On Integration of Parsing and Tree Matching Schemes

M. Vilares Ferro
D. Cabrero Souto
F.J. Ribadas Pena

In this paper, we present a proposal intended to demonstrate the applicability of tabulation techniques to pattern recognition problems, when we deal 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.

To attain this goal, we study the existing relationship between parsing schema and tree-to-tree correction in a dynamic programming frame. As a result, we describe an algorithm that efficiently profits from sharing in syntactic structures.

1 INTRODUCTION

One critical aspect of an information system is indexing, that is, the representation of concepts to get a well formed data structure for search. Until recently, this task was accomplished by creating a bibliographic citation that references the original text. This approach allows a reduction in time and space bounds, although it does not facilitate the user from finding relevant information. In effect, it is the combination of the 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. This information is inherent in the document and query. So, matching becomes a possible mechanism to extract a common pattern from multiple data and we could use it for indexing and querying in information retrieval systems.

However, the language intended to represent the document can often only be approximately defined, and ambiguity arises. So, it is convenient to merge parse trees as much as possible into a single structure that allows them to share common parts. Queries could also vary from the indices 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 A DYNAMIC FRAME FOR PARSING

We introduce ICE [Vilares & Dion 1994], a parsing frame in dynamic programming. Our aim is to parse sentences in the language $\mathcal{L}(\mathcal{G})$ generated by a context-free grammar $\mathcal{G} = (N, \Sigma, P, S)$, where N is the set of non-terminals, Σ the set of terminal symbols, P the rules and S the start symbol. The empty string will be represented by ε.

2.1 The operational model

We assume that, using a standard technique, we produce a *push-down automaton* (PDA) from the grammar $\mathcal{G}$. In practice, we chose a LALR(1) device, possibly non-deterministic, which will allow us to improve sharing of computations. Formally, a PDA can be represented as a 7-tuple $\mathcal{A} = (\mathcal{Q}, \Sigma, \Delta, \delta, q_0, Z_0, \mathcal{Q}_f)$ where: $\mathcal{Q}$ is the set of states, Σ the set of input symbols, Δ the set of stack symbols, q_0 the initial state, Z_0 the initial stack symbol, $\mathcal{Q}_f$ the set of final states, and δ a finite set of transitions of the form $p\ X\ a \mapsto q\ Y$ with $p, q \in \mathcal{Q}$, $a \in \Sigma \cup \{\varepsilon\}$ and $X, Y \in \Delta \cup \{\varepsilon\}$.

Let the PDA be in a configuration $(p, X\alpha, ax)$, where p is the current state, $X\alpha$ is the stack contents with X on the top, ax is the remaining input where the symbol a is the next to be shifted, $x \in \Sigma^*$. The application of $p\ X\ a \mapsto q\ Y$ results in a configuration $(q, Y\alpha, x)$ where the terminal symbol a has been scanned, X has been popped, and Y has been pushed. If the terminal symbol a is ε in the transition, no input symbol is scanned. If X is ε then no stack symbol is popped from the stack. In a similar manner, if Y is ε then no stack symbol is pushed on the stack. In the case of ambiguous recognition, several such transitions can be applied.

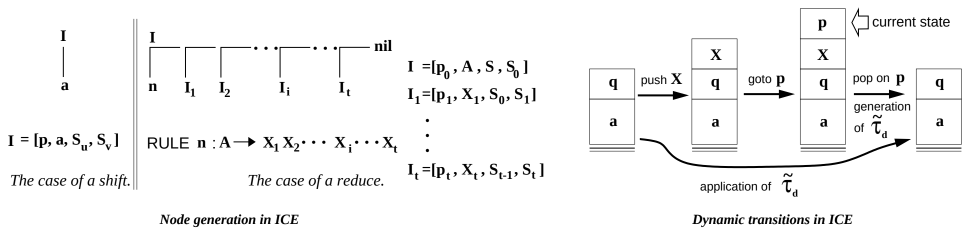


Figure 1: Node generation and dynamic transitions in ICE

We proceed by building *items*, compact representations of the recognizer stacks. New items are produced by applying transitions to existing ones, until no new application is possible. We associate a set of items S_i^w, called *itemset*, for the symbol w_i at the position i in the input string of length n, $w_{1..n}$.

An item has the form $[p, X, S_j^w, S_i^w]$, where p is a PDA state, X is a stack symbol, S_j^w is the *back pointer* to the itemset associated to the input symbol w_j at which we began to look for that configuration of the automaton, and S_i^w is the current itemset. A merit ordering technique guarantees fairness and completeness, while operations may add more items to the current itemset and may also put items in the itemset corresponding to the following token to be analyzed from the input string. To ignore redundant items we use a simple subsumption relation based on the equality.

2.2 The parser

We represent a parse as the chain of the context-free rules used in a leftmost reduction of the input sentence [Vilares & Dion 1994], 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. 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 sharing of a part of the descendants of a given node, in fact, a consequence of the binary nature of the transition protocol previously described. This feature allows to prove [Vilares & Dion 1994] that time complexity for the parser is, in the worst case, $\mathcal{O}(n^3)$ when the length of the input string is n. Space complexity is, also in the worst case, $\mathcal{O}(n^2)$.

Items are used as non-terminals of an output grammar $\mathcal{G}_o = (N_o, \Sigma_o, P_o, S_o)$, where N_o is the set of all items, Σ_o the set of input symbols of the original grammar $\mathcal{G}$, and the rules in P_o are constructed together with their left-hand-side item I by the parsing algorithm. We generate a rule for the output grammar each time a reduce or a shift action from the grammar defining the language is applied on the stack, as shown in Fig. 1. The start symbol S_o is the last item produced by a successful computation.

We build a *push-down transducer* (PDT), $\mathcal{T}_\mathcal{G} = (\mathcal{Q}, \Sigma, \Delta, \Pi, \delta, q_0, Z_0, \mathcal{Q}_f)$, from our PDA model augmenting it with a component Π representing the set of output symbols, and considering transitions in δ of the form $p \;\; X \;\; a \;\mapsto\; q \;\; Y \;\; u$ with $p, q \in \mathcal{Q}$; $a \in \Sigma \cup \{\varepsilon\}$; $X, Y \in \Delta \cup \{\varepsilon\}$, and $u \in \Pi^*$.

Given $\tau = \delta(p, X, a) \ni (q, Y, u)$, we translate it to items of the following form:

$$\begin{array}{llcll}
1. & \tilde{\delta}([p, X, S^w_j, S^w_i], a) & \ni & ([q, \varepsilon, S^w_i, S^w_i], \varepsilon) & \textbf{if} \;\; Y = X \\
2. & \tilde{\delta}([p, X, S^w_j, S^w_i], a) & \ni & ([p, Y, S^w_i, S^w_{i+1}], I_0 \rightarrow a) & \textbf{if} \;\; Y = a \\
3. & \tilde{\delta}([p, X, S^w_j, S^w_i], a) & \ni & ([p, Y, S^w_i, S^w_i], I_1 \rightarrow I_2) & \textbf{if} \;\; Y \in N \\
4. & \tilde{\delta}([p, \varepsilon, S^w_j, S^w_i], a) & \ni & \tilde{\delta}_d([q, \varepsilon, S^w_l, S^w_i], a) \ni ([q, \varepsilon, S^w_l, S^w_j], I_3 \rightarrow I_4 I_5) & \textbf{if} \;\; Y = \varepsilon \\
 & & & \forall q \in \mathcal{Q} \text{ such that: } \exists\, \delta(q, X, \varepsilon) \ni (p, X, \varepsilon) &
\end{array}$$

with: $\tilde{\delta} : \text{It} \times \Sigma \cup \{\varepsilon\} \longrightarrow \{\text{It} \cup \tilde{\delta}_d\} \times \Pi^*$ $\qquad$ $\tilde{\delta}_d : \text{It} \times \Sigma \cup \{\varepsilon\} \longrightarrow \text{It}$
and

$$\begin{array}{lll}
I_0 = [p, Y, S^w_i, S^w_{i+1}] & I_1 = [p, Y, S^w_i, S^w_i] & I_2 = [p, X, S^w_j, S^w_i] \\
I_3 = [q, \varepsilon, S^w_l, S^w_i] & I_4 = [q, X, S^w_l, S^w_j] & I_5 = [p, \varepsilon, S^w_j, S^w_i]
\end{array}$$

where *It* is the set of all parse items and $\tilde{\delta}_d$ is called the set of *dynamic transitions*. Succinctly, we can describe the preceding cases as follows:

1. A goto action from the state p to state q under transition X in the LALR(1) automaton.
2. A push of terminal a from state p. The new item belongs to the next itemset S^w_{i+1}.
3. A push of non-terminal Y from state p.
4. A pop action from state p, where q is an ancestor of p under transition X. In this case, we do not generate a new item, but a *dynamic transition* $\tilde{\tau}_d$ to treat the absence of information about the rest of the stack. This transition is applicable to the configuration resulting of the first one, but also on those to be generated and sharing the same syntactic structure, as shown in Fig. 1.

3 A DYNAMIC FRAME FOR APPROXIMATE TREE MATCHING

We introduce the Zhang and Shasha's approach [Zhang & Shasha 1989] to determine the distance between two trees as measured by the number of edit operations needed to transform one tree into the other.

3.1 The operational model

Given trees, T_1 and T_2, we define an *edit operation* as a pair $a \rightarrow b$, $a \in \text{labels}(T_1) \cup \{\varepsilon\}$, $b \in \text{labels}(T_2) \cup \{\varepsilon\}$, $(a, b) \neq (\varepsilon, \varepsilon)$. Deleting a node a, $a \rightarrow \varepsilon$, means making the children of a become the children of the parent of a and then removing a. Inserting, $\varepsilon \rightarrow b$, is the complement of delete. Changing a node, $a \rightarrow b$, means changing the node a label into b. Generic examples of such edit operations are shown in Fig. 2.

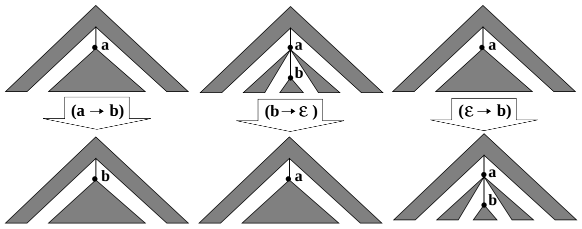


Figure 2: Edit operations

Each edit operation has an associated cost, $\gamma(a \rightarrow b)$, defined by a metric γ, that we can extend to a sequence S of edit operations $s_1,\ s_2,\ \ldots,\ s_n$ in the form $\gamma(S) = \sum_{i=1}^{|S|}(\gamma(s_i))$. Formally, the distance between T_1 and T_2 is defined by the metric:

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

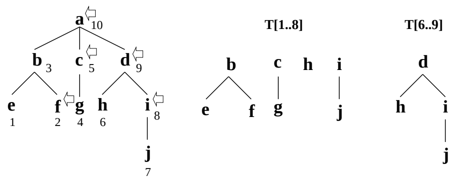
In order to simplify the computation of δ, Tai introduces in [Tai 1978] a particular kind of edit sequences called *mappings*. Given a tree T, we shall consider a postorder traversal to name each node i uniquely by $T[i]$, as shown in the left-hand side of Fig. 3. Then, a mapping from T_1 to T_2 is a 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** $j_1 = j_2$	(one-to-one)
$T_1[i_1]$ is to the left of $T_1[i_2]$ **iff** $T_2[j_1]$ is to the left of $T_2[j_2]$	(sibling order)
$T_1[i_1]$ is an ancestor of $T_1[i_2]$ **iff** $T_2[j_1]$ is an ancestor of $T_2[j_2]$	(ancestor order)

We show, in the right-hand side of Fig. 3, an example of mapping between two trees. The rightmost diagram includes a sequence of edit operations not constituting a mapping. 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 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 reduce the computation effort, focusing only on edit sequences being a mapping.



Postorder traversal of a tree.

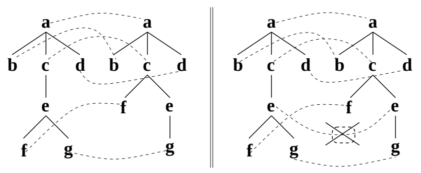

An example on mappings.

Figure 3: Postorder traversal and mappings

3.2 The algorithm

A major characteristic of the Zhang and Shasha's algorithm is its bottom-up oriented approach. The minimum cost mapping between two nodes in different trees depends only on mapping the nodes and their children. More exactly, given the *l_keyroots*(T), the set of all nodes in T which have a left sibling plus the root, *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 subforest of T induced by the nodes numbered i to j inclusive, as it is shown in the left-hand side of Fig. 3. In particular, we have $T[l(i)..i]$ is the tree rooted at $T[i]$. We define the *forest edition distance* as a generalization of δ, in the form

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

that we shall denote *forest_dist*$(i_1..i_2, j_1..j_2)$ 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 *tree_dist*(T_1, T_2) applying the formulas that follows, for nodes $i_1 \in anc(i)$ and $j_1 \in anc(j)$, such as illustrated in Fig. 4, taking into account the different cases we have that forest_dist$(l(i_1)..i, l(j_1)..j)$ is:

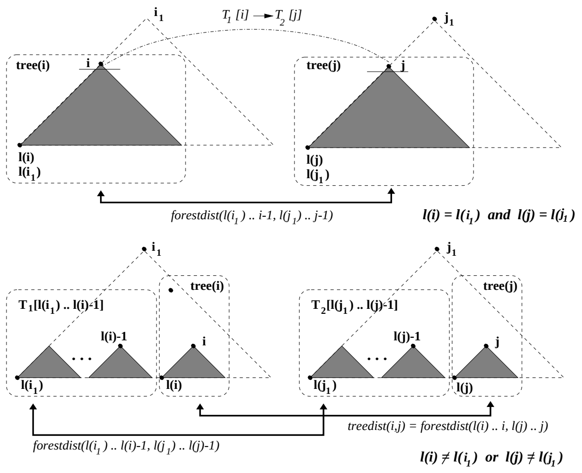

i_1
$T_1[i] \to T_2[j]$
j_1
tree(i)
i
tree(j)
j
l(i)
$l(i_1)$
l(j)
$l(j_1)$
forestdist($l(i_1)$ .. i-1, $l(j_1)$ .. j-1)
$l(i) = l(i_1)$ and $l(j) = l(j_1)$
i_1
tree(i)
$T_1[l(i_1) .. l(i)-1]$
l(i)-1
i
$l(i_1)$
l(i)
j_1
tree(j)
$T_2[l(j_1) .. l(j)-1]$
l(j)-1
j
$l(j_1)$
l(j)
treedist(i,j) = forestdist(l(i) .. i, l(j) .. j)
forestdist($l(i_1)$ .. l(i)-1, $l(j_1)$ .. l(j)-1)
$l(i) \neq l(i_1)$ or $l(j) \neq l(j_1)$

Figure 4: The forest distance in Zhang and Shasha's algorithm

$$\min \left\{ \begin{array}{lll} \text{forest\_dist}(l(i_1)..i-1,\ l(j_1)..j) & + & \gamma(T_1[i] \to \varepsilon), \\ \text{forest\_dist}(l(i_1)..i,\ l(j_1)..j-1) & + & \gamma(\varepsilon \to T_2[j]), \\ \text{forest\_dist}(l(i_1)..i-1,\ l(j_1)..j-1) & + & \gamma(T_1[i] \to T_2[j]) \end{array} \right\}$$

iff $l(i) = l(i_1)$ **and** $l(j) = l(j_1)$

$$\min \left\{ \begin{array}{lll} \text{forest\_dist}(l(i_1)..i-1,\ l(j_1)..j) & + & \gamma(T_1[i] \to \varepsilon), \\ \text{forest\_dist}(l(i_1)..i,\ l(j_1)..j-1) & + & \gamma(\varepsilon \to T_2[j]), \\ \text{forest\_dist}(l(i_1)..l(i)-1,\ l(j_1)..l(j)-1) & + & \text{tree\_dist}(i,j) \end{array} \right\}$$

otherwise

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

$$\text{tree\_dist}(T_1, T_2) = \text{forest\_dist}(l(\text{root}(T_1))..\text{root}(T_1), l(\text{root}(T_2))..\text{root}(T_2))$$

It is important to remark that conditions of the type $l(i_1) \neq l(i)$ (resp. $l(j_1) \neq l(j)$) rely on nodes in *l_keyroots*(T_1) (resp. nodes in *l_keyroots*(T_2)), as it is also the case of *root*(T_1) (resp. *root*(T_2)). The time complexity of this algorithm is $\mathcal{O}(n_1 n_2 min(d_1, l_1) min(d_2, l_2))$, and the space complexity is $\mathcal{O}(n_1 n_2)$, both in the worst case, where n_i is the number of nodes in the tree T_i, d_i is the depth of T_i and l_i is the number of leaves in T_i, $i = 1, 2$.

4 RELATING PARSING AND TREE MATCHING

The major question of previous related works [Zhang & Shasha 1989], is the tree distance algorithm itself. However, in dealing with information retrieval often parsing and tree-to-tree correction are topologically related. Typically, this relation is the case when we deal with unrestricted natural language texts, including ambiguous sentences, and the syntactic representation used for indexing can be composed by trees with some common parts. In this context, the number of matching operations would be uselessly multiplied and sharing of data structures and computations would save space needed to represent the trees. To attain this goal, we first need to isolate the factors at the origin of structural sharing.

4.1 Sharing of a tail of sons

To get the best performance, it is necessary to understand the mechanisms that cause the phenomenon of tree duplication. In this case, our parsing framework is interesting because it identifies syntactic and computation structures in the same concept, the item.

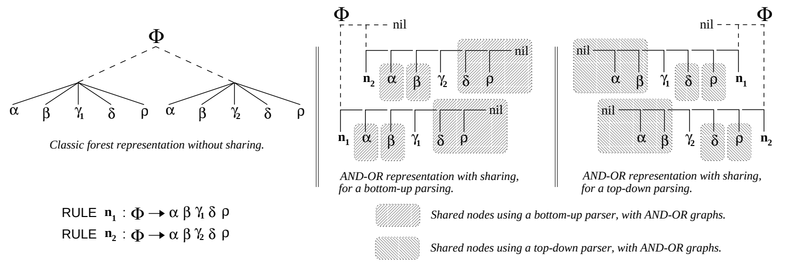

Φ
α β γ1 δ ρ α β γ2 δ ρ
Classic forest representation without sharing.
RULE n1 : Φ → α β γ1 δ ρ
RULE n2 : Φ → α β γ2 δ ρ
nil
n2 α β γ2 δ ρ
n1 α β γ1 δ ρ
AND-OR representation with sharing, for a bottom-up parsing.
n1 n2
AND-OR representation with sharing, for a top-down parsing.
Shared nodes using a bottom-up parser, with AND-OR graphs.
Shared nodes using a top-down parser, with AND-OR graphs.

Figure 5: How shared forest are built using an AND-OR *formalism*

Given that actions on the PDA depend on the first and possibly second elements in the stack, the output grammar is a *2-form* one and 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 relies to the type of search used to built the forest. Breadth first search results on bottom-up constructions and depth first search results on top-down ones, as it is shown in Fig. 5.

Taking into account that our parsing scheme is bottom-up, rightmost search of tree constituents should be considered in order to take advantage from sharing achieved during the parsing process.

4.2 Adapting the approximate tree matching strategy

One major observation we noted is that Zhang and Shasha consider a postorder traversal, computing the forest distance by left-recursion on this search. Realizing that our parser shares computations from right-to-left, we contend that an efficient integration of both algorithms requires to redefine 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 shown in Fig. 6. We also introduce *r_keyroots*(T) 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.

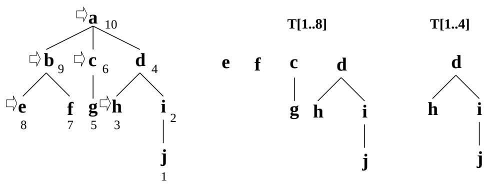


Figure 6: The forest distance using an inverse postorder numbering

From here, the alternative construction for the forest edition distance is analogous to the original algorithm, as shown in Fig. 7. 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 that forest_dist$(r(i_1)..i, r(j_1)..j)$ is :

$$\min \left\{ \begin{array}{lcl} \text{forest\_dist}(r(i_1)..i-1,\ r(j_1)..j) & + & \gamma(T_1[i] \rightarrow \varepsilon), \\ \text{forest\_dist}(r(i_1)..i,\ r(j_1)..j-1) & + & \gamma(\varepsilon \rightarrow T_2[j]), \\ \text{forest\_dist}(r(i_1)..i-1,\ r(j_1)..j-1) & + & \gamma(T_1[i] \rightarrow T_2[j]) \end{array} \right\}$$

iff $r(i) = r(i_1)$ **and** $r(j) = r(j_1)$

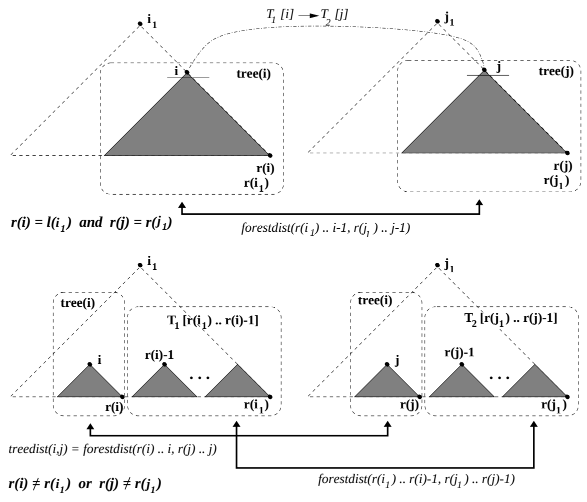
$$\min \left\{ \begin{array}{lcl} \text{forest\_dist}(r(i_1)..i-1,\ r(j_1)..j) & + & \gamma(T_1[i] \rightarrow \varepsilon), \\ \text{forest\_dist}(r(i_1)..i,\ r(j_1)..j-1) & + & \gamma(\varepsilon \rightarrow T_2[j]), \\ \text{forest\_dist}(r(i_1)..r(i)-1,\ r(j_1)..r(j)-1) & + & \text{tree\_dist}(i,j) \end{array} \right\}$$

otherwise

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

$$\text{tree\_dist}(T_1, T_2) = \text{forest\_dist}(\text{root}(T_1)..r(\text{root}(T_1)), \text{root}(T_2)..r(\text{root}(T_2)))$$

Lastly, time and space bounds are the same as in the classic postorder version.



$T_1[i] \rightarrow T_2[j]$
i_1
i
tree(i)
r(i)
$r(i_1)$
j_1
j
tree(j)
r(j)
$r(j_1)$
$r(i) = l(i_1)$ and $r(j) = r(j_1)$
$forestdist(r(i_1) .. i-1, r(j_1) .. j-1)$
tree(i)
$T_1 [r(i_1) .. r(i)-1]$
r(i)-1
tree(i)
$T_2 [r(j_1) .. r(j)-1]$
r(j)-1
$treedist(i,j) = forestdist(r(i) .. i, r(j) .. j)$
$r(i) \neq r(i_1)$ or $r(j) \neq r(j_1)$
$forestdist(r(i_1) .. r(i)-1, r(j_1) .. r(j)-1)$

Figure 7: The forest distance in our proposal

5 APPROXIMATE MATCHING IN SHARED FOREST

Once we have motivated and formalized the adaptation of a classic approximate matching approach to efficiently deal with bottom-up parsers, we now offer a simple explanation about how both environments, parsing, and approximate tree matching, can be 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 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 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.

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 situation is shown in Fig. 8 using a classic representation and the AND-OR graphs. Here, the part shaded in light color refers to nodes whose distance have been computed in the inverse postorder before the OR node $T_2[k]$. The part shaded in dark color represents a shared structure. The notation "• • •" in figures representing AND-OR graphs, expresses 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 deal with once the distance for the shared structure has been computed.

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

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

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

$$\min \left\{ \begin{array}{lcl} \text{forest\_dist}(r(i_1)..i-1,\ r(j_1)..\hat{\jmath}) & + & \gamma(T_1[i] \to \varepsilon), \\ \text{forest\_dist}(r(i_1)..i,\ r(j_1)..\hat{\jmath}-1) & + & \gamma(\varepsilon \to T_2[\hat{\jmath}]), \\ \text{forest\_dist}(r(i_1)..r(i)-1,\ r(j_1)..r(\hat{\jmath})-1) & + & \text{tree\_dist}(i,\hat{\jmath}) \end{array} \right\}$$
$$\textbf{otherwise}$$

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 *forest_dist*$(r(i_1)..i-1,\ r(j_1)..\hat{\jmath}),\ \hat{\jmath} \in \{j',\ j''\}$ have been computed by the matching algorithm in a previous step. So, in this case, parse sharing has no consequences on the natural computation for the distances.

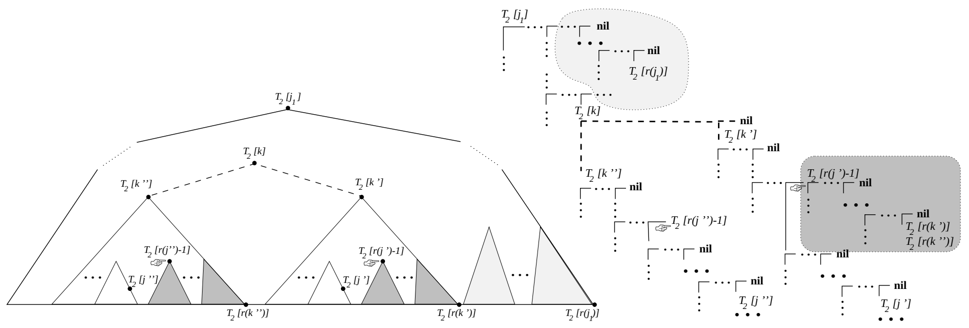


Figure 8: Sharing into a same r_keyroot

2. Two cases are possible in relation to the nature of nodes $T_2[\hat{\jmath}],\ \hat{\jmath} \in \{j',\ j''\}$:

   - 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 *forest_dist*$(r(i_1)..i,\ r(j_1)..\hat{\jmath} - 1),\ \hat{\jmath} \in \{j',\ j''\}$ 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 *forest_dist*$(r(i_1)..r(i) - 1,\ r(j_1)..r(\hat{\jmath}) - 1),\ \hat{\jmath} \in \{j',\ j''\}$ 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 *forest_dist*$(r(i_1)..i,\ r(\hat{\jmath}_1)..\hat{\jmath})$, where pairs $(\hat{\jmath}_1, \hat{\jmath})$ are in $\{(j'_1,\ j'),\ (j''_1, j''\ )\}$. Formally, we have that these values are given by:

$$\min \left\{ \begin{array}{lll} \text{forest\_dist}(r(i_1)..i-1,\ r(\hat{\jmath}_1)..\hat{\jmath}) & + & \gamma(T_1[i] \to \varepsilon), \\ \text{forest\_dist}(r(i_1)..i,\ r(\hat{\jmath}_1)..\hat{\jmath}-1) & + & \gamma(\varepsilon \to T_2[\hat{\jmath}]), \\ \text{forest\_dist}(r(i_1)..i-1,\ r(\hat{\jmath}_1)..\hat{\jmath}-1) & + & \gamma(T_1[i] \to T_2[\hat{\jmath}]) \end{array} \right\}$$
iff $r(i) = r(i_1)$ **and** $r(\hat{\jmath}) = r(\hat{\jmath}_1)$

$$\min \left\{ \begin{array}{lll} \text{forest\_dist}(r(i_1)..i-1,\ r(\hat{\jmath}_1)..\hat{\jmath}) & + & \gamma(T_1[i] \to \varepsilon), \\ \text{forest\_dist}(r(i_1)..i,\ r(\hat{\jmath}_1)..\hat{\jmath}-1) & + & \gamma(\varepsilon \to T_2[\hat{\jmath}]), \\ \text{forest\_dist}(r(i_1)..r(i)-1,\ r(\hat{\jmath}_1)..r(\hat{\jmath})-1) & + & \text{tree\_dist}(i,\hat{\jmath}) \end{array} \right\}$$
otherwise

The situation, shown in Fig. 9, makes possible $r(i) = r(i_1)$ and $r(\hat{\jmath}) = r(\hat{\jmath}_1)$. In the 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.

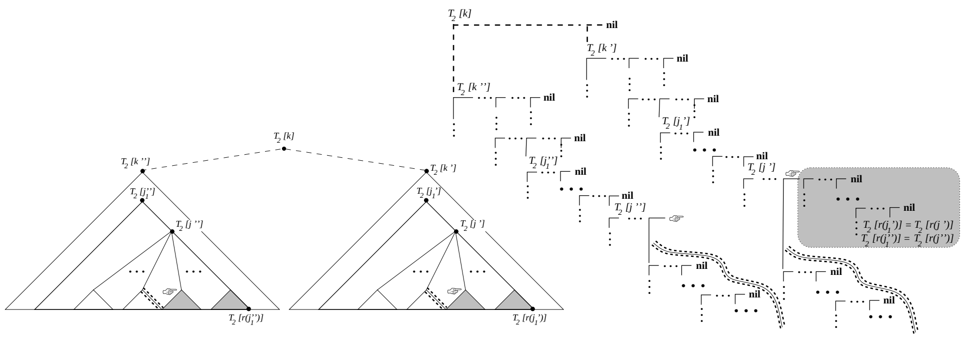


Figure 9: Sharing between different r_keyroots (first case)

Taking into account our parsing strategy, which identifies syntactic structures and computations, we conclude that the distances *forest_dist*$(r(i_1)..i,\ r(\hat{\jmath}_1)..\hat{\jmath})$, with $(\hat{\jmath}_1,\hat{\jmath}) \in \{(j_1',\ j'),\ (j_1'',j''\ )\}$ do not depend on previous computations over the shared tail, as shown in the left-hand side of Fig. 9. 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 are denoted by a double pointed line in Fig. 9.

We consider now the second case, that is, the computation of the forest distance when $r(\hat{\jmath}_1) \neq r(\hat{\jmath})$, as shown in Fig. 10. Here, in relation to each one of the three alternative values needed to compute the minimum, we have that:

1. For *forest_dist*$(r(i_1)..i-1,\ r(\hat{\jmath}_1)..\hat{\jmath}),\ (\hat{\jmath}_1,\hat{\jmath}) \in \{(j_1',\ j'),\ (j_1'',j''\ )\}$ the matching algorithm has computed the needed values 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 *forest_dist*$(r(i_1)..i,\ r(\hat{\jmath}_1)..\hat{\jmath} - 1)$ with $(\hat{\jmath}_1, \hat{\jmath}), \in \{(j_1',\ j'),\ (j_1'', j''\ )\}$, are also the same.
   - Otherwise, following the inverse postorder, we arrive to the rightmost leaves of $T_2[j']$ and $T_2[j'']$, where we can apply the reasoning considered in the previous case.

3. Values for the distances *forest_dist*$(r(i_1)..r(i) - 1,\ r(\hat{j}_1)..r(\hat{\jmath}) - 1),\ \hat{\jmath} \in \{j',\ j''\}$ 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.

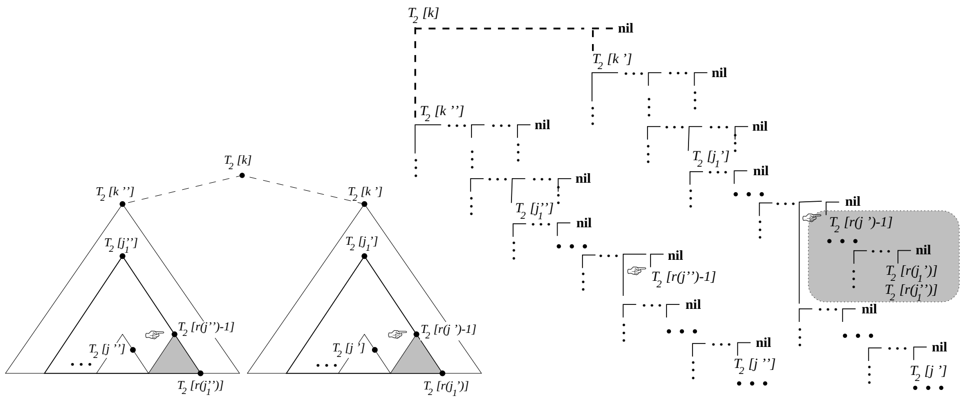


Figure 10: *Sharing between different r_keyroots (second case)*

6 EXPERIMENTAL RESULTS

To facilitate understanding we consider a simple example to illustrate our discussion: the language, $\mathcal{L}$, of arithmetic expressions. We compare our proposal with Tai [Tai 1978], and Zhang and Shasha's algorithm [Zhang & Shasha 1989]. We consider three different grammars generating $\mathcal{L}$: two deterministic, $\mathcal{G}_L$ and $\mathcal{G}_R$, representing respectively the left and right associative versions for the arithmetic operators; and one non-deterministic $\mathcal{G}_N$. To simplify the explanation, we focus on matching phenomena assuming that parsers are built using ICE [Vilares & Dion 1994]. Lexical information is common in all cases, and tests have been applied on target 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 expression. Given that, in the non-deterministic case, $\mathcal{G}_N$ contains a rule ”Exp ::= Exp + Exp”, these programs have a number of ambiguous parses which grows exponentially with *i*. This number is:

$$C_0 = C_1 = 1 \quad \text{and} \quad C_i = \left(\begin{array}{c} 2i \\ i \end{array} \right) \frac{1}{i+1}, \text{ if } i > 1$$

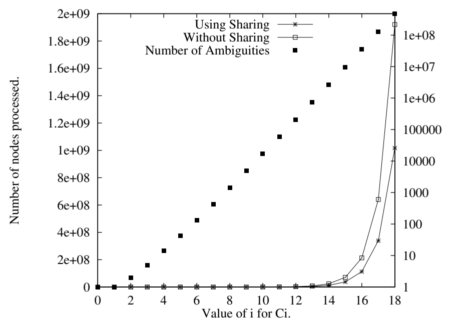
allowing us to study the compilation schema when highly redundant computations appears. As pattern, we have used deterministic parse trees from inputs of the form $a_1 + b_1 + a_3 + b_3 + \ldots b_{i-1} + a_{i-1} + b_{i+1} + a_{i+1}$, where $b_j \neq a_{j-1}$, for all $j \in \{1, 3, \ldots i-1, i+1\}$.

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 of the pattern matching algorithm. So, the rightmost diagram in Fig. 11 proves the adaptation of our proposal (resp. Zhang and Shasha’s algorithm) to left-recursive (resp. right-recursive) derivations. This corroborates our previous theoretical conclusions and justifies the interest of the work presented. These practical tests also show the independence of Tai’s algorithm of the grammar rules topology. This is basically 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 symmetric. Here, we evaluate the gain in efficiency due to sharing of computations in a dynamic frame, as shown in the leftmost diagram of Fig. 11.

7 CONCLUSIONS

Approximate tree matching can be adapted to deal with shared forest. To improve the performances, the impact of the parsing strategy in the resulting shared structure should be studied. This allows to formally justify the type of traversal used to visit nodes during the matching, taking the maximum advantage from parse sharing.



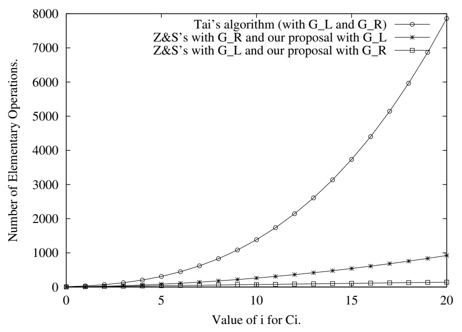


Figure 11: Results on approximate tree matching

REFERENCES

Kuo-Chung Tai, 1978.*Syntactic error correction in programming languages*. IEEE Transactions on Software Engineering, SE-4(5), pages 414-425.

M. Vilares and B. A. Dion, 1994.*Efficient incremental parsing for context-free languages*. Proc. of the 5^{th} IEEE International Conference on Computer Languages, pages 241-252, Toluose, France.

K. Zhang and D. Shasha, 1989. *Simple fast algorithms for the editing distance between trees and related problems*. In SIAM Journal of Computing, volume 18, pages 1245-1262.

Manuel Vilares Ferro is a professor at the Computer Science Department, University of A Coruña, Campus de Elviña s/n, 15071 A Coruña, SPAIN. *vilares@dc.fi.udc.es*.

David Cabrero Souto is a doctoral student at the Computer Science Department, University of A Coruña. *cabrero@dc.fi.udc.es*.

Francisco J. Ribadas Pena is a doctoral student at the Computer Science Department, University of A Coruña. *ribadas@mail2.udc.es*.

This work has been partially supported by projects XUGA 20402B97 of the Autonomous Government of Galicia, project PGIDT99XI10502B of Spanish Government and project 1FD97-0047-C04-02 by the European Community.