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  W. FERNANDEZ de la VEGA 1
février 1996 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.
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.
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 $\Delta$, 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 $\Delta$ are partitioned into equivalence classes $C_1, \ldots, C_l$ recursively defined as follows:

- every node is equivalent to itself;
- two nodes of $\Delta$, none of which is the root of a component tree of $\Delta$, 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 $\Delta$ are equivalent iff $T_i$ and $T_j$ are isomorphic oriented trees.

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

- $\Delta$: the given forest structure;
- $n(\Delta)$: the order (number of nodes) of $\Delta$;
- $T_i$: the subtree rooted at a node of class $i$;
- $n(T_i)$: the order of $T_i$;
- $\ell_i$: the level of the nodes of class $i$ (assuming that the roots of the component trees are at level 0);
- $\text{card}(i)$: the cardinality of class $i$;
- $h_i$: the height of the subtree rooted at a node of class $i$;
- $D_k^i$: the set of classes to which belong the $k$th descendants of a node of class $i$; obviously, $D_1^i = \{i\}$ and $D_k^i$ corresponds to the children of nodes of class $i$;
- $\sigma_i$: 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 $\Delta$. The number of labellings of $\Delta$ 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 $\Delta$. 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 $\alpha(\Delta) = C_n^{n(\Delta)}$, 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

$$K_i = C_n^{\text{card}(i)-1} C_n^{\text{card}(j)}$$

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

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

2.2 Performance measures

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

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

We shall denote by $c^1_R$ (resp., $c^2_R$, $c^3_R$, and $c^4_R$), 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 $\gamma$ in the previous notations, i.e., $\gamma^1_R$, $\gamma^2_R$, 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., $c^1_Q$, $c^2_Q$, 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 $\text{bfs}(\alpha, \beta)$ consists of searching, in a breadth-first-search manner ([11]), the nodes of the subtree rooted at $\alpha$ until either $\beta$ is found, or the whole subtree is exhausted.

Concerning query $R(x, \beta)$, the result of the execution of $r_\alpha$ (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 $R(\alpha, y)$. 

3
Algorithm 1. The direct method algorithm.

begin
  execute \( r \) and store tuples
  repeat
    execute \( r \), 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 $R(\cdot, \cdot)$ do
    $R(\alpha, \beta)$: starting from $\beta$ use procedure 1 to climb the tree up to its root
    until either $\alpha$ is found or the root of the tree is attained;
    $R(z, \beta)$: execute $r_z$
    starting from $\beta$ use procedure 1 to climb the tree up to its root;
    store all the nodes on the path;
    ($\#$) $R(z, y)$: execute $r_z$
    for each node value $\xi$ obtained from the execution of line ($\#$) do
      execute procedure 2 by substituting $\xi$ for $z$. 

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 \( c^R_R \) is given by:

\[
c^R_R = \sum_{i=1}^{C} \frac{\text{card}(i)}{n(\Delta)} n(T_i) c^R_Q.
\]

**Proof:** Let \( \alpha \) 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 \( c^R_Q \). Thus the execution cost, obtained by averaging the costs \( n(T_i)c^R_Q \) over all the classes with the weights \( \text{card}(C_i) \), is given by (2). \( \blacksquare \)

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

We first need the following proposition.

**Proposition 3.** Suppose that \( \alpha \) is the label of a node of class \( C_i \). Then the conditional probability \( p_{\beta|\alpha} \) that a given label \( \beta \) is contained in the subtree rooted at \( \alpha \) 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 \( \beta \).

We can now proceed to the evaluation of \( c^R_{R^2} \).

**Proposition 4.** The execution cost \( c^R_{R^2} \) is given by:

\[
c^R_{R^2} = \sum_{i=1}^{C} \frac{\text{card}(i)}{n(\Delta)} \left[ \frac{n(\Delta) - n(T_i)}{n(\Delta) - 1} X(i) + \frac{n(T_i) - 1}{n(\Delta) - 1} Y(i) \right]
\]

where

\[
X(i) = n(T_i)(c^R_Q + c^R_1)
\]

\[
Y(i) = \frac{1}{2} \sum_{j=0}^{h_i-1} \sum_{k=0}^{g_j} \left[ g_j^k (g_j^k + f_k^j) + \frac{g_j^k (g_j^k - 1)}{2} + \frac{g_j^k f_k^j}{2} \right] (c^R_Q + c^R_1) + c^R_{R^2}.
\]

**Proof:** Let \( \alpha \) be the label of a node of class \( i \). We distinguish two possibilities for \( \beta \).

(i) The label \( \beta \) is not contained in the subtree rooted at \( \alpha \). The probability of this event is \( 1 - p_{\beta|\alpha} = (n(\Delta) - n(T_i))/(n(\Delta) - 1) \).

The whole subtree rooted at \( \alpha \) (including \( \alpha \)) will be searched with cost \( X(i) \) given by expression 4.

(ii) The label \( \beta \) is contained in the subtree rooted at \( \alpha \). This happens with probability \( p_{\beta|\alpha} \). The subtree will be searched until the father of \( \beta \) is attained. According to the uniformity property, which applies to the subtree too, the label \( \beta \) is uniformly distributed over the nodes of this subtree, excluding \( \alpha \). This implies that the same property holds for the father of \( \beta \) with respect to all the internal nodes of the subtree. For each node visited before the father of \( \beta \), the cost is equal to \( c^R_Q + c^R_1 \), while the cost of searching from the father of \( \beta \) is \( c^R_{R^2} \).

We recall here that under the bfs method, the nodes are searched level by level starting from node \( \alpha \) down to the level preceding immediately that of \( \beta \), this last level being visited until
the father of \( \beta \) is attained. Let us suppose that \( \beta \)'s father is situated at level \( j \) of the subtree. Before \( \beta \)'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:

\[ g^i_j : \text{the number of internal nodes at level } j \text{ of a subtree rooted at a node of class } i, \ 0 \leq j < h_i; \]
\[ f^i_j : \text{the number of leaves at level } j \text{ of a subtree rooted at a node of class } i, \ 0 \leq j \leq h_i. \]

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

\[
g^i_0 = \begin{cases} 1_{(D^i_1 \neq \emptyset)} \\
\sum_{k_1 \in D^i_1} \sigma_{k_1} \left( \sum_{k_2 \in D^i_{k_1}} \sigma_{k_2} \left( \sum_{k_j \in D^i_{k_{j-1}}} \sigma_{k_j} \right) \right), & 1 \leq j < h_i 
\end{cases}
\]

\[
f^i_0 = \begin{cases} 1_{(D^i_1 = \emptyset)} \\
\sum_{k_1 \in D^i_1} \sigma_{k_1} \left( \sum_{k_2 \in D^i_{k_1}} \sigma_{k_2} \left( \sum_{k_j \in D^i_{k_{j-1}}} \sigma_{k_j} \right) \right), & 1 \leq j \leq h_i 
\end{cases}
\]

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 \( g^i_j \) (expression (6)), \( f^i_j \) (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 \( \beta \), the latter being an internal node at level \( j \) of the subtree rooted at \( \alpha \), 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 \( \beta \) and is equal to \( \sum_{k_1} (k_1 - 1) \). 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:

\[ Z(n, m) = \left[ n/(n+m) \right] Z(n-1, m) + \left[ m/(n+m) \right] [n + Z(n, m-1)], \]

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 \( g^i_j \) and \( f^i_j \) 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. \( \blacksquare \)

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

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

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

4.1.4 The query $R(x, y)$

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

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

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

4.2 The intermediate storage algorithm

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

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 $\beta$ is the label of a node of class $i$ (level $l_i$). Then the probability $p'_x|\beta$ of the event that a given label $\alpha$ is situated on the path from the root of the component tree containing $\beta$ down to $\beta$ is given by $\alpha | \delta - 1 - \lambda(\Delta) - 1$ Furthermore, given that $\alpha$ is situated on the path
4.2.2 The query \( R(x, \beta) \)

Proposition 8. The execution cost \( c^2_R \) under the intermediate storage method is equal to:

\[
c^2_R = \sum_{i=1}^{c} \frac{\text{card}(i)}{n(\Delta)} (l_i + 1)c^2_Q.
\]

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

4.2.3 The query \( R(x, y) \)

Proposition 9. The mean execution cost \( \bar{c}^2_R \) under the intermediate storage algorithm satisfies:

\[
\bar{c}^2_R \sim \bar{c}^2_Q + [n(\Delta) - 1](c^2_R - c^2_Q).
\]

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 \( c^2_R = O(\log n) \) for the full trees whereas \( c^2_R \) 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 \( \gamma^1_k \) for the query \( R(\alpha, y) \) satisfies

\[
\gamma^1_k \sim c^1_Q \sqrt{\frac{\pi n}{2}}.
\]

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 \( 1 \leq k \leq n \). 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 \( 1 \leq k \leq n - 1 \), \( g(n, k+1)/g(n, k) = (n-k)/(kn) \). It is easy to deduce from this the assertions \( g(n) \leq en^{n-1} \) and

\[
g(n) \sim en^{n-1}.
\]
Let us denote by $m$ the size of the subtree $T$ rooted at $\alpha$ and by $V(T)$ its node set; furthermore, for each $k \geq 0$ let us denote by $m_k$ the number of nodes of this subtree belonging to its $k$th
Since \( \epsilon' \) and \( \epsilon'' \) are arbitrarily small for sufficiently big \( q \) it follows that we have

\[
\sum_{i=2}^{n} (l - 1)f(n, l) \sim \pi e^2(n - 1)^{n-1/2}
\]

and, using (13), (15) and (20):

\[
\gamma_{R^2}^{12} \sim \frac{1}{\sqrt{2\pi n}}((c_Q^{12} + c_Q^1) \pi e^2(n - 1)^{n-1/2}),
\]

or

\[
\gamma_{R^2}^{12} \sim (c_Q^{12} + c_Q^1)\sqrt{n}.
\]

Furthermore, by using (16), (17), (20), (21) and after some easy algebra, we deduce that \( \gamma_{R^2}^{12} = o(\gamma_{R^2}^{12}) \).

So, (21) gives the asymptotic expression of \( \gamma_{R^2}^{12} \).

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

Replacing in (8) \( c_R^{12} \) by the value obtained for \( \gamma_{R^2}^{12} \), we get

\[
\gamma_R^2 \sim c_Q^1 + c_Q^2 + (n - 1)\gamma_{R^2}^{12}.
\]

### 5.1.4 The query \( R(x, y) \)

Replacing in (9) \( c_R^1 \) by the value obtained for \( \gamma_R^1 \), we get

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

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

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

\[
\begin{align*}
\gamma_R^1 & \sim c_Q^1 (\pi n^{3/2}) \\
\gamma_R^{12} & \sim (c_Q^{12} + c_Q^1) \pi n^{1/2} \\
\gamma_R^2 & \sim (c_Q^{12} + c_Q^1) \pi n^{3/2} \\
\gamma_R^0 & \sim c_Q^2 \pi n^{3/2}.
\end{align*}
\]

### 5.2 Mean execution costs for the intermediate storage method
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

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

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

**Theorem 2.** With algorithm 2, the mean execution costs $\gamma^2_R$, $\gamma^2_R$, and $\gamma^0_R$ 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

$$
\gamma^2_R \sim (\pi n/2)^{1/2} c^2_Q,
$$

$$
\gamma^2_R \sim (\pi n/2)^{1/2} c^2_Q,
$$

$$
\gamma^0_R \sim (\pi/2)^{1/2} n^{3/2} c^2_Q.
$$

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.


