CAHIER DU LAMSADE

Laboratoire d'Analyse et Modélisation de Systèmes pour l'Aide à la Décision
(Université Paris-Dauphine)
Unité de Recherche Associée au CNRS n° 825

AVERAGE-CASE COMPLEXITY FOR THE EXECUTION OF RECURSIVE DEFINITIONS ON RELATIONAL DATABASES

CAHIER N° 135
février 1996

W. FERNANDEZ de la VEGA [1]
V.Th. PASCHOS [2]
A.N. STAFYLOPATIS [3]

received: October 1995.

[1] *LRI, Université de Paris-Sud, Centre d'Orsay, 91405 Orsay Cedex, France, e.mail; lalo@lri.fr.*

[2] *LAMSADE, Université Paris-Dauphine, Place du Maréchal De Lattre de Tassigny, 75775 Paris Cedex 16, France, e.mail: paschos@lamsade.dauphine.fr.*

[3] *Computer Science Division, National Technical University of Athens, 15773 Zographou, Athens, Greece, e.mail: andreas@theseas.ntua.gr.*

1 Introduction **1**

2 The model **1**
2.1 Some properties of the model 2
2.2 Performance measures 3

3 The algorithms **3**
3.1 The direct method algorithm 3
3.2 The intermediate storage algorithm 4
3.3 The answers to the queries 5
3.3.1 Δ is fixed 5
3.3.2 Considering all the forests on a fixed number of nodes 5

4 The execution costs for a given EDB **6**
4.1 The direct method 6
4.1.1 The query $R(\alpha, y)$ 6
4.1.2 The query $R(\alpha, \beta)$ 6
4.1.3 The query $R(x, \beta)$ 7
4.1.4 The query $R(x, y)$ 8
4.2 The intermediate storage algorithm 8
4.2.1 The query $R(\alpha, \beta)$ 8
4.2.2 The query $R(x, \beta)$ 9
4.2.3 The query $R(x, y)$ 9

5 Average-case analysis **9**
5.1 The direct method 9
5.1.1 The query $R(\alpha, y)$ 9
5.1.2 The query $R(\alpha, \beta)$ 9
5.1.3 The query $R(x, \beta)$ 11
5.1.4 The query $R(x, y)$ 11
5.2 Mean execution costs for the intermediate storage method 11
5.2.1 The query $R(\alpha, \beta)$ 11
5.2.2 The query $R(x, \beta)$ 11
5.2.3 The query $R(x, y)$ 12

6 Conclusions **12**

Complexité moyenne pour évaluer des requêtes basées sur des définitions récursives dans les bases de données relationnelles

Résumé

Les coûts d'exécution de divers types de requêtes dans les bases de données sont établis pour deux algorithmes d'évaluation de requêtes dans le cas où les relations de base de données sont représentées par des forêts d'arbres orientés étiquetés. Les coûts d'exécution sont tout d'abord calculés pour une forêt donnée. Puis les moyennes de ces coûts sont calculées en considérant toutes les bases de données qui peuvent être représentées par une forêt avec un nombre donné de nœuds.

Mots-clés: base de données relationnelle, définition récursive, arbre, algorithme, complexité.

Average-case complexity for the execution of recursive definitions on relational databases

Abstract

The execution costs of various types of database queries are evaluated for two common query evaluation algorithms in the case where the database relations are represented by forests of labelled oriented trees. The execution costs are computed first for a given forest. Then, the averages of these costs, computed over all databases representable by forests with a given number of nodes, are also evaluated.

Keywords: relational database, recursive definition, tree, algorithm, complexity.

1 Introduction

In [6] the mean execution costs of some query evaluation algorithms were examined, for database relations represented by full tree structures. In the present paper we extend these results to the case of any forest-like structure. Moreover, we perform an average-case analysis over all forests on a given number of nodes. We refer to [6] for motivation and for references to former results. The simplicity of the forest structure allows us to perform a fairly detailed analysis and to derive tight time bounds.

Our work complements the one done by Bancilhon and Ramakrishnan ([2]). These authors treat relations represented by some particular tree structures (full trees), by inverted tree structures and also by what they call cylinders (covering graphs of a particular class of graded partial orders), whereas we consider any kind of tree, in fact any forest, which may represent a database relation.

We present two query evaluation techniques which we call the *direct method* and the *intermediate storage method*, respectively. Using database terminology, the direct method is a prolog-like top-down evaluation which uses a reordering of goals in order to ensure termination (we will be more precise below); the intermediate storage method is a two-stage method: first, the constants are pushed into the recursive rules and then, the query is evaluated in a bottom-up semi-naïve fashion ([2]).

The derivation of mean execution costs has now become an important chapter in the analysis of algorithms ([10]). Average-case analysis provides results which, to some extent, summarize the salient features of the behaviour of the algorithm and can highlight aspects of the problem that are not visible through the most commonly used worst-case analysis. In fact, by using average-case evaluation, we can gain a quick insight into the properties of the algorithms without depending on information that is really too detailed to be handled in practice. Of course, we do not imply that this mean execution costs, based on the hypothesis of equal occurrence probabilities for the distinct possible queries, or, at the higher level, of equal occurrence probabilities for the distinct possible queries forest structures for the EDB relation, reflect exactly the actual costs in every situation occurring in practice. However, we have undertaken to treat the average-case complexity analysis of logic programming algorithms in a systematic way and to produce asymptotic expressions for this complexity; this, to our knowledge, is the first time that such a theoretical thought process is undertaken.

The paper is organized as follows. In the next section we introduce the model and discuss its properties. In section 3 we present the two algorithms, the direct and the intermediate storage methods. Section 4 deals with the study of the average-case complexity of these two methods when the forest structure of the EDB relation is fixed. Finally, in section 5, asymptotical expressions for the mean execution costs (taken over all forests) are given.

2 The model

Let Q denote a binary relation on some set X (data set). We are concerned with the analysis of the cost of queries $R(x, y)$ defined recursively as follows:

$$\begin{aligned} r_e &: R(x,y) \leftarrow Q(x,y) \\ r_r &: R(x,y) \leftarrow Q(x,z), R(z,y) \end{aligned}$$

Clearly, the relation R holds precisely for the pairs (x, y) for which there is a directed path from x to y in the digraph of Q. Accordingly, Q is called *external database (EDB) relation*, R, defined by means of the above rules, is called *internal database (IDB) relation*, while the above rules are called the transitive closure definition in the database terminology.

We will assume in this paper that the graph of Q is a forest F of labelled oriented trees, that is, trees, with labeled nodes, where the left-to-right order of subtrees is immaterial(for more details on the definition of this type of trees, see [3]) called *labelled oriented forest.* So, the problem studied in this paper is the computation of the average-case complexity of the two algorithms (algorithms 1 and 2), presented in section 3, when they operate on labelled oriented forests.

2.1 Some properties of the model

We consider the nodes of the forest as distinguishable points, since they represent distinct values of the domain of Q. For convenience, we fix the set of labels: $X = \{1, 2, \ldots, n\}$.

We define as usual the execution cost for each type of query of interest when applied to the EDB represented by a fixed forest Δ, as the average, taken over all instances of the query, of the number of steps used (by some given algorithm) to evaluate an instance of this query. Then we will compute, again for each fixed query, the mean of the execution costs taken over all distinct forests with a given number of nodes.

The following notion of node equivalence is basic in our work, since, as it will be seen, all our calculations are expressed in terms of classes of equivalent nodes.

Definition 1 (node equivalence). Consider a forest $\Delta = \{T_1, \ldots, T_k\}$. The nodes of Δ are partitioned into equivalence classes $C_1, \ldots, C_l$ recursively defined as follows:

- every node is equivalent to itself;
- two nodes of Δ, none of which is the root of a component tree of Δ, are equivalent if their fathers are equivalent and, moreover, the oriented subtrees emanating from these nodes are isomorphic;
- two roots of component trees T_i and T_j of Δ are equivalent iff T_i and T_j are isomorphic oriented trees. ∎

Given an oriented forest Δ and the partition of its nodes into equivalence classes $C_1, C_2, \ldots, C_l$, we will use the following notations:

Δ:	the given forest structure;
$n(\Delta)$:	the order (number of nodes) of Δ;
T_i:	the subtree rooted at a node of class i;
$n(T_i)$:	the order of T_i;
l_i:	the level of the nodes of class i (assuming that the roots of the component trees are at level 0);
card(i):	the cardinality of class i;
h_i:	the height of the subtree rooted at a node of class i;
D_i^k:	the set of classes to which belong the kth descendants of a node of class i; obviously, $D_i^0 = \{i\}$ and D_i^1 corresponds to the children of nodes of class i;
σ_i:	the number of nodes of class i having the same father (by definition of the classes, all the nodes of each class have the same number of children of each class).

The node equivalence relation leads to the following "uniformity" proposition.

Proposition 1. *Consider a fixed oriented forest Δ. The number of labellings of Δ in which a given label is assigned to a node of a given equivalence class is equal to the cardinality of this class multiplied by a constant, which is the same for all classes.*

Proof: Let $\alpha(\Delta)$ be the number of distinct labellings of Δ. Let K_i denote the number of these labellings in which the label j is assigned to some node in class i. Setting $n = n(\Delta)$, we have $\alpha(\Delta) = C^n_{\text{card}(1),\text{card}(2),\ldots,\text{card}(l)}$, i.e., $\alpha(\Delta)$ is equal to the number of ways of distributing the n labels within the classes (indeed exchanging two labels within the same class, results in the same labelled forest and two distinct distributions of the labels give *distinct* labelled forests), and $K_i = C^{n-1}_{\text{card}(i)-1} C^{n-\text{card}(i)}_{\text{card}(1),\ldots,\text{card}(i-1),\text{card}(i+1),\ldots,\text{card}(l)}$, where the first term accounts for the choices of the labels of class C_i, other than label j, and the second for the number of ways in which the remaining labels can be distributed in the rest of the classes. We can write now

$$K_i = \alpha(\Delta)\frac{\binom{n-1}{\text{card}(i)-1}}{\binom{n}{\text{card}(i)}} = \text{card}(i)\frac{\alpha(\Delta)}{n(\Delta)}. \blacksquare \tag{1}$$

2.2 Performance measures

Let again Δ be any forest on n nodes. The queries that are usually considered fall into the following categories:

- list the descendants in Δ of a particular node α: query $R(\alpha, x)$;
- does there exist in Δ a path linking two particular nodes α and β?: query $R(\alpha, \beta)$;
- list the ascendants in Δ of a particular node β: query $R(y, \beta)$;
- find the paths linking every pair (x, y) of nodes in Δ: query $R(x, y)$.

We shall denote by c^1_R (resp., c^{12}_R, c^2_R, and c^0_R), the corresponding execution cost (for a given forest on n nodes) of the query $R(\alpha, y)$ (resp., $R(\alpha, \beta)$, $R(x, \beta)$ and $R(x, y)$). The mean execution costs (averaged over all forests on n nodes) will be denoted by replacing c by γ in the previous notations, i.e., γ^1_R, γ^{12}_R, and so on. Finally, the costs for the relation Q will be denoted by replacing by Q the subscript R in the above notations, i.e., c^1_Q, c^{12}_Q, and so on. We assume as usual that the quantities c_Q are given. Details on this matter can be found in [9].

3 The algorithms

We describe in this section the two query evaluation algorithms, the average-case complexity of which we study in the sequel. Moreover, for each query, we describe the type of the obtained answer in terms of forest's parameters.

3.1 The direct method algorithm

The first algorithm (algorithm 1), called direct method, is a kind of exhaustive procedure which, without applying any cost reduction technique, performs a prolog-like top-down evaluation, this evaluation using a reordering of goals in order to ensure termination.

Concerning query $R(\alpha, \beta)$, procedure **bfs**(α, β) consists of searching, in a breadth-first-search manner ([1]), the nodes of the subtree rooted at α until either β is found, or the whole subtree is exhausted.

Concerning query $R(x, \beta)$, the result of the execution of r_e (line $(**)$) is the list of all the nodes of the forest. Moreover, once this execution is completed, all the values of the nodes are known and, consequently, the evaluation of the query $R(x, \beta)$ is reduced to the evaluation of the query $R(\alpha, \beta)$.

Finally, for query $R(x, y)$, once all the values of the domain of Q are known, the evaluation of $R(x, y)$ is reduced to the evaluation of $R(\alpha, y)$.

```
begin
    case R(·,·) do
        R(α,y): execute r_e with α fixed;
                store the results
                repeat
(*)                 evaluate Q(α,z) of r_r with α fixed;
                    for each ζ such that Q(α,ζ) = true do
                        execute (*) by substituting ζ for α
                    od
                until no new results
        R(α,β): bfs(α,β)
(**)    R(x,β): execute r_e with both variables free
                for each node value ζ obtained from the execution of line (**) do
                    execute case R(α,β) by substituting ζ for α
                od
(***)   R(x,y): execute r_e with both variables free;
                for each node value ζ obtained from the execution of line (**) do
                    execute case R(α,y) by substituting ζ for α
                od
    od
end.
```

Algorithm 1. The direct method algorithm.

```
begin
    execute r_e and store tuples
    repeat
        execute r_r putting in the place of the recursive predicate the last stored tuples;
        store new tuples
    until no new tuples
end;
```

Procedure 1. The query pre-processing.

3.2 The intermediate storage algorithm

The intermediate storage method (algorithm 2) is a two-stage method: first, during a query pre-processing, the constants are pushed into the recursive rules; next, the query is evaluated in a bottom-up semi-naïve manner.

The query pre-processing is a natural cost-reduction strategy reducing the number of accesses to the disk where data are stored; it is described in procedure 1.

For the application of procedure 1 to the transitive closure definition, the following condition must hold: *when there exist bound variables, at least one bound variable corresponding to some attribute must have the same value in the occurrences of the recursive predicate on both sides of the recursive rule, so that the previously stored tuples may be used.* For instance, the intermediate storage method cannot be applied for the query $R(\alpha, y)$.

As a matter of fact, the procedure works by creating chains of tuples such that, for any two successive tuples in a chain, the second attribute of the first tuple has the same value as the first attribute of the second tuple. The applicability condition mentioned above ensures that such

```
begin
    case R(·,·) do
        R(α,β): starting from β use procedure 1 to climb the tree up to its root
                until either α is found or the root of the tree is attained;
        R(x,β): execute r_e;
                starting from β use procedure 1 to climb the tree up to its root;
                store all the nodes on the path;
(#)     R(x,y): execute r_e;
                for each node value ζ obtained from the execution of line (#) do
                    execute case R(x,β) by substituting ζ for β
                od
    od
end.
```

Algorithm 2. The intermediate storage algorithm.

chains can be created. For more details on intermediate storage, we refer to [5].

3.3 The answers to the queries

As usually, in order to analyze the average-case complexity of the devised algorithms we first need to characterize the type of the answer in terms of input parameters.

3.3.1 Δ is fixed

For **case** $R(\alpha, y)$ (admitted only by algorithm 1), the answer will be the list of the labels of the nodes of the subtree emanating from α. Moreover, the case where α is a leaf is trivial, since it yields the empty set.

For **case** $R(\alpha, \beta)$, there is an answer to the query iff β is contained in the subtree rooted at α; so the answer here will be "yes" or "no".

For **case** $R(x, \beta)$, the answer will be the list of labels on the path from β to the root of the tree containing β.

Finally, for **case** $R(x, y)$, the answer will be the list (empty in the case where x and y belong to two distinct trees of Δ) of the nodes on the paths between all the pairs of nodes of Δ.

3.3.2 Considering all the forests on a fixed number of nodes

The average cost for **case** $R(\alpha, y)$ using algorithm 1 is a function of the average order of a subtree (the average being taken over all labelled oriented forests) of a labelled oriented forest on n nodes.

For **case** $R(\alpha, \beta)$, according to algorithm 1, this query involves either the exploration of the whole subtree of α when β does not belong to this subtree (in this case the average cost will be equal to the one of the **case** $R(\alpha, y)$) or, else, all the nodes of this subtree belonging to the levels strictly higher than the level of β (**bfs** search; in this case, a kind of average length of forest path, over all the forests on a fixed number of nodes, will be implicated in the average cost computation).

For the two last **cases** ($R(x, \beta)$ and $R(x, y)$) the average cost will be, once more, a function of the average forest path size, over all the labelled oriented forests on a given number of nodes.

4 The execution costs for a given EDB

4.1 The direct method

4.1.1 The query $R(\alpha, y)$

Proposition 2. *The execution cost c_R^1 is given by:*

$$c_R^1 = \sum_{i=1}^{C} \frac{\text{card}(i)}{n(\Delta)}\, n(T_i)\, c_Q^1. \tag{2}$$

Proof: Let α be the label of a node in class C_i. The corresponding proportion of labellings is equal to $K_i/\alpha(\Delta) = \text{card}(i)/n(\Delta)$, according to (1). For each of the $n(T_i)$ nodes of T_i the file containing Q will be searched with cost c_Q^1. Thus the execution cost, obtained by averaging the costs $n(T_i)c_Q^1$ over all the classes with the weights $\text{card}(C_i)$, is given by (2). ∎

4.1.2 The query $R(\alpha, \beta)$

We first need the following proposition.

Proposition 3. *Suppose that α is the label of a node of class C_i. Then the conditional probability $p_{\beta/\alpha}$ that a given label β is contained in the subtree rooted at α is given by $p_{\beta/\alpha} = (n(T_i) - 1)/(n(\Delta) - 1)$.*

For the proof, we only have to observe that the number of labellings considered in which a given subset of labels $L \subseteq \{1, 2, \ldots, n\} \setminus \{\alpha\}$ with $|L| = n(T_i) - 1$ is assigned to the nodes of T_i is independent of L. The probability above is just the proportion of these sets which contain β.

We can now proceed to the evaluation of c_R^{12}.

Proposition 4. *The execution cost c_R^{12} is given by:*

$$c_R^{12} = \sum_{i=1}^{C} \frac{\text{card}(i)}{n(\Delta)} \left[\frac{n(\Delta) - n(T_i)}{n(\Delta) - 1} X(i) + \frac{n(T_i) - 1}{n(\Delta) - 1} Y(i) \right] \tag{3}$$

where

$$X(i) = n(T_i)(c_Q^{12} + c_Q^1) \tag{4}$$

$$Y(i) = \frac{1}{\sum_{j=0}^{h_i - 1} g_j^i} \sum_{j=1}^{h_i - 1} \left[g_j^i \sum_{k=0}^{j-1} (g_k^i + f_k^i) + \frac{g_j^i (g_j^i - 1)}{2} + \frac{g_j^i f_j^i}{2} \right] (c_Q^{12} + c_Q^1) + c_Q^{12}. \tag{5}$$

Proof: Let α be the label of a node of class i. We distinguish two possibilities for β.

(i) The label β is not contained in the subtree rooted at α. The probability of this event is $1 - p_{\beta/\alpha} = (n(\Delta) - n(T_i))/(n(\Delta) - 1)$.
The whole subtree rooted at α (including α) will be searched with cost $X(i)$ given by expression 4.

(ii) The label β is contained in the subtree rooted at α. This happens with probability $p_{\beta/\alpha}$. The subtree will be searched until the father of β is attained. According to the uniformity property, which applies to the subtree too, the label β is uniformly distributed over the nodes of this subtree, excluding α. This implies that the same property holds for the father of β with respect to all the internal nodes of the subtree. For each node visited before the father of β, the cost is equal to $c_Q^{12} + c_Q^1$, while the cost of searching from the father of β is c_Q^{12}.

We recall here that under the **bfs** method, the nodes are searched level by level starting from node α down to the level preceding immediately that of β, this last level being visited until

the father of β is attained. Let us suppose that β's father is situated at level j of the subtree. Before β's father is reached, all nodes at levels with index lower than j will have already been visited and also perhaps some nodes at level j.

Let us define the following quantities:

g_j^i: the number of internal nodes at level j of a subtree rooted at a node of class i, $0 \leq j < h_i$;

f_j^i: the number of leaves at level j of a subtree rooted at a node of class i, $0 \leq j \leq h_i$.

These quantities can be expressed as follows in terms of the quantities defined in section 2:

$$\begin{aligned} g_0^i &= 1_{\{D_i^1 \neq \emptyset\}} \\ g_j^i &= \sum_{k_1 \in D_i^1} \sigma_{k_1} \Big(\sum_{k_2 \in D_{k_1}^1} \sigma_{k_2} \cdots \Big(\sum_{\substack{k_j \in D_{k_{j-1}}^1 \\ D_{k_j}^1 \neq \emptyset}} \sigma_{k_j} \Big) \cdots \Big), \; 1 \leq j < h_i \end{aligned} \tag{6}$$

$$\begin{aligned} f_0^i &= 1_{\{D_i^1 = \emptyset\}} \\ f_j^i &= \sum_{k_1 \in D_i^1} \sigma_{k_1} \Big(\sum_{k_2 \in D_{k_1}^1} \sigma_{k_2} \cdots \Big(\sum_{\substack{k_j \in D_{k_{j-1}}^1 \\ D_{k_j}^1 = \emptyset}} \sigma_{k_j} \Big) \cdots \Big), \; 1 \leq j \leq h_i \end{aligned} \tag{7}$$

where the symbol $1_{\{X\}}$ denotes the indicator function of the event X.

The execution cost $Y(i)$ (expression 5) for case **(ii)** is obtained by averaging over all internal nodes the search cost corresponding to previously visited nodes.
Let us note here that since the indices have ranges linear in n, the quantities g_j^i (expression (6)), f_j^i (expression (7)) and, consequently $Y(i)$ (expression (5)) can all be computed in polynomial time.

In (5) the expression in square brackets gives the number of nodes visited before the father of β, the latter being an internal node at level j of the subtree rooted at α, summed over all internal nodes of that level. The first term in this expression accounts for nodes at levels with index lower than j, that have been visited. The second term corresponds to internal nodes at level j that have been visited before the father of β and is equal to $\sum_{k=1}^{g_j^i}(k-1)$. The third term accounts for leaves at level j, that are visited. Its expression is derived as follows. Let us suppose that there are $n+m$ nodes at a given level where n is the number of internal nodes and m is the number of leaves. We denote by $Z(n,m)$ the average of the sum, taken over the internal nodes of the considered level, of the number of leaves that have been visited before each of these nodes, the average being taken over all possible arrangements of the $n+m$ nodes. We can write: $Z(n,m) = [n/(n+m)]Z(n-1,m) + [m/(n+m)][n + Z(n,m-1)]$, where the first term on the right-hand side corresponds to the event that during the search we first encounter an internal node and the second term to the event that we first encounter a leaf. With the initial conditions $Z(n,0) = 0$ and $Z(0,m) = 0$, the above expression for $Z(n,m)$ yields: $Z(n,m) = nm/2$. By substituting g_j^i and f_j^i for n and m, respectively, we complete the derivation of (5).

The expressions in (4) and (5) for cases **(i)** and **(ii)**, respectively, are combined in (3) to yield the execution cost. ∎

4.1.3 The query $R(x,\beta)$

We assume that the edges of the forest are stored as usual with direct access to their first node. The execution of line (**) of algorithm 1 is required because we need here direct access to the second node of each edge.

Proposition 5. *The execution cost c_R^2 verifies*

$$c_R^2 \sim c_Q^2 + c_Q^0 + [n(\Delta) - 1]c_R^{12}. \tag{8}$$

Proof: Equation (8) is immediately derived from **case** $R(x, \beta)$ of algorithm 1. ∎

4.1.4 The query $R(x, y)$

Proposition 6. *The execution cost c_R^0 verifies*

$$c_R^0 \sim c_Q^0 + [n(\Delta) - 1]c_R^1. \tag{9}$$

Proof: Expression in (9) is immediately derived from **case** $R(x, y)$ of algorithm 1. ∎

4.2 The intermediate storage algorithm

4.2.1 The query $R(\alpha, \beta)$

Before proceeding to the evaluation of the execution cost we need the following observation, the proof of which is completely similar to the one of proposition 3 and is omitted.

Suppose that β is the label of a node of class i (level l_i). Then the probability $p'_{\alpha/\beta}$ of the event that a given label α is situated on the path from the root of the component tree containing β down to β is given by $p'_{\alpha/\beta} = l_i/(n(\Delta) - 1)$. Furthermore, given that α is situated on the path from the root of the component tree containing β down to β, the probabilities that α labels any node on this path are equal.

Proposition 7. *The execution cost c_R^{12} using the intermediate storage algorithm is equal to:*

$$c_R^{12} = \sum_{i=1}^{C} \frac{\mathrm{card}(i)}{n(\Delta)} \left[\left(1 - \frac{l_i}{n(\Delta) - 1} \right) (l_i + 1) + \frac{l_i}{n(\Delta) - 1} \frac{l_i + 1}{2} \right] c_Q^2. \tag{10}$$

Proof: Suppose that β is the label of a node of class i (level l_i). The forest will be searched from β to the root of the component tree containing β until the label α is found or until the tuples are unsuccessfully exhausted. At each step of the search procedure the cost is c_Q^{12} in case of success and $c_Q^{12} + c_Q^2$ in case of failure. Since, however, we are searching on a tree structure, we have $c_Q^{12} = c_Q^2$, because there is at most one tuple having β as the value of its second attribute. Moreover, we need not to search twice in case of failure, since a single search will provide us with the appropriate tuple whether the step is successful or not. Hence the total search cost for all the steps of the procedure is equal to c_Q^2.

We distinguish two possibilities:

(j) The label α does not belong to the path from β to the root of the component tree containing β. In this case the whole path is (unsuccesfully) searched. This event happens with probability $1 - p'_{\alpha/\beta} = 1 - [l_i/(n(\Delta) - 1)]$. The corresponding search cost is equal to:

$$X(i) = (l_i + 1)c_Q^2. \tag{11}$$

(jj) The label α lies on the path from β to the root of the corresponding component tree with probability $p'_{\alpha/\beta}$.

The path will be searched until the child of α is attained. As observed before, the label α is uniformly distributed over the l_i nodes on the path from the root of the component tree to β, and hence the position of its child on the path will be uniformly distributed over l_i nodes (including β). The execution cost will be in this case:

$$Y(i) = \frac{1}{l_i} \sum_{j=1}^{l_i} j\, c_Q^2 = \frac{l_i + 1}{2} c_Q^2,\ l_i > 0. \tag{12}$$

By using (11) and (12) for cases **(j)** and **(jj)**, respectively, we obtain the expression in (10). ∎

4.2.2 The query $R(x,\beta)$

Proposition 8. *The execution cost c_R^2 under the intermediate storage method is equal to:*

$$c_R^2 = \sum_{i=1}^{C} \frac{\mathrm{card}(i)}{n(\Delta)} (l_i + 1) c_Q^2.$$

Proof: Immediate from case $R(x,\beta)$ of algorithm 2. ∎

4.2.3 The query $R(x,y)$

Proposition 9. *The mean execution cost c_R^0 under the intermediate storage algorithm satisfies:*

$$c_R^0 \sim c_Q^0 + [n(\Delta) - 1](c_R^2 - c_Q^2).$$

Proof: The proof of the proposition is immediately obtained from case $R(x,y)$ of algorithm 2. ∎

5 Average-case analysis

When one specializes the results of the previous section to particular classes of trees such as full regular trees ([6], see also [2]) or chains ([6]), one sees that a great variability takes place concerning the costs. For instance, we have $c_R^1 = O(\log n)$ for the full trees whereas c_R^1 is linear in n for chains. It seems thus appropriate to study the behaviour of our algorithms on "most" cases. This is what we do in this section, where we average the values of the execution costs of the queries, over all the forests on a fixed number n of nodes.

5.1 The direct method

5.1.1 The query $R(\alpha,y)$

Consider a labelled tree T with $n+1$ nodes and notice that there is a one to one correspondence between the set of subtrees of T, except T itself, and the subtrees of the forest "pending" from the neighbours of the root of T. Let $f(n)$ (resp., $g(n)$) denote the average order of a subtree of a random rooted tree on n nodes (resp., of a random oriented forest on n nodes of $R(\alpha,y)$). This correspondence implies clearly $f(n+1) = [n/(n+1)]g(n) + 1$. It is well known (see [4]) that $f(n) \sim (\pi n/2)^{1/2}$. Hence the average cost γ_R^1 for the query $R(\alpha,y)$ satisfies

$$\gamma_R^1 \sim c_Q^1 \sqrt{\frac{\pi n}{2}}.$$

5.1.2 The query $R(\alpha,\beta)$

We will need two results concerning the number $g(n)$ of oriented forests on n nodes. Let us first recall the following lemma, due to Rényi ([7]).

Lemma 1. *Let $1 \le k \le n$. Denote by $F(n,k)$ the number of forests on $V = \{1,2,\ldots,n\}$ which have k components and in which the nodes $1,2,\ldots,k$ belong to distinct components. Then, $F(n,k) = kn^{n-k-1}$.*

Since in an oriented forest the roots of the components are arbitrary, we get $g(n) = \sum_{k=1}^{n} g(n,k)$ with $g(n,k) = kC_k^n n^{n-k-1}$. We have, for $1 \le k \le n-1$, $g(n,k+1)/g(n,k) = (n-k)/(kn)$. It is easy to deduce from this the assertions $g(n) \le en^{n-1}$ and

$$g(n) \sim en^{n-1}. \tag{13}$$

Let us denote by m the size of the subtree T rooted at α and by $V(T)$ its node set; furthermore, for each $h \geq 0$, let us denote by m_h the number of nodes of this subtree belonging to its hth level. The total number of steps needed by the queries $R(\alpha, \beta)$ for $\beta \in V(T)$ is $N(T) = \sum_{h\geq 1}\left(m_{h+1}\sum_{k=1}^{h} m_k\right) = (1/2)[(m-1)^2 - \sum_{h\geq 1} m_h^2]$.
We shall denote by N_n the mean value of $N(T)$ on the set of trees on n nodes. It follows from known results on the moments of the sizes m_h of the levels ([8]) that we have

$$N_n \sim \frac{n^2}{2}. \tag{14}$$

We will also need an estimate for the number $f(n,l)$ of forests on n nodes in which the subtree of a node with given label has order l. Let $g(n)$ denote the number of oriented forests on n nodes. Clearly, the number of distinct trees with a given root and $l-1$ other nodes chosen between $n-1$ other given nodes is $C_{l-1}^{n-1} l^{l-2}$.

Now given such a tree T and a forest on the complementary set of nodes, we get a complete forest (on n nodes) in exactly $n-l+1$ ways, namely by linking the root of T to one node of the given forest or by just adding the tree to the forest as a new component. Hence we have $f(n,l) = C_{l-1}^{n-1} l^{l-2} g(n-l)(n-l+1)$.

We are now well prepared to derive an asymptotic equivalent to the mean cost γ_R^{12} of the query $R(\alpha,\beta)$. Let us set $\gamma_R^{12} = \gamma_{R,o}^{12} + \gamma_{R,i}^{12}$, where $\gamma_{R,o}^{12}$ denotes the cost corresponding to the case where β belongs to the tree of α and $\gamma_{R,i}^{12}$ the cost corresponding to the other case. We have

$$\gamma_{R,o}^{12} = \frac{1}{g(n)}(c_Q^{12} + c_Q^1)\sum_{l=2}^{n}(l-1)f(n,l) \tag{15}$$

$$\gamma_{R,i}^{12} = c_Q^{12} + \frac{1}{g(n)}(c_Q^{12} + c_Q^1)\sum_{l=2}^{n} f(n,l)N_n. \tag{16}$$

We shall prove that $\gamma_{R,i}^{12} = o(\gamma_{R,o}^{12})$.
We have, for $q \leq l \leq n-q$, using Stirling's formula and (13),

$$\begin{aligned} f(n,l) &= (1+\epsilon(q))e\sqrt{\frac{n-1}{2\pi(l-1)(n-l)}}\frac{(n-1)^{n-1}}{(l-1)^{l-1}(n-l)^{n-l}} l^{l-2}(n-l+1)n^{n-1} \\ &= (1+\epsilon(q))e^2(n-1)^{n-1/2}\frac{1}{l-1}\frac{1}{\sqrt{2\pi(l-1)(n-l)}} \end{aligned} \tag{17}$$

where $\epsilon(q) \to 0$ uniformly as $q \to \infty$. From expression (17) we have

$$\begin{aligned} \sum_{l=2}^{n}(l-1)f(n,l) &\geq (1+\epsilon(q))e^2(n-1)^{n-1/2}\sum_{l=q}^{n-q}\frac{1}{\sqrt{2\pi(l-1)(n-l)}} \\ &\geq (1+\epsilon'(q))\pi e^2(n-1)^{n-1/2} \end{aligned} \tag{18}$$

where the last inequality is obtained by approximating the left side sum by an integral.

Using, for $l < q$ (resp., $l > n-q$), the inequality $1/[(l-1)(n-l)]^{1/2} \leq 1/(n-1)^{1/2}$, we get from (17) in the case where $nq^{-2} \to \infty$:

$$\begin{aligned} \sum_{l=2}^{n}(l-1)f(n,l) &\leq e^2(n-1)^{n-1/2}[(1+\epsilon(q))\pi + \frac{2q}{\sqrt{n-1}}] \\ &\leq (1+\epsilon''(q))\pi e^2(n-1)^{n-1/2}. \end{aligned} \tag{19}$$

Since ϵ' and ϵ'' are arbitrarily small for sufficiently big q it follows that we have

$$\sum_{l=2}^{n}(l-1)f(n,l) \sim \pi e^2(n-1)^{n-1/2} \tag{20}$$

and, using (13), (15) and (20): $\gamma^{12}_{R,o} \sim [1/(en^{n-1})](c^{12}_Q + c^1_Q)\pi e^2(n-1)^{n-1/2}$, or

$$\gamma^{12}_{R,o} \sim (c^{12}_Q + c^1_Q)\pi\sqrt{n}. \tag{21}$$

Furthermore, by using (16), (17), (20), (21) and after some easy algebra, we deduce that $\gamma^{12}_{R,i} = o(\gamma^{12}_{R,o})$.
So, (21) gives the asymptotic expression of γ^{12}_R.

5.1.3 The query $R(x,\beta)$

Replacing in (8) c^{12}_R by the value obtained for γ^{12}_R, we get

$$\gamma^2_R \sim c^2_Q + c^0_Q + (n-1)\gamma^{12}_R.$$

5.1.4 The query $R(x,y)$

Replacing in (9) c^1_R by the value obtained for γ^1_R, we get

$$\gamma^0_R \sim c^0_Q + (n-1)\gamma^1_R.$$

The following theorem sums up our results concerning the mean execution costs for the direct method.

Theorem 1. *The mean execution costs γ^1_R, γ^{12}_R, γ^2_R and γ^0_R of algorithm 1 for the queries $R(\alpha,y)$, $R(\alpha,\beta)$, $R(x,\beta)$ and $R(x,y)$, respectively, where the mean is taken over all data base relations represented by forests on n nodes and over all bindings of the variables, satisfy*

$$\begin{aligned}
\gamma^1_R &\sim c^1_Q(\pi n/2)^{1/2} \\
\gamma^{12}_R &\sim (c^{12}_Q + c^1_Q)\pi n^{1/2} \\
\gamma^2_R &\sim (c^{12}_Q + c^1_Q)\pi n^{3/2} \\
\gamma^0_R &\sim c^1_Q \pi n^{3/2}.
\end{aligned}$$

5.2 Mean execution costs for the intermediate storage method

5.2.1 The query $R(\alpha,\beta)$

It is easily seen as in section 5.1.2 that the main contribution to the mean cost comes from the case where β does not belong to the subtree rooted at α. Thus, since in this case, according to section 4.2.1, the query amounts to search the path from β to the root of its tree, the mean cost verifies

$$\gamma^{12}_R \sim \sqrt{\frac{\pi n}{2}}c^2_Q.$$

5.2.2 The query $R(x,\beta)$

Recall that the solutions here are all the nodes on the path from β to the root of its tree. Thus we have similarly as above

$$\gamma^2_R \sim \sqrt{\frac{\pi n}{2}}c^2_Q.$$

5.2.3 The query $R(x, y)$

Replacing the quantities c_R^0 and c_R^2 in proposition 9 by their averages, we get

$$\gamma_R^0 \sim c_Q^0 + (n-1)(\gamma_R^2 - c_Q^2).$$

The following theorem sums up the results concerning the intermediate storage method.

Theorem 2. *With algorithm 2, the mean execution costs γ_R^{12}, γ_R^2 and γ_R^0 for the considered queries $R(\alpha, \beta)$, $R(x, \beta)$ and $R(x, y)$, respectively, where the mean is taken over all data base relations represented by oriented forests on n nodes and over all bindings of the variables, satisfy*

$$\begin{aligned} \gamma_R^{12} &\sim (\pi n/2)^{1/2} c_Q^2 \\ \gamma_R^2 &\sim (\pi n/2)^{1/2} c_Q^2 \\ \gamma_R^0 &\sim (\pi/2)^{1/2} n^{3/2} c_Q^2. \end{aligned}$$

It is seen that the intermediate storage method brings in the case of the query $R(x, \beta)$ a considerable improvement over the direct method.

6 Conclusions

We have presented an average-case complexity analysis of two simple and natural algorithms performing the evaluation of the transitive closure. Both methods are proven to be quite efficient when they operate on relations represented by labelled oriented trees, or forests of labelled oriented trees.

The complexity of each algorithm has been studied in two cases: for any given forest structure we have obtained expressions for the execution cost of the most usual queries; our results are derived, in this case, using a notion of equivalent nodes which is very natural and leads to significant simplifications in the analysis; next, we have derived expressions for the mean costs, the mean being taken over all the possible forest structures with a fixed number of nodes.

At a first level, our approach has provided a formalism for describing the underlying structure mainly based on the notion of equivalent nodes and then has allowed us to obtain expressions for the complexity of the algorithms by averaging over all possible queries on a given database structure. At a second level, average-case results have been obtained by taking into account all possible structures with a given number of nodes. The first level can be used to characterize any given situation but requires a rather detailed representation of the structure. The second level allows an abstract characterization which uses no representation at all and leads to simpler expressions.

Let us point out that our strategy and analysis can be used for most of the evaluation methods appearing in [2].

Finally, a very interesting extension of this work is the study of the average-case complexity of algorithms 1 and 2 whenever the EDB relation Q is represented by means of a directed acyclic graph.

References

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, 1975.

[2] F. Banchilhon and R. Ramakrishnan, *An amateur's introduction to recursive query processing strategies*, Proc. SIGMOD'86, pp. 16-52, 1986.

[3] D. E. Knuth, *The art of computer programming (vol. 1: Fundamental algorithms)*, Addison-Wesley, 1973.

[4] A. Meir and J. W. Moon, *On the altitude of nodes in random trees*, Canad. J. Mathematics 30(5), pp. 997-1015, 1978.

[5] J. Naughton, *Data independent recursion in deductive databases*, Proc. PODS'86, pp. 267-279, 1986.

[6] V. Th. Paschos and A. N. Stafylopatis, *Evaluation of the execution cost of recursive definitions*, The Computer Journal 35, pp. A429-A437, 1992.

[7] A. Rényi, *Some remarks on the theory of trees*, Publ. Math. Inst. Hungar. Acad. Sci. 4, pp. 73-85, 1959.

[8] V. E. Stepanov, *On the distribution of the number of vertices in strata of a random tree*, Th. Prob. Appl. 14(1), pp. 65-78, 1969.

[9] J. D. Ullman, *Principles of database and knowledge-base systems* (vols I and II), Computer Science Press, 1988.

[10] J. S. Vitter and Ph. Flajolet, *Average-case analysis of algorithms and data structures*, in *Handbook of theoretical computer science* (vol. A), pp. 431-525, J. V. Leeuwen ed., Elsevier, 1990.