $f_d(r(i_1)..i,\ r(\hat{\jmath}_1)..\hat{\jmath}{-}1)$ with $(\hat{\jmath}_1,\hat{\jmath}), \in \{(j'_1,\ j'),\ (j''_1, j''\ )\}$, are also the same.
   - Otherwise, following the inverse postorder, we arrive at the rightmost leaves of $T_2[j']$ and $T_2[j'']$, where we can apply the reasoning considered in the previous case.

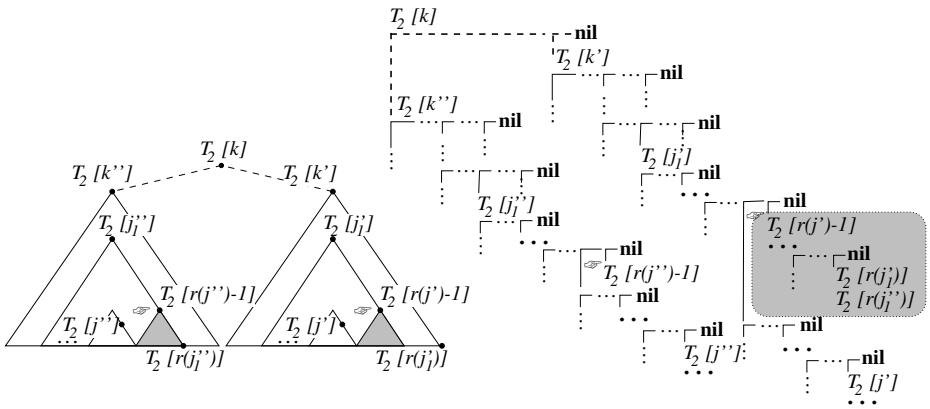

Fig. 9. Sharing between different r_keyroots (second case)

3. Values for the distances $f_d(r(i_1)..r(i)-1,\ r(\hat{j}_1)..r(\hat{\jmath})-1)$, $\hat{\jmath} \in \{j',\ j''\}$ are identical, given that the trees rooted by nodes $T_2[r(\hat{\jmath})-1]$, $\hat{\jmath} \in \{j',\ j''\}$ are shared by the parser.

6 Experimental Results

We consider the language of arithmetical expressions to illustrate the discussion, comparing our proposal with Tai [6], and Zhang and Shasha's algorithm [8]. We consider two deterministic grammars, $\mathcal{G}_L$ and $\mathcal{G}_R$, representing respectively the left and right associative versions for the arithmetic operators; and a non-deterministic one $\mathcal{G}_N$. We assume that parsers are built using ICE [7], and tests have been applied on data inputs of the form $a_1+a_2+\ldots+a_i+a_{i+1}$, with i even, representing the number of addition operators. In the non-deterministic case, these programs have a number of ambiguous parses which grows exponentially with i. This number is:

$$C_0 = C_1 = 1 \quad \text{and} \quad C_i = \binom{2i}{i}\frac{1}{i+1}, \text{ if } i > 1$$

As our pattern, we have used deterministic parse trees from inputs of the form $a_1 + b_1 + a_3 + b_3 + \ldots + b_{i-1} + a_{i-1} + b_{i+1} + a_{i+1}$, where $b_j \neq a_{j-1}$, for all $j \in \{1, 3, \ldots i-1, i+1\}$.

In the deterministic case, patterns are built from the left-associative (resp. right-associative) interpretation for $\mathcal{G}_L$ (resp. $\mathcal{G}_R$), which allows us to evaluate the impact of traversal orientation in the performance. So, Fig. 10 proves the adaptation of our proposal (resp. Zhang and Shasha's algorithm) to left-recursive (resp. right-recursive) derivations, which corroborates our conclusions. These tests also show the independence of Tai's algorithm from the grammar rules topology. This is due to the fact that mapping between two nodes is not computed from the mapping between their descendents, but from their ancestors, where structural sharing is not allowed by the parser. As a consequence, Tai's approach does not benefit from the dynamic programming architecture.

In the non-deterministic case, patterns are built from the left-associative interpretation of the query, which is not relevant given that rules in $\mathcal{G}_N$ are symmetrical. Here, we evaluate the gain in efficiency due to sharing of computations in a dynamic frame, as is also shown in Fig. 10.

7 Conclusions

Most classic approaches for dealing with tree-to-tree correction have been proposed in the context of dynamic programming algorithms. In turn, tabulation techniques are becoming a common way of dealing with highly redundant computations occurring, for instance, in natural language processing. However, no previous work has been developed in order to profit from this characteristic in the tree matching domain.

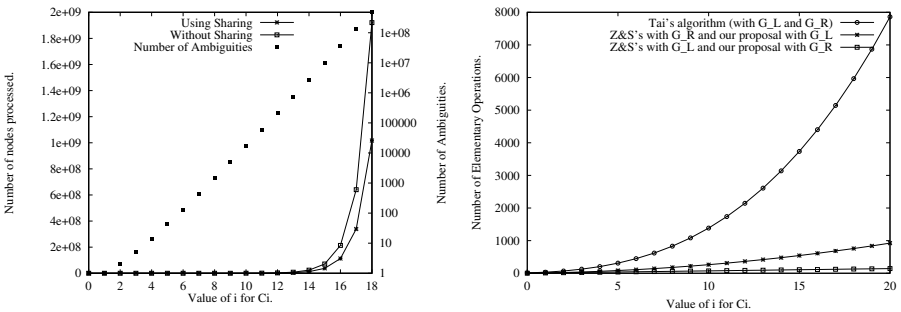

Fig. 10. Results on approximate tree matching

In this paper, we have proved that tree matching can be adapted to deal with shared forest in the context of information retrieval. To do this, the impact of the parsing strategy on the resulting shared structure should be studied. This allows formal justification of the type of traversal used to visit nodes during the matching, obtaining the maximum advantage from parse sharing.

References

1. Kilpelainen, P., Mannila, H.: Grammatical Tree Matching. Lecture Notes in Computer Science **644** (1992) 159–171.
2. Kilpelainen, P.: Tree Matching Problems with Applications to Structured Text Databases. Ph.D. Thesis, Department of Computer Science, University of Helsinki, 1992. Helsinki, Finland
3. Smeaton, Alan F. : Incorporating Syntactic Information into a Document Retrieval Strategy: An Investigation. Proc. the 9th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 103–113, 1986.
4. Smeaton, A.F. and van Rijsbergen, C.J.: Experiments on Incorporating Syntactic Processing of User Queries into a Document Retrieval Strategy. Proc. of the 11th International ACM-SIGIR Conference on Research and Development in Information Retrieval, pp. 31-54. Grenoble, France, 1988.
5. Smeaton, A.F and O'Donell, R. anf Kelley,F.: Indexing Structures Derived from Syntax in TREC-3: System Description. Proc. of 3rd Text REtrieval Conference (TREC-3), D.K. Harman (ed.), NIST Special Publication, 1994.
6. Tai, Kuo-Chung.: Syntactic error correction in programming languages. IEEE Transactions on Software Engineering **4(5)** (1978) 414–425.
7. Vilares, M., Dion, B.A.: Efficient incremental parsing for context-free languages. Proc. of the 5th IEEE International Conference on Computer Languages (1994) 241–252, Toulouse, France.
8. Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. SIAM Journal on Computing (1989) **18** 1245–1262.
9. Zhang, K. and Shasha, D. and Wang, J.T.L. Approximate Tree Matching in the Presence of Variable Length Don't Cares. Journal of Algorithms, pages 33–66, vol **16 (1)**, 1994