Pattern Matching as a Dynamic Facility to get Aboutness

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

Abstract

The goal of our work is to reduce the cost of evaluating queries in sophisticated retrieval systems. Our aim is to support the use of user-defined multiple views of the documents by applying retrieval techniques based on pattern-matching. One view can contain all the structure explicitly, while another can contain only part of the structure visible. Preliminary experimental results show that query evaluation time can be reduced by more than half with little impact on space complexity.

1 Introduction

The notion of *aboutness* is fundamental to information retrieval (IR). It appears in questions related to the definition of a document about the representation of an information need, and can be used to determine whether a term is a generalization of another term, in which case the information carried by the last is said to be about the information carried by the former. In most of actual IR systems, information is carried by a fixed set of relationship types over an underlying set of terms. These relationships, called indexes, are often based on a simple bibliographic citation that references the original text, and not always reflect the value of the concepts being presented. This leads us to considerate more sophisticated index formalisms, such as context-free grammars [2], better adapted to encapsulate the essence of these concepts. This information is inherent in the document and query. So, using context-free grammars as indexing model, matching becomes a natural way to evaluate queries in IR systems.

However, the correct indexing does not guarantee by itself an adequate exploitation for IR formalisms, that should be considered with respect to their applicability. In this sense, the operational model should also provide the flexibility necessary to access different views of the text, with a well-founded theoretical background.

M. Vilares, F.J. Ribadas and V.M. Darriba are with the Computer Science Department, University of A Coruña. Campus de Elviña s/n, 15071 A Coruña, Spain. E-mail: {vilares, ribadas, darriba}@dc.fi.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 EU.

Finally, speed is fundamental in the success of IR systems. Sophisticated retrieval implies expensive strategies, compounding the problem of providing answers quickly and efficiently. This depends on the capacity to exploit the output of a document analyzer in a domain where the language intended to represent the text can only be approximately defined, and ambiguity arises. Our aim is to reconcile practical IR and efficient context-free parsing in a dynamic programming frame using pattern matching as basis to get aboutness, and taking advantage of structural sharing in ambiguous parsing.

2 The parsing model

We chose ICE [5] as parsing model. The kernel of our proposal is an extended push-down transducer (PDT), formally a 8-tuple $\mathcal{T}_{\mathcal{G}} = (\mathcal{Q}, \Sigma, \Delta, \Pi, \delta, q_0, Z_0, \mathcal{Q}_f)$, where: $\mathcal{Q}$ is the set of states, Σ the set of input word symbols, Δ the set of stack symbols, Π the set of output symbols, q_0 the initial state, Z_0 the initial stack symbol, and $\mathcal{Q}_f$ the set of final states. In relation to δ, it is a finite set of transitions of the form $\tau = \delta(p, X, a) \ni (q, Y, u)$ with $p, q \in \mathcal{Q}$; $a \in \Sigma \cup \{\varepsilon\}$; $X, Y \in \Delta \cup \{\varepsilon\}$, and $u \in \Pi^*$.

The parser proceeds by building *items*, compact representations of the PDT 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 is of the form $[p, X, S_j^w, S_i^w]$, where p is a PDT state, X is a stack symbol, S_j^w is the *back pointer* to the itemset associated to the input symbol w_i at which we began to look for that configuration of the automaton, and S_i^w is the current itemset. These items, called S^1, provide an optimal treatment of sharing of computations and syntactic structures for non-deterministic inputs [5]. A merit ordering guarantees fairness and completeness.

We represent a parse as the chain of the context-free rules used in a leftmost reduction of the input sentence, whose non-terminals are items. The output grammar is then represented in finite shared form by an AND-OR graph that in our case is precisely the shared-forest [5]. Formally, given $\tau = \delta(p, X, a) \ni (q, Y, u)$, the transitional formalism works as follows:

1. $\tilde{\delta}([p, X, S_j^w, S_i^w], a) \ni ([q, \varepsilon, S_i^w, S_i^w], \varepsilon)$
2. $\tilde{\delta}([p, X, S_j^w, S_i^w], a) \ni ([p, Y, S_i^w, S_{i+1}^w], a)$
3. $\tilde{\delta}([p, X, S_j^w, S_i^w], a) \ni (I_1, I_1 \to I_2)$
4. $\tilde{\delta}([p, \varepsilon, S_j^w, S_i^w], a) \ni \tilde{\tau}_d$

if and only if

$$1.\ Y = X, \quad 2.\ Y = a, \quad 3.\ Y \in N,$$
$$4.\ Y = \varepsilon,\ \forall q \in \mathcal{Q} \text{ such that } \exists\, \delta(q, X, \varepsilon) \ni (p, X, \varepsilon)$$

respectively, where we have considered:

$$\tilde{\tau}_d = \tilde{\delta}_d([q, \varepsilon, S_l^w, S_j^w], a) \ni ([q, \varepsilon, S_l^w, S_i^w], I_3 \to I_4 I_5)$$

$$I_1 = [p, Y, S_i^w, S_i^w], \quad I_2 = [p, X, S_j^w, S_i^w], \quad I_3 = [q, \varepsilon, S_l^w, S_i^w]$$
$$I_4 = [q, X, S_l^w, S_j^w], \quad I_5 = [p, \varepsilon, S_j^w, S_i^w]$$

and

$$\begin{cases} \tilde{\delta} : \mathrm{It} \times \Sigma \cup \{\varepsilon\} \longrightarrow \{\mathrm{It} \cup \tilde{\delta}_d\} \times \Pi^* \\ \tilde{\delta}_d : \mathrm{It} \times \Sigma \cup \{\varepsilon\} \longrightarrow \mathrm{It} \end{cases}$$

where *It* is the set of parse items and $\tilde{\delta}_d$ is 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 PDT.
2. A push of $a \in \Sigma$ 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 state p under transition X in the PDT. 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 not only to the configuration resulting of the first one, but also on those to be generated and sharing the same syntactic structure.

The time complexity (resp. space complexity) is, in the worst case, $\mathcal{O}(n^3)$ (resp. $\mathcal{O}(n^2)$) for inputs $w_{1..n}$. This complexity is linear for deterministic inputs, which favours the performances, since ambiguities often have a local behavior.

3 The editing distance

Given P, a pattern tree, and D, a data tree, we define an *edit operation* as a pair $a \to b,\ a \in \mathrm{labels}(P) \cup \{\varepsilon\},\ b \in \mathrm{labels}(D)\ \cup\ \{\varepsilon\},\ (a,b) \neq (\varepsilon,\varepsilon)$, where ε represents the empty string. We can delete a node $(a \to \varepsilon)$, insert a node $(\varepsilon \to b)$, and change a node $(a \to b)$. Each edit operation has an associated cost, $\gamma(a \to b)$, 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 P and D is defined by the metric:

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

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

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

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,P,D) is computed from relabeling, deleting and inserting operations, as follows:

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

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

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

which allows us to focus on edit sequences being a mapping. We show in Fig. 2 one example of mapping between two trees, and a sequence of edit operations not constituting a mapping. 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. We shall proceed through the nodes determining mappings from all leaf *r_keyroots* first, then all *r_keyroots* at the next higher level, and so on to the root. The set of *r_keyroots*(T) is indicated by arrows in Fig. 1.

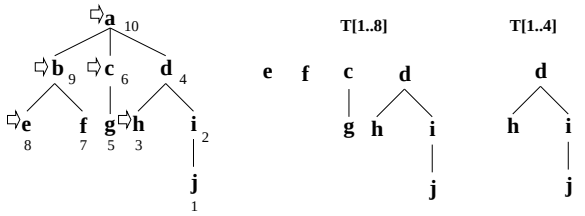


Figure 1: The forest distance using an inverse postorder numbering

In dealing with approximate VLDC pattern matching, some structural details can be omitted in the target tree and different strategies are then applicable. Following Zhang *et al.* in [7] we introduce two different definitions to VLDC matching:

- The VLDC substitutes for part of a path from the root to a leaf of the data tree. We represent such a substitution, shown in Fig. 2, by a vertical bar "|", and call it a *path*-VLDC.
- The VLDC matches part of such a path and all the subtrees emanating from the nodes of that path, except possibly at the lowest node of that path. At the lowest node, the VLDC symbol can substitute for a set of leftmost subtrees and a set of rightmost subtrees. We call this an *umbrella*-VLDC, and represent it by a circumflex "$\wedge$", such it is shown in Fig. 2.

To formalize the consideration of pattern trees with VLDC's requires the capture of the notion of VLDC-*substitution* for nodes in the target tree P labeled | or $\wedge$ as previously introduced. So, given a data tree D and a substitution s on P, we redefine:

$$\delta(P, D) = \min_{s \in \mathcal{S}}\{\delta(P, D, s)\}$$

where $\mathcal{S}$ is the set of all possible VLDC-substitutions, and $\delta(P, D, s)$ is the distance $\delta(\bar{P}, D)$, being $\bar{P}$ the result of apply the substitution s to P. As a consequence, no cost is induced by VLDC-substitutions.

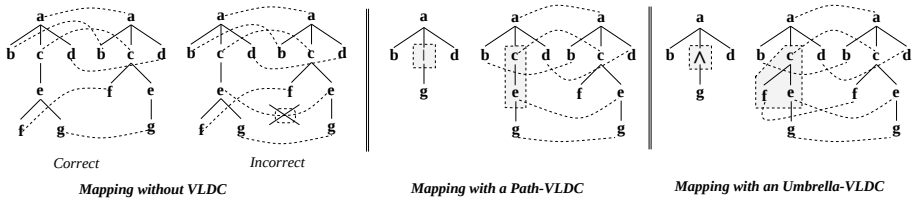


Figure 2: An example on mappings

4 Approximate VLDC tree matching

The major question of Zhang *et al.* in [7], is the tree distance algorithm itself. However, parsing and tree-to-tree correction are topologically related and, to get the best performance, it is necessary to understand the mechanisms that cause the phenomenon of tree duplication.

A major factor to take into account is the syntactic representation used. We choice to work in the parsing context described for ICE [5]. 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 appearance, since context-free grammars can be represented by AND-OR graphs that in our case are precisely the shared-forest graph [1]. 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. This allows us to gain in sharing efficiency in relation to classic representations, such it is shown in Fig. 3.

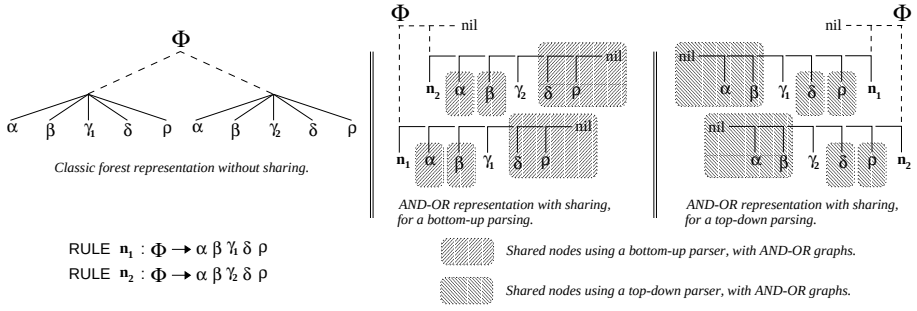


Figure 3: How shared forest are built using an AND-OR formalism

It is also important the parsing scheme applied. So, 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 also shown in Fig. 3.

At this level, one major observation we noted is that Zhang *et al.* 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, we introduce $r(i)$ (resp. $\mathrm{anc}(i)$) as the rightmost leaf descendent 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, as it is shown in Fig. 1. In particular, we have $T[r(i)..i]$ is the tree rooted at $T[i]$. We now define the *forest edition distance* between a target tree P and a data tree D, as a generalization of δ, in the form

$$\mathrm{f\_d}(P[s_1..s_2], D[t_1..t_2]) = \delta(P[s_1..s_2], D[t_1..t_2])$$

that we shall denote $\mathrm{f\_d}(s_1..s_2, t_1..t_2)$ when the context is clear. Intuitively, this concept computes the distance between two nodes, $P[s_2]$ and $D[t_2]$, in the context of their left siblings in the corresponding trees, while the tree distance, $\delta(P[s_2], D[t_2])$, is computed only from their descendants.

To be precise, given a pattern tree P and a data tree D, we can compute the editing distance $\mathrm{t\_d}(P, D)$ applying the formulae that follow [4], for nodes $i \in \mathrm{anc}(s)$ and $j \in \mathrm{anc}(t)$, assuming $P[s]$ is not an incomplete structure:

$$\mathrm{f\_d}(r(i)..s, r(j)..t) = \begin{cases} \min \left\{ \begin{array}{lcl} \mathrm{f\_d}(r(i)..s-1, r(j)..t) & + & \gamma(P[s] \to \varepsilon), \\ \mathrm{f\_d}(r(i)..s, r(j)..t-1) & + & \gamma(\varepsilon \to D[t]), \\ \mathrm{f\_d}(r(i)..s-1, & & \\ \quad r(j)..t-1) & + & \gamma(P[s] \to D[t]) \end{array} \right\} \\ \textbf{iff } r(s) = r(i) \textbf{ and } r(t) = r(j) \\ \min \left\{ \begin{array}{lcl} \mathrm{f\_d}(r(i)..s-1, r(j)..t) & + & \gamma(P[s] \to \varepsilon), \\ \mathrm{f\_d}(r(i)..s, r(j)..t-1) & + & \gamma(\varepsilon \to D[t]), \\ \mathrm{f\_d}(r(i)..r(s)-1, r(j)..r(t)-1) & + & \mathrm{t\_d}(s,t) \end{array} \right\} \\ \textbf{otherwise} \end{cases}$$

When $P[s]$ is either "|" or "$\wedge$" formulae must be adapted, we first assume $P[s]$ is "|":

$$\mathrm{f\_d}(r(i)..s, r(j)..t) = \min \left\{ \begin{array}{lcl} \mathrm{f\_d}(r(i)..s-1,\ r(j)..t) & + & \gamma(P[s] \to \varepsilon), \\ \mathrm{f\_d}(r(i)..s,\ r(j)..t-1) & + & \gamma(\varepsilon \to D[t]), \\ \mathrm{f\_d}(r(i)..s-1,\ r(j)..t-1) & + & \gamma(P[s] \to D[t]), \\ \mathrm{f\_d}(\phi, D[r(j)]..t-1) & + & \min_{t_k}\{\mathrm{t\_d}(s,t_k) \\ & - & \mathrm{t\_d}(\phi,t_k)\}, \\ & & 1 \le k \le n_t \end{array} \right\}$$

For the case where $P[s]$ is "$\wedge$", formulae are the following:

$$\mathrm{f\_d}(r(i)..s, r(j)..t) = \min \left\{ \begin{array}{lcl} \mathrm{f\_d}(r(i)..s-1,\ r(j)..t) & + & \gamma(P[s] \to \varepsilon), \\ \mathrm{f\_d}(r(i)..s,\ r(j)..t-1) & + & \gamma(\varepsilon \to D[t]), \\ \mathrm{f\_d}(r(i)..s-1,\ r(j)..t-1) & + & \gamma(P[s] \to D[t]), \\ \min_{t_k}\{\mathrm{t\_d}(s,t_k)\}, & & 1 \le k \le n_t, \\ \min_{t_k}\{\mathrm{s\_f\_d}(r(i)..s-1, r(j)..t_k)\}, & & 1 \le k \le n_t \end{array} \right\}$$

where $D[t_k]$, $1 \le k \le n_t$, are children of $D[t]$. If $D[t]$ is a leaf, that is $t = r(j)$, then only the first three expressions are present. We define the *suffix forest distance*

between F_P and F_D, forests in the pattern P and the data tree D respectively, denoted $\text{s\_f\_d}(F_P, F_D)$, as the distance between F_P and $\bar{F}_D$, where $\bar{F}_D$ is a sub-forest of F_D with some consecutive complete subtrees removed from the left all having the same parent. Formally we have that

$$\text{s\_f\_d}(F_P, F_D) = \min_{\bar{F}_D}\{\text{f\_d}(F_P, \bar{F}_D)\}$$

From a computational point of view, it can be proved that

$$\text{s\_f\_d}(r(i)..s, r(j)..t) = \begin{cases} \min\left\{ \begin{array}{l} \text{f\_d}(r(i)..s,\ \phi), \\ \text{f\_d}(r(i)..s,\ r(j)..t) \end{array} \right\} \\ \textbf{iff}\ r(t) = r(j) \\ \\ \min\left\{ \begin{array}{lcl} \text{s\_f\_d}(r(i)..s-1,\ r(j)..t) & + & \gamma(P[s] \to \varepsilon), \\ \text{s\_f\_d}(r(i)..s,\ r(j)..t-1) & + & \gamma(\varepsilon \to D[\hat{t}]), \\ \text{s\_f\_d}(r(i)..r(s)-1,\ r(j)..r(t)-1) & + & \text{t\_d}(s,t) \end{array} \right\} \\ \textbf{otherwise} \end{cases}$$

To compute $t\_d(P, D)$ it will be sufficient to take into account that

$$\text{t\_d}(P, D) = \text{f\_d}(\text{root}(P)..r(\text{root}(P)), \text{root}(D)..r(\text{root}(D)))$$

Time bound is $\mathcal{O}(\mid P \mid\mid D \mid \min(\text{depth}(P), \text{leaves}(P))\min(\text{depth}(D), \text{leaves}(D)))$ in the worst case, where $\mid P \mid$ (resp. $\mid D \mid$) is the number of nodes in the pattern tree P (resp. in the data tree D).

5 Approximate VLDC matching in shared forest

To start with, let P be a labeled ordered tree where some structural details has been omitted, and D an AND-OR graph, both of them built using our parsing frame. We shall identify P with a query and D with a part of the syntactic representation for a textual database with a certain degree of ambiguity. The presence of OR nodes in D 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 $P[s]$ be the current node in the inverse postorder for P, and $i \in \text{anc}(s)$ a *r_keyroot*. Given an OR node $D[k]$ we can distinguish two situations, depending on the situation of this OR node and the situation of the *r_keyroots* of D.

5.1 Sharing into a same *r_keyroot*

Let $D[t']$ and $D[t'']$ be the nodes we are dealing with for two branches labeled $D[k']$ and $D[k'']$ of the OR node $r(D[k])$. We have that $j \in anc(t') \cap anc(t'')$, that is, the tree rooted at the *r_keyroot* $D[j]$ includes the OR alternatives $D[k']$ and $D[k'']$.

This situation is shown in Fig. 4 using a classic representation and the AND-OR graphs. Here, the lightly shaded part refers to nodes whose distance have been computed in the inverse postorder before the OR node $D[k]$. 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 $D[r(t')-1]$ and $D[r(t'')-1]$ are the same, that is, their corresponding subtrees are shared. So, $D[r(t')]$ (resp. $D[r(t'')]$) is the following node in $D[k']$ (resp. $D[k'']$) to deal with once the distance for the shared structure has been computed.

At this point, our aim is to compute the value for $f\_d(r(i)..s,\ r(j)..\hat{t}),\ \hat{t} \in \{t',\ t''\}$, proving that we can translate parse sharing on sharing on computations for these distances.

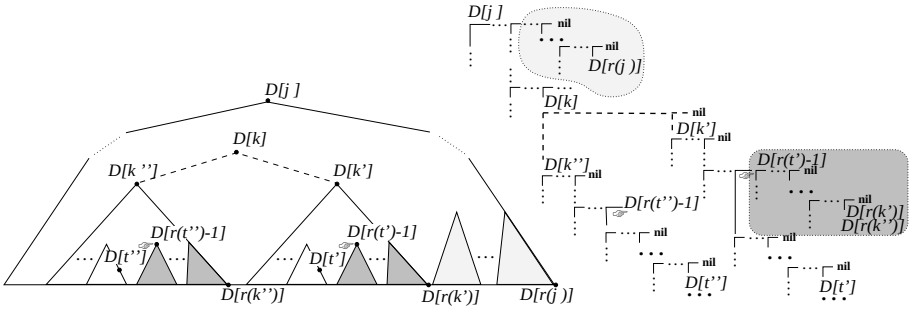


Figure 4: Sharing into a same *r_keyroot*

Since we have assumed there is a shared structure between $D[r(\hat{t})]$ and $D[r(j)]$, we conclude that $r(j) \neq r(\hat{t})$ and the values for $f\_d(r(i)..s,\ r(j)..\hat{t}),\ \hat{t} \in \{t',\ t''\}$ are given by:

$$\mathrm{f\_d}(r(i)..s, r(j)..\hat{t}) = \min \left\{ \begin{array}{lcl} \mathrm{f\_d}(r(i)..s-1,\ r(j)..\hat{t}) & + & \gamma(P[s] \rightarrow \varepsilon), \\ \mathrm{f\_d}(r(i)..s,\ r(j)..\hat{t}-1) & + & \gamma(\varepsilon \rightarrow D[\hat{t}]), \\ \mathrm{f\_d}(r(i)..r(s)-1,\ r(j)..r(\hat{t})-1) & + & \mathrm{t\_d}(s, \hat{t}) \end{array} \right\}$$

where $\hat{t} \in \{t',\ t''\}$. We can interpret this three alternatives as follows:

1. The values for $f\_d(r(i)..s-1,\ r(j)..\hat{t}),\ \hat{t} \in \{t',\ t''\}$ 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 $D[\hat{t}],\ \hat{t} \in \{t',\ t''\}$, these are:

   - If both nodes are leaves, then $r(\hat{t}) = \hat{t}$. As a consequence, we also have that $D[t'-1] = D[r(t')-1] = D[r(t'')-1] = D[t''-1]$, and the values $f\_d(r(i)..s,\ r(j)..\hat{t}-1),\ \hat{t} \in \{t',\ t''\}$ are also the same.
   - Otherwise, following the inverse postorder, we would arrive at the rightmost leaves of $D[t']$ and $D[t'']$, where we could apply the reasoning considered in the previous case.

3. Values for the distances $f\_d(r(s)..r(i)-1,\ r(j)..r(\hat{t})-1),\ \hat{t} \in \{t',\ t''\}$ are identical, given that nodes $D[r(\hat{t})-1],\ \hat{t} \in \{t',\ t''\}$ are shared by the parser.

A similar reasoning can be applied to compute the values for s_f_d, avoiding redundant computations.

5.2 Sharing between different *r_keyroots*

We have that $j' \in anc(t')$ and $j'' \in anc(t'')$, with $j' \neq j''$, are two *r_keyroots*. We also have an OR node $D[k]$ 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*, $D[j']$ (resp. $D[j'']$), in the branch labeled $D[k']$ (resp. $D[k'']$).

Our aim is to compute the value for distances $f\_d(r(i)..s,\ r(\hat{\jmath})..\hat{t})$, where pairs $(\hat{\jmath}, \hat{t})$ are in $\{(j',\ t'),\ (j'',t'')\}$. Formally, we have that these values are given by:

$$
\mathrm{f\_d}(r(i)..s, r(\hat{\jmath})..\hat{t}) = \left\{ \begin{array}{l}
\left\{ \begin{array}{l}
\min \left\{ \begin{array}{lcl}
\mathrm{f\_d}(r(i)..s-1,\ r(\hat{\jmath})..\hat{t}) & + & \gamma(P[s] \rightarrow \varepsilon), \\
\mathrm{f\_d}(r(i)..s,\ r(\hat{\jmath})..\hat{t}-1) & + & \gamma(\varepsilon \rightarrow D[\hat{t}]), \\
\mathrm{f\_d}(r(i)..s-1,\ r(\hat{\jmath})..\hat{t}-1) & + & \gamma(P[s] \rightarrow D[\hat{t}]), \\
\mathrm{f\_d}(\phi, D[r(\hat{\jmath})]..\hat{t}-1) & + & \min_{\hat{t}_k}\{\mathrm{t\_d}(s,\hat{t}_k) \\
 & - & \mathrm{t\_d}(\phi,\hat{t}_k)\}, \\
 & & 1 \leq k \leq n_{\hat{t}}
\end{array} \right\} \\
\textbf{iff } P[s] = | \\
\\
\min \left\{ \begin{array}{lcl}
\mathrm{f\_d}(r(i)..s-1,\ r(\hat{\jmath})..\hat{t}) & + & \gamma(P[s] \rightarrow \varepsilon), \\
\mathrm{f\_d}(r(i)..s,\ r(\hat{\jmath})..\hat{t}-1) & + & \gamma(\varepsilon \rightarrow D[\hat{t}]), \\
\mathrm{f\_d}(r(i)..s-1,\ r(\hat{\jmath})..\hat{t}-1) & + & \gamma(P[s] \rightarrow D[\hat{t}]), \\
\min_{\hat{t}_k}\{\mathrm{t\_d}(s,\hat{t}_k), & & 1 \leq k \leq n_{\hat{t}}, \\
\min_{\hat{t}_k}\{\mathrm{s\_f\_d}(r(i)..s-1, r(\hat{\jmath})..\hat{t}_k), & & 1 \leq k \leq n_{\hat{t}}
\end{array} \right\} \\
\textbf{iff } P[s] = \wedge \\
\\
\min \left\{ \begin{array}{lcl}
\mathrm{f\_d}(r(i)..s-1,\ r(\hat{\jmath})..\hat{t}) & + & \gamma(P[s] \rightarrow \varepsilon), \\
\mathrm{f\_d}(r(i)..s,\ r(\hat{\jmath})..\hat{t}-1) & + & \gamma(\varepsilon \rightarrow D[\hat{t}]), \\
\mathrm{f\_d}(r(i)..s-1,\ r(\hat{\jmath})..\hat{t}-1) & + & \gamma(P[s] \rightarrow D[\hat{t}])
\end{array} \right\} \\
\textbf{otherwise}
\end{array} \right. \\
\textbf{iff } r(s) = r(i) \textbf{ and } r(\hat{t}) = r(\hat{\jmath}) \\
\\
\min \left\{ \begin{array}{lcl}
\mathrm{f\_d}(r(i)..s-1,\ r(\hat{\jmath})..\hat{t}) & + & \gamma(P[s] \rightarrow \varepsilon), \\
\mathrm{f\_d}(r(i)..s,\ r(\hat{\jmath})..\hat{t}-1) & + & \gamma(\varepsilon \rightarrow D[\hat{t}]), \\
\mathrm{f\_d}(r(i)..r(s)-1,\ r(\hat{\jmath})..r(\hat{t})-1) & + & \mathrm{t\_d}(s,\hat{t})
\end{array} \right\} \\
\textbf{otherwise}
\end{array} \right.
$$

The situation, shown in the first case of Fig. 5, makes possible $r(s) = r(i)$ and $r(\hat{t}) = r(\hat{\jmath})$. In this first case, we can assume that a tail of sons is shared by nodes $D[\hat{t}],\ \hat{t} \in \{t',\ t''\}$. We can also assume that this tail is proper given that, otherwise, our parser guarantees that the nodes $D[\hat{t}],\ \hat{t} \in \{t',\ t''\}$ are also shared. Taking into account our parsing strategy, we conclude that the structural 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 the first case of Fig. 5.

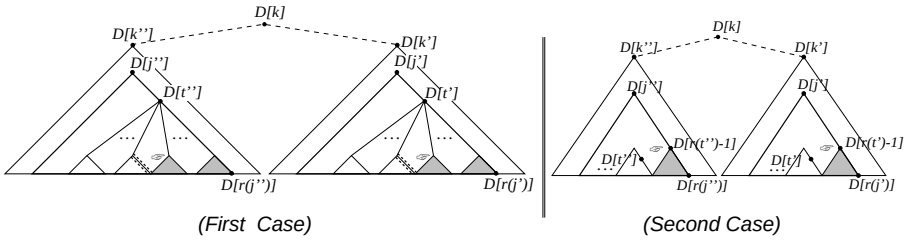


Figure 5: Sharing between different *r_keyroots*

We consider now the second case, which is shown in the right-hand side of Fig. 5. This case is similar to the situation showed in the previous section, sharing into a same *r_keyroot*. Here we have again three alternative values to compute the minimum, and we can make similar considerations about how the parser shared structures affect the computation for the distances.

6 A practical example

In order to illustrate our previous work, we return to the discussion about parse sharing into a same *r_keyroot*. We shall consider a simple example: the language of arithmetical expressions. We consider a non-deterministic grammar, $\mathcal{G}_N$, generating the language; and the input sentence $a_1 + a_2 + a_3 + a_4$, from which we shall build the parse shared data forest. We show in Fig. 6 a part of this parse shared data forest and the pattern tree used in this example.

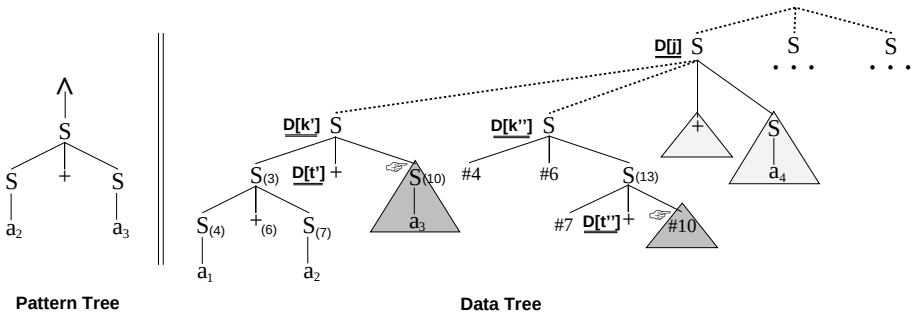
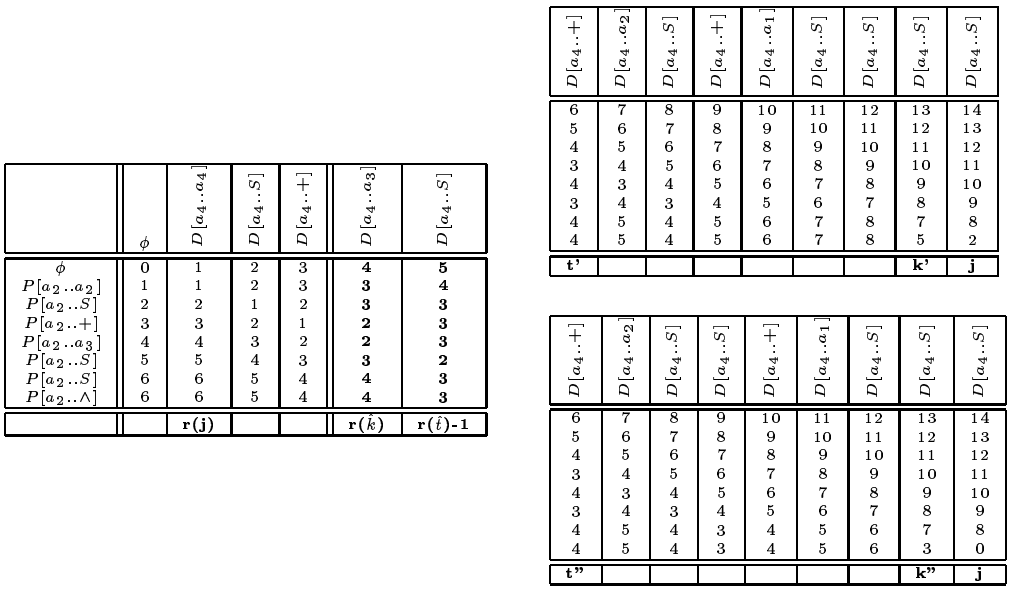


Figure 6: An example on sharing into a same r_keyroot

As in the general case, $D[t']$ and $D[t'']$ are the nodes we are dealing with. Lightly shaded parts refer to nodes whose distance have been computed in the inverse postorder before the OR node $D[k]$, that is, the forest $D[r(j)..r(k)-1]$. The heavily shaded parts represent a shared structure, which corresponds to the two forests $D[r(\hat{k})..r(\hat{[t]})-1]$ with $\hat{k} \in \{k', k''\}$ and $\hat{t} \in \{t', t''\}$.

In relation with the concrete computation of the editing distance, Fig. 7 shows a set of tables with the values of the induced forest distances for the *r_keyroot* j using our pattern tree. These distances have been computed using a discrete metric for the cost associated to the edit operations: the delete and insert operations cost is one unit; the change operation cost is one if the two labels are different, and zero otherwise.

Each row in the tables corresponds with each of the induced subforests that can be built in the pattern tree starting in the node "a_2". The same explanation is applicable to the columns, there is one column for each induced subforest in the data tree starting in "a_4". In the leftmost table we show the values for the common part of the computations. This table has two parts, the first storing the distances for the common forest prior to the OR-node. The second one, in boldface, shows the columns of values associated to the shared structure. The other two tables in the right-hand side of the figure store the distances computed for the nodes in the two alternatives of the OR-node following the shared subtree.



	ϕ	$D[a_4..a_4]$	$D[a_4..S]$	$D[a_4..+]$	$D[a_4..a_3]$	$D[a_4..S]$
ϕ	0	1	2	3	**4**	**5**
$P[a_2..a_2]$	1	1	2	3	**3**	**4**
$P[a_2..S]$	2	2	1	2	**3**	**3**
$P[a_2..+]$	3	3	2	1	**2**	**3**
$P[a_2..a_3]$	4	4	3	2	**2**	**3**
$P[a_2..S]$	5	5	4	3	**3**	**2**
$P[a_2..S]$	6	6	5	4	**4**	**3**
$P[a_2..\wedge]$	6	6	5	4	**4**	**3**
		r(j)			**r($\hat{k}$)**	**r($\hat{t}$)-1**

$D[a_4..+]$	$D[a_4..a_2]$	$D[a_4..S]$	$D[a_4..+]$	$D[a_4..a_1]$	$D[a_4..S]$	$D[a_4..S]$	$D[a_4..S]$	$D[a_4..S]$
6	7	8	9	10	11	12	13	14
5	6	7	8	9	10	11	12	13
4	5	6	7	8	9	10	11	12
3	4	5	6	7	8	9	10	11
4	3	4	5	6	7	8	9	10
3	4	3	4	5	6	7	8	9
4	5	4	5	6	7	8	7	8
4	5	4	5	6	7	8	5	2
t'							**k'**	**j**

$D[a_4..+]$	$D[a_4..a_2]$	$D[a_4..S]$	$D[a_4..S]$	$D[a_4..+]$	$D[a_4..a_1]$	$D[a_4..S]$	$D[a_4..S]$	$D[a_4..S]$
6	7	8	9	10	11	12	13	14
5	6	7	8	9	10	11	12	13
4	5	6	7	8	9	10	11	12
3	4	5	6	7	8	9	10	11
4	3	4	5	6	7	8	9	10
3	4	3	4	5	6	7	8	9
4	5	4	3	4	5	6	7	8
4	5	4	3	4	5	6	3	0
t"							**k"**	**j**

Figure 7: Forest distances matrix for the example *r_keyroot*

We center our example in the computations when we are dealing with $P[s] = a_3$, that is, the index s points to the node labeled "a_3" in our pattern tree P. We are computing distances for the induced subforest starting at the node labeled "a_2" and ending in the node labeled "a_3", which corresponds to the distances stored in the row $P[a_2..a_3]$. Following the explanation in a previous section about sharing into a same *r_keyroot*, we need three values to compute the distances $f\_d(r(i)..s,\ r(j)..\hat{t})$. These are:

1. The value $f\_d(r(i)..s-1,\ r(j)..\hat{t})$. These distances have been computed in a previous step of the algorithm, when we were dealing with the node $P[s-1]$ in the pattern tree. In our example of Fig. 7, this means taking the values stored in the row $P[a_2..+]$ of the columns labeled t' and t''. For t' we must take the distance value 3 and add to it the cost $\gamma(\text{"}a_3\text{"} \rightarrow \varepsilon) = 1$. For t'' we add $\gamma(\text{"}a_3\text{"} \rightarrow \varepsilon) = 1$ to the value 3.

2. The value $f\_d(r(i)..s,\ r(j)..\hat{t}-1)$. In our example, since t' and t'' are both leaves, we have that $r(\hat{t}) = \hat{t}$, and those two distances correspond to values associated to the last node in the shared structure. In Fig. 7, this means taking the value from the row $P[a_2..a3]$ of the column labeled $r(\hat{t}-1)$, that is, from

the last column of the sub-matrix associated to the shared structure. We take the value 3 from the row $P[a_2..a3]$ from this column, for t'. We add to it the cost $\gamma(\varepsilon \rightarrow \text{“+”}) = 1$ and for t'' we add $\gamma(\varepsilon \rightarrow \text{“+”}) = 1$.

3. The value $f\_d(r(i)..r(s)-1,\ r(j)..r(\hat{t})-1)$. In this case, the indexes $r(t')-1$ and $r(t'')-1$ refer to the same node, which is the root of the shared structure. So, in our example we have that the index $r(s)-1$ points to the node labeled “+” in the pattern tree. This means that we have to take the distances stored in the $P[a_2..+]$ row of the column labeled $r(\hat{t})-1$ in Fig. 7.

   The distances $t\_d(s,\hat{t})$ have been computed in a previou step and are stored in the t_d matrix; for t' this value is 1 and for t'' it is 1. We add each one of those values with the forest distance strored in the $P[a_2..+]$ row of the column labeled $r(\hat{t})-1$, that is 3.

Finally, following the algorithm, we must obtain the minimum of the three values computed for t' and t'', getting that $f\_d(r(i)..s,\ r(j)..t') = 4$ and $f\_d(r(i)..s,\ r(j)..t'') = 4$, and storing these values in the corresponding forest distance tables.

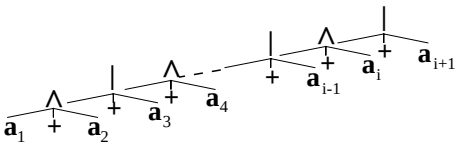
7 Experimental results

We use the language of arithmetical expressions considering the non-deterministic grammar, $\mathcal{G}_{\mathcal{N}}$, and two deterministic grammars, $\mathcal{G}_L$ and $\mathcal{G}_R$, representing respectively the left and right associative versions for the arithmetic operators. With the non-deterministic grammar we show the gain in efficiency due to the use of sharing structures. And the deterministic grammars will be used to compare our proposal with Zhang *et al.* [7].

To simplify the explanation, we focus on matching phenomena assuming that parsers are built using ICE [5]. 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. These programs have a number of ambiguous parses which grows exponentially with i. This number is:

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

As pattern, we have used deterministic parse trees of the form



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, the rightmost diagram in Fig. 8 proves the adaptation of our proposal (resp. Zhang *et al.* algorithm) to left-recursive (resp. right-recursive) derivations, which corroborates our conclusions.

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, such as is shown in the leftmost diagram of Fig. 8.

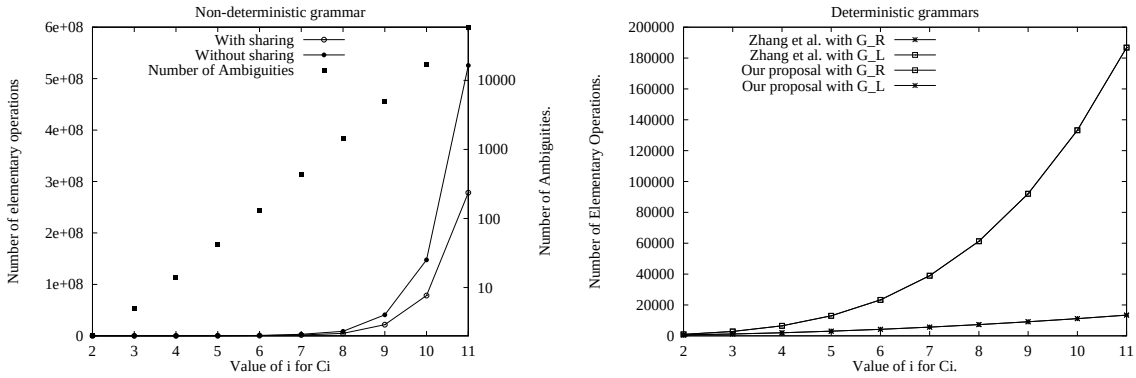


Figure 8: Results on approximate VLDC matching

8 Conclusions

Any practical approach to IR is based on two assumptions: a common representation for documents and requests, and simplifying assumptions for the retrieval functions. Approximate VLDC pattern matching enables one to extract information from complicated sentences in the database, and experimental results seem to corroborate the practical interest.

References

[1] S. Billot and B. Lang. The structure of shared forest in ambiguous parsing. In *Proc. 27th Annual Meeting of the ACL*, 1989.

[2] P. Kilpelainen and H. Mannila. Grammatical tree matching. *Lecture Notes in Computer Science*, 644:159–171, 1992.

[3] K.-C. Tai. Syntactic error correction in programming languages. *IEEE Transactions on Software Engineering*, SE-4(5):414–425, 1978.

[4] M. Vilares, D. Cabrero, and F. Ribadas. Approximate matching in shared forest. In S. F. . P. Sabatier, editor, *Sixth Int. Workshop on Natural Language Understanding and Logic Programming*, pages 59–72, Las Cruces, New Mexico, 1999.

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

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

[7] K. Zhang, D. Shasha, and J. T. L. Wang. Approximate tree matching in the presence of variable length don't cares. *Journal of Algorithms*, 16(1):33–66, Jan. 1994.