On Pattern-Matching as Query Facility

M. Vilares[1], F.J. Ribadas[1], and J. Graña[2]

[1] Department of Informatics, University of Vigo
Campus As Lagoas s/n, 32004 Ourense, Spain
http://grupocole.org
[2] Department of Computer Science, University of A Coruña
Campus de Elviña s/n, 15071 A Coruña, Spain
{vilares,grana}@udc.es, ribadas@uvigo.es

Abstract. Pattern-matching can be used to develop query languages for information retrieval/extraction tasks. The operational basis is simple, since documents and queries can be represented using the same kind of structure, documents or fragments that satisfy the query and which would be those that are closest to its structural representation. In this sense, pattern-matching makes it possible to combine a variety of structural constraints in flexible ways, allowing the query to be defined approximately, or even omitting some structural details.
Our goal is to analyze how the lack of structural information in queries may degrade performance, and to illustrate the relation between the computational work and the effort applied by the user to describe these queries.

1 Introduction

Representing documents and queries as structures is a natural way to introduce pattern-matching as an operational model for a query language [2]. In relation to other approaches, this one provides the flexibility to access different views of the database, since it is not always evident how to do this using techniques based on classic indexing methods. This capacity can be used to adapt a formal query model to practical user queries, in order to design query languages that are closer to natural ones, where the level of precision is not easy, or even impossible, to define.

In the context considered, pattern-matching can be studied from two different points of view: comparing structures that can only be approximately defined [7], or introducing *variable length don't care* (VLDC) technology in order to omit structural details [8]. In the first case, pattern-matching can be applied to deal with queries that can only be approximately defined, which often occurs. In the second case, the technique can be applied to deal with lack of information in queries, either because it is unavailable to the user, or simply because the user wants to reduce his own workload. Both strategies can correspond to complementary phases in the definition of a query considering interactive expansion [1] and, from the point of view of the user, they make it possible to

control the level of detail in the retrieval/extraction process [6]. Whatever the choice, it impacts on both system performance and implementation techniques.

A subject of additional interest is the exploitation of sharing between target structures with common nodes, which may lead to an increase in computational efficiency in approximate pattern-matching [4, 5]. In effect, although in the case of the query language ambiguity could probably be eliminated, imprecision in the language intended to represent the document produces ambiguity. Since it is desirable to consider all possible interpretations for semantic processing, it is convenient to merge structures as much as possible, sharing common parts. This could be applied to a variety of problems, such as natural language processing, where ambiguity and structural sharing are common [3]. Another case is molecular evolution, with many examples indicating that gene duplication with fusion has occurred extensively in the past. This provides an as yet unexplored route to the evolution of new functions from existing proteins.

Our aim in this paper is to look for both theoretical foundations and practical constraints in the existing relationship between structural and computational complexity in dealing with query languages based on pattern-matching, and to focus on the treatment of structural sharing. In this way, we hope to bring to light some of the factors governing the mechanisms behind the practice, and their impact on costs from the user's choices in query processing.

2 The editing distance

Given P, a pattern tree, and D, a data tree, we define an *edit operation* as a pair $a \to b$, $a \in \text{labels}(P) \cup \{\varepsilon\}$, $b \in \text{labels}(D) \cup \{\varepsilon\}$, $(a, b) \neq (\varepsilon, \varepsilon)$, 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 a 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 is 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 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 which are a mapping. We show in Fig. 2 one example of mapping between two trees, and a sequence of edit operations which do not constitute 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 proceed through the nodes, first determining mappings from all leaf r_keyroots, 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. In dealing with approximate VLDC pattern-matching, different strategies are then applicable. Following Zhang *et al.* in [8], we introduce two different definitions for VLDC matching:

- The VLDC substitutes 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 a set of leftmost subtrees and a set of rightmost subtrees. We call this an *umbrella*-VLDC, and represent it by a circumflex "∧", as shown in Fig. 2.

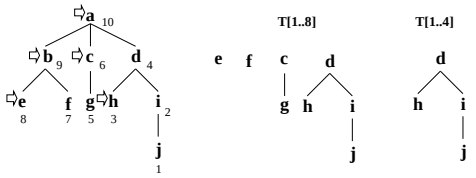

Fig. 1. The forest distance using an inverse postorder numbering

We now capture the use of VLDC symbols. 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 applying the substitution s to P.

3 Pattern-matching and parsing

Parsing and tree-to-tree correction are related and we need to understand the mechanisms that lead to the tree duplication in order to gain efficiency.

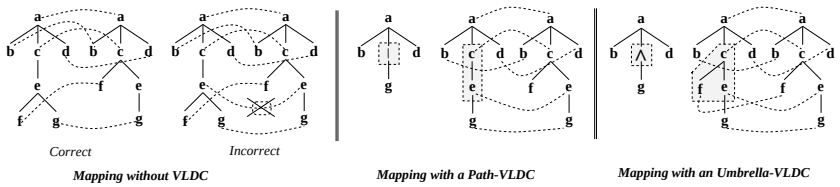

Fig. 2. Examples of mappings

3.1 Factors of interaction

The first factor is the syntactic representation used. We represent a parse in finite shared form as the chain of the context-free rules used in a leftmost reduction of the input sentence [3]. The resulting grammar is equivalent to an AND-OR graph, whose AND-nodes are the usual parse-tree nodes, while OR-nodes are ambiguities. Sharing of structures is represented by nodes accessed by more than one other node and it may correspond to sharing of a complete tree, but also to sharing of a part of the descendants of a given node, as shown in Fig. 3.

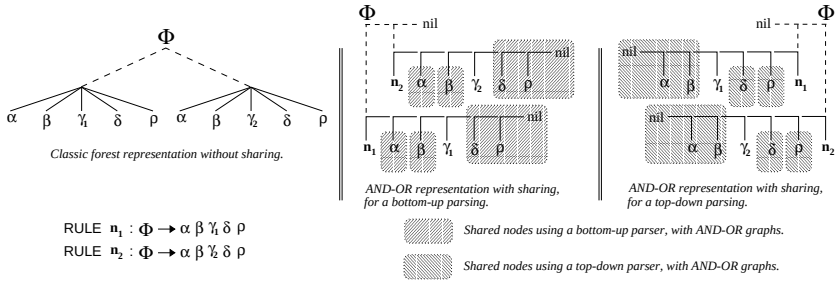

Fig. 3. How shared forests are built using an AND-OR formalism

Another factor is 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 depends on 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 also shown in Fig. 3. Here, one major observation we noted is that Zhang *et al.* consider a postorder traversal, computing the forest distance by left-recursion on this search. So, we would need to consider a top-down parser to avoid redundant computations. However, these parsers are not computationally efficient, and a bottom-up approach requires a rightmost search of tree constituents. This implies redefining the original finding strategy.

3.2 The forest edition distance

We introduce $r(i)$ (resp. 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 is shown in Fig. 1. In particular, we have $T[r(i)..i]$ as 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

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

that we denote $\text{f\_d}(s_1..s_2, t_1..t_2)$ 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, we can compute the editing distance $\text{t\_d}(P, D)$ applying the formulae that follow, for nodes $i \in \text{anc}(s)$ and $j \in \text{anc}(t)$, assuming $P[s]$ is not an incomplete structure:

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

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

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

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

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

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, as $\text{s\_f\_d}(F_P, F_D) = \min_{\bar{F}_D}\{\text{f\_d}(F_P, \bar{F}_D)\}$, where $\bar{F}_D$ is a sub-forest

of F_D with some consecutive complete subtrees removed from the left, all of them having the same parent. From a computational point of view, it can be proved that

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

To compute $t\_d(P, D)$ it is 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 $\mid T \mid$ is the number of nodes in a tree T. We talk about *elementary operations* to refer to each one of these minimum values computed.

4 Pattern-matching and shared forest

Let P be a labeled ordered tree where some structural details have been omitted, and D an AND-OR graph. We identify P with a query and D with a part of the syntactic representation for a database with a certain degree of ambiguity. 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.

4.1 Sharing into a same r_keyroot

Let $D[t']$ and $D[t'']$ be the nodes we are dealing with in parallel 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 $D[k'']$. Such a situation is shown in Fig. 4. Here, the lightly shaded part refers to nodes whose distance has been computed in the inverse postorder before the OR-node $D[k]$. The heavily shaded part represents a shared structure. The notation "• • •" expresses the fact that we descend along the rightmost branch of the corresponding tree.

We 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. 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 to sharing in computations for these distances. 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:

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

where $\hat{t} \in \{t',\ t''\}$. We can interpret these 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 no consequences for the natural computation of the distances.
2. Two cases are possible in relation to the nature of nodes $D[\hat{t}],\ \hat{t} \in \{t',\ t''\}$:
   - If both nodes are leaves, then $r(\hat{t}) = \hat{t}$. We have then 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.

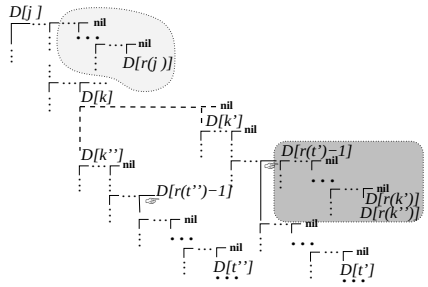

Fig. 4. Sharing into a same r_keyroot

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

4.2 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, with an OR node $D[k]$ being a common ancestor of these two nodes. We suppose that the r_keyroots are in different branches, namely, 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:

$$
\text{f\_d}(r(i)..s, r(\hat{\jmath})..\hat{t}) = \left\{ \begin{array}{l}
\left\{ \begin{array}{l}
\min \left\{ \begin{array}{ll}
\text{f\_d}(r(i)..s-1,\ r(\hat{\jmath})..\hat{t}) & +\gamma(P[s] \rightarrow \varepsilon), \\
\text{f\_d}(r(i)..s,\ r(\hat{\jmath})..\hat{t}-1) & +\gamma(\varepsilon \rightarrow D[\hat{t}]), \\
\text{f\_d}(r(i)..s-1,\ r(\hat{\jmath})..\hat{t}-1) & +\gamma(P[s] \rightarrow D[\hat{t}]), \\
\text{f\_d}(\phi, D[r(\hat{\jmath})]..\hat{t}-1) & +\min_{\hat{t}_k}\{\text{t\_d}(s,\hat{t}_k) - \\
 & \quad \text{t\_d}(\phi,\hat{t}_k)\} \quad 1 \leq k \leq n_{\hat{t}}
\end{array} \right\} \\
\textbf{iff } P[s] = | \\
\\
\min \left\{ \begin{array}{ll}
\text{f\_d}(r(i)..s-1,\ r(\hat{\jmath})..\hat{t}) & +\gamma(P[s] \rightarrow \varepsilon), \\
\text{f\_d}(r(i)..s,\ r(\hat{\jmath})..\hat{t}-1) & +\gamma(\varepsilon \rightarrow D[\hat{t}]), \\
\text{f\_d}(r(i)..s-1,\ r(\hat{\jmath})..\hat{t}-1) & +\gamma(P[s] \rightarrow D[\hat{t}]), \\
\min_{\hat{t}_k}\{\text{t\_d}(s,\hat{t}_k), & 1 \leq k \leq n_{\hat{t}}, \\
\min_{\hat{t}_k}\{\text{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}{ll}
\text{f\_d}(r(i)..s-1,\ r(\hat{\jmath})..\hat{t}) & +\gamma(P[s] \rightarrow \varepsilon), \\
\text{f\_d}(r(i)..s,\ r(\hat{\jmath})..\hat{t}-1) & +\gamma(\varepsilon \rightarrow D[\hat{t}]), \\
\text{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}{ll}
\text{f\_d}(r(i)..s-1,\ r(\hat{\jmath})..\hat{t}) & +\gamma(P[s] \rightarrow \varepsilon), \\
\text{f\_d}(r(i)..s,\ r(\hat{\jmath})..\hat{t}-1) & +\gamma(\varepsilon \rightarrow D[\hat{t}]), \\
\text{f\_d}(r(i)..r(s)-1,\ r(\hat{\jmath})..r(\hat{t})-1) & +\text{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 $r(\hat{t}) = r(\hat{\jmath})$. We can assume that a tail of sons is shared by nodes $D[\hat{t}]$, $\hat{t} \in \{t',\ t''\}$, as well as 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 that we identify syntactic structures and computations, we conclude that the distances $f\_d(r(i)..s,\ r(\hat{\jmath})..\hat{t})$, with $(\hat{\jmath},\hat{t}) \in \{(j',\ t'),\ (j'', t'')\}$ do not depend on previous computations over the shared tail. This sharing has no effect on the computation, although it does affect the computation of distances for nodes in the rightmost branch of the tree immediately to the left of the shared tail of sons, denoted by a double dotted line in the second case of Fig. 5.

The computation of the forest distance when $r(\hat{t}) \neq r(\hat{\jmath})$, is shown in the second case of Fig. 5. In relation to each one of the three alternative values used to compute the minimum, we have that:

1. The values for $f\_d(r(i)..s-1,\ r(\hat{\jmath})..\hat{t})$, $(\hat{\jmath},\hat{t}) \in \{(j',\ t'),\ (j'', t'')\}$ 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 $D[\hat{t}]$, $\hat{t} \in \{(t',\ t'')$:
   - If both nodes are leaves, then $r(\hat{t}) = \hat{t}$. We have then that $D[t'-1] = D[r(t')-1] = D[r(t'')-1] = D[t''-1]$, and therefore the values for

distances $f\_d(r(i)..s,\ r(\hat{\jmath})..\hat{t}-1)$ with $(\hat{\jmath},\hat{t}),\in\{(j',\ t'),\ (j'',t'')\}$, are also the same.

- Otherwise, following the inverse postorder, we arrive at the rightmost leaves of $D[t']$ and $D[t'']$, where we can apply the reasoning considered in the previous case.

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

As in the case of sharing in a same r_keyroot, a similar reasoning can be applied to compute the values for *s_f_d*, avoiding redundant computations.

5 Experimental results

To deal with approximate pattern-matching as a query facility, we are interested in both considering the point of view of the user, which determines the choice of one or other VLDC symbol, and in showing the influence of this choice on the overall computational cost. We also estimate the impact of sharing on the process described. This is of interest, because user queries can vary widely from the norm. Thus, the goal is to find a pattern which most closely matches the user query. Ambiguity arises, since this query can be considered to be a distorted version of any of several possible patterns. So, sharing saves on the space needed to represent these structures, and also on their later processing.

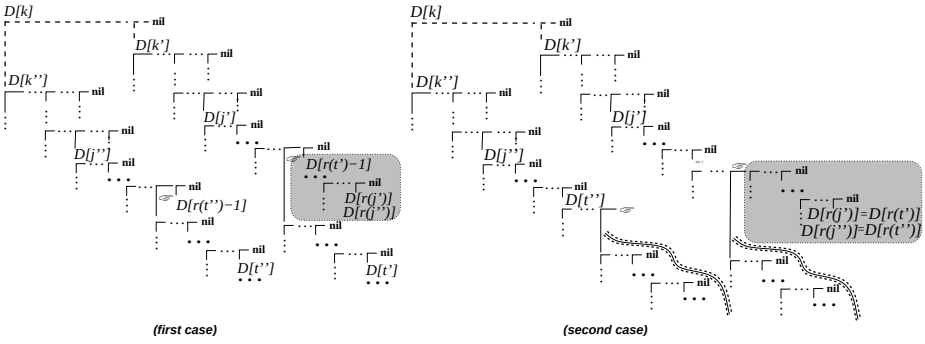

Fig. 5. Sharing between different r_keyroots

In order to interpret the practical results, we need a formal environment. So, we define a set of properties which are of interest in an approximate pattern-matching algorithm. The first property is the *abstraction*, a ratio between the number of nodes participating in the substitutions of VLDC symbols related to the total number of nodes in the data tree. Intuitively, the abstraction measures the entity of the regions covered by VLDC symbols. If we consider that greater

levels of abstraction in the query means a smaller effort of the user to express it, we have a simple criterion to reflect this effort.

The second property is the *effort*, the number of elementary operations applied, which measures the computational effort. The third property is the *performance*, defined as the ratio between the number of the VLDC operations contributing to the final distance, and the total number of VLDC operations actually computed. This allows us to estimate the actual exploitation of the computations involving VLDC symbols. Finally, to measure the *quality* of the pattern-matching process, we consider the ratio between the distance obtained by the algorithm and the number of nodes in the data tree. To perform the tests, we have considered shared forest obtained from the parsing of sentences of the form *"John gives the cake to (the friend of)^i Mary"*, for value of $i = 7$, using the following non-deterministic grammar of English:

(1) S → NP VP
(2) VP → verb NP PP
(3) NP → noun
(4) NP → det noun
(5) NP → det noun PP
(6) NP → det noun PP PP
(7) PP → prep NP

Given that the grammar contains a rule "*NP → det noun PP PP*", these sentences have a number of ambiguous parses which grows exponentially with i. This allow us to evaluate our proposal in a strongly ambiguous context, in spite of the simplicity of the grammar. As patterns, we have used a set of deterministic parse trees. To start with, we have considered a deterministic parse tree for the sentence *"John gives the cake to (the friend of)^5 Mary"*. We then generate new pattern trees by means of the inclusion of VLDC nodes in such a way that the vagueness of the resulting patterns increases, as shown in Fig. 6. To show the difference between the VLDC symbols, we have considered two different sets of patterns, one including only "∧" symbols and another with only "|" symbols.

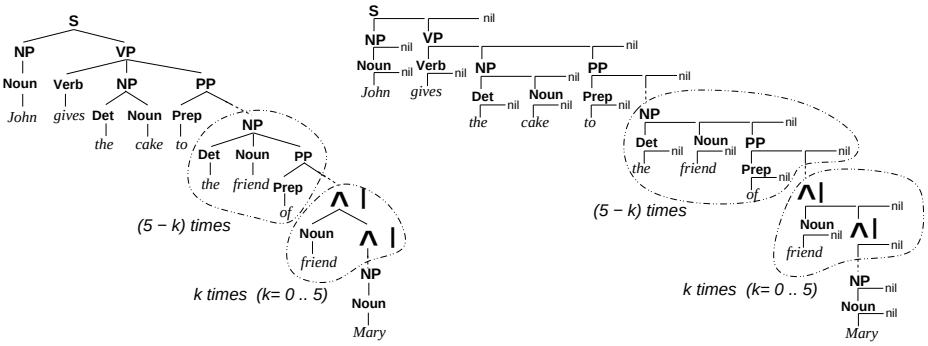

Fig. 6. Pattern trees, classic and AND-OR graph, used in our tests

The experimental results are shown in Figs. 7, 8 and 9. In all cases, we represent the level of abstraction in the patterns used on the X-axis. As expected,

abstraction values increase as long as the pattern tree includes more VLDC symbols. It should also be noticed that the abstraction we get with patterns using only "|" VLDC is much more reduced than in patterns that include "∧" nodes. This a consequence of the kind of VLDC substitutions performed in the "|" nodes, that includes only a sub-path of nodes, but not complete subtrees. All tests illustrated reflect evidence an important reduction in the computational cost due to structural sharing.

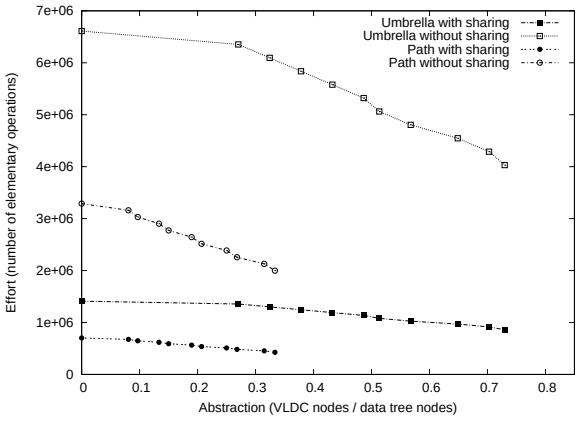

Fig. 7. Tests on effort

In Fig. 7, we show the effort values obtained for different patterns with an increasing degree of abstraction. In our example, the tests show a reduction in the effort when the abstraction increases. This is due to the fact that patterns with a greater abstraction have fewer nodes than more precise ones. In the case of patterns with only "|" VLDC symbols, the effort is smaller, because the algorithm does not need to compute the *s_f_d* values. As a consequence, the use of "∧" symbols always increases the effort, with approximately twice the number of elementary operations being needed to compute the final distance. From an intuitive point of view, these results indicate that the computational effort is in inverse correspondence to the description effort applied by the user to express the query, which seems natural.

In Fig 8, we can see that the highest performance is given for patterns with only "|" VLDC symbols. This behavior is again derived from the fact that in those patterns there is no need to compute *s_f_d* values. Although the number of VLDC operations used may be less, the total number of VLDC computations is much smaller than in the case of "∧" patterns, since the algorithm does not have to compute *s_f_d*'s. Intuitively, this is due to the fact that vagueness in the description of the query favours the pattern-matching process since structural constraints are relaxed.

Finally, in Fig. 9, we show the influence of the different kinds of patterns on the final distance obtained by the algorithm in our tests. The most interesting

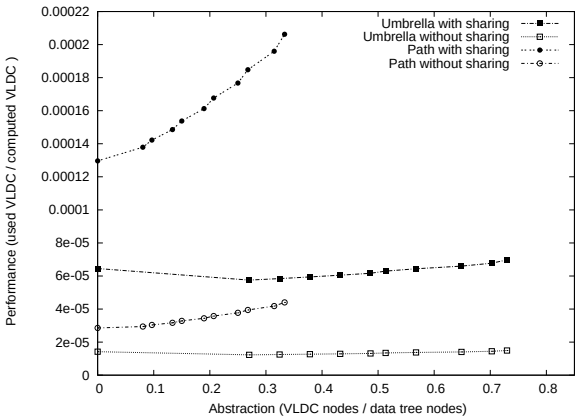

Fig. 8. Tests on performance

feature of these results is the behavior when the pattern has only "|" VLDC symbols. In this case, as the pattern gets less specific, so the distance increases. The reason is that, when the "|" pattern becomes more general, there are fewer nodes specified in the pattern, and, in the best mapping, it is necessary to include insert operations for those nodes. In patterns with "Λ" VLDC symbols, the VLDC substitutions cover these inserted nodes, making the distance zero. From an intuitive point of view, this means that "|" patterns are more inflexible than patterns with "Λ", and more sensitive to the data tree topology. So, this characteristic means that the behaviour of the "|" substitutions must be taken into account when building patterns that include this symbol

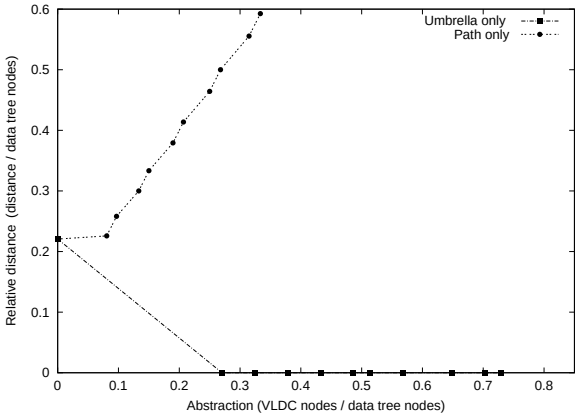

Fig. 9. Tests on quality

6 Conclusions

Practical information retrieval/extraction systems often focuses on systems for which the priority is the recall, to the detriment of precision. This gap is justified by efficiency gains assuming that if the system is too slow it will be intolerable to use, regardless of its ability to identify relevant documents, which may degrade the performance seriously. In this sense, proposals based on approximate pattern-matching technology allow to modulate both the complexity in the query description and computational effort, providing a valid starting point to estimate the impact of these factors in the quality of the answer.

Acknowledgments

This work has been partially supported by the Spanish Government under projects TIC2000-0370-C02-01 and HP2001-0044, and the Autonomous Government of Galicia under project PGIDT01PXI10506PN.

References

1. Efthimis N. Efthimiadis. Interactive query expansion: a user-based evaluation in a relevance feedback environment. *Journal of the American Society for Information Science*, 51(11):989–1003, 2000.
2. P. Kilpeläinen and H. Mannila. Query primitives for tree-structured data. *Lecture Notes in Computer Science*, 807:213–225, 1994.
3. M. Vilares. *Efficient Incremental Parsing for Context-Free Languages*. PhD thesis, University of Nice. ISBN 2-7261-0768-0, France, 1992.
4. M. Vilares, F.J. Ribadas, and V.M. Darriba. Approximate pattern matching in shared-forest. *Lecture Notes in Artificial Intelligence*, 1873:322–333, 2000.
5. M. Vilares, F.J. Ribadas, and V.M. Darriba. Approximate VLDC pattern matching in shared-forest. *Lecture Notes in Artificial Intelligence*, 2004:483–494, 2001.
6. J.T.L. Wang, X. Wang, D. Shasha, B.A. Shapiro, K. Zhang, Q. Ma, and Z. Weinberg. An approximate search engine for structural databases. *SIGMOD Record*, 29(2):584–584, 2000.
7. K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.
8. 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, January 1994.