Computationally efficient sup-t transitive closure for sparse fuzzy binary relations ⋆

Manolis Wallace, Yannis Avrithis and Stefanos Kollias ⋆⋆

School of Electrical and Computer Engineering,
National Technical University of Athens, Greece

Abstract

The property of transitivity is one of the most important for fuzzy binary relations, especially in the cases when they are used for the representation of real life similarity or ordering information. As far as the algorithmic part of the actual calculation of the transitive closure of such relations is concerned, works in the literature mainly focus on crisp symmetric relations, paying little attention to the case of general fuzzy binary relations. Most works that deal with the algorithmic part of the transitive closure of fuzzy relations only focus on the case of max-min transitivity, disregarding other types of transitivity. In this paper, after formalizing the notion of sparseness and providing a representation model for sparse relations that displays both computational and storage merits, we propose an algorithm for the incremental update of fuzzy sup-t transitive relations. The incremental transitive update (ITU) algorithm achieves the re-establishment of transitivity when an already transitive relation is only locally disturbed. Based on this algorithm, we propose an extension to handle the sup-t transitive closure of any fuzzy binary relation, through a novel incremental transitive closure (ITC) algorithm. The ITU and ITC algorithms can be applied on any fuzzy binary relation and t-norm; properties such as reflexivity, symmetricity and idempotency are not a requirement. Under the specified assumptions for the average sparse relation, both of the proposed algorithms have considerably smaller computational complexity than the conventional approach; this is both established theoretically and verified via appropriate computing experiments.

Key words: transitive closure, complexity, sparse matrix, fuzzy partial ordering relations

⋆ This work has been partially funded by the HERACLETUS project of the Greek Ministry of Education

⋆⋆Corresponding author: Manolis Wallace. 14 Aglavrou Str., Koukaki, Athens, 117 41, Attica, GREECE. Email: wallace@image.ntua.gr. Tel.: +30 210 772 3039. Fax.: +30 210 772 2492.

1 Introduction

Fuzzy binary relations and their properties have an important role in modelling of information in various scientific and applied fields. With descriptive power ranging from the simple representation of information [47] to the representation of ontological [28] and multimedia structures [2][34], the fields of application of fuzzy relations are countless. A field in which these relations have a very special role is that of fuzzy information retrieval [10][12][39]. In that framework, fuzzy relational knowledge representation can contribute to tasks such as query expansion [3], document analysis [4][40], multimedia analysis [38], user profiling [41] and others.

In most cases, the fuzzy relation property that is most important for the representation of real life information is that of *transitivity*; continuous *Archimedean* transitivity is a property that describes the propagation of information in a very natural and intuitive way. Therefore, it is only natural that numerous references in the literature discuss the representation [23][26][31] and theoretical properties of transitive binary relations [11][13][15][16][17][18][24][35][45].

The transitive property of binary relations, due to its physical meaning, is closely related to the study of graphs. In that framework, *transitive closure* of a relation is equivalent to the detection of the pairs of vertices that are either directly connected or connected via some path. Thus, the majority of existing literature on transitive closure algorithms and has focused mainly on the cases of undirected crisp graphs [33] and crisp graphs [9][32][36][43], most of which are based on the work of Warshall [44]. In [37], in addition to the computational complexity of the process of the transitive closure, its I/O complexity is examined as well; this study, similarly to the the ones mentioned above, is also limited to the crisp case.

Transitive closure of general fuzzy binary relations has also been treated in the literature. See, for example, [14][27][29]. In the latter two an impressive complexity of $O(n^2)$ is achieved; a result already accomplished in [19] with a different methodology. The application of all four reported algorithms is limited to similarity, i.e. symmetric and reflexive relations, and to the case of max-min transitivity, i.e. to a single non Archimedean case. Overall, not many works in the literature attempt to tackle the sup-t transitive closure of general fuzzy binary relations; most of the attention is focused solely on the case of max-min transitivity. (Exceptions to this can be found in [20][21].) More importantly, there are no references in the literature specializing in the case of *sparse relations*; as the binary relations considered in the fields of ontological representation and fuzzy information retrieval are both sparse and large, the handling of sparse transitive relations is an issue with gaining importance. This is the topic of this paper.

Given the size of the considered binary relations, their representation is an equally important problem to that of their handling. Specifically, the memory required for the storage of an $n \times n$ relation becomes prohibitive, as n surpasses some reasonably small threshold; we may overcome this problem with the utilization of a sparse representation format, since these relations are typically very sparse; this of course adds an overhead to the time required to access a specific element.In this paper we formalize the notion of sparseness and provide a representation model for sparse relations that displays both computational and storage merits. Based on this representation model, we develop an algorithm for the computationally efficient incremental update of fuzzy transitive relations; the incremental transitive update (ITU) algorithm focuses on the re-establishment of transitivity when an already transitive relation is only locally disturbed, and is suitable for any type of sup-t transitivity and any type of fuzzy relation. Based on this algorithm, we propose an extension to handle the sup-t transitive closure of any fuzzy binary relation, through a novel incremental transitive closure (ITC) algorithm; this algorithm is computationally efficient and has some unique properties. Under the specified assumptions for the average sparse relation, both of the proposed algorithms have considerably smaller computational complexity than the conventional approach; this is both established theoretically and verified via appropriate computing experiments.

The structure of this paper is as follows: In section 2, after formally defining what "sparse relation" means in this work, we present the sparse representation model we follow, as the properties of this data model directly affect the properties of the algorithms to follow. Continuing, in section 3 we discuss the properties of the conventional transitive closure algorithm, when combined with the representation model presented in section 2. In sections 4 and 5 we present the two algorithms that we propose. Specifically, in section 4 we present our approach for incremental update of a binary relation and discuss its storage and computational merits and in section 5 we extend our discussion to explain how this can be utilized to handle the complete transitive closure of a relation as well. Section 6 lists some indicative experimental results that support and validate our theory and section 7 lists our concluding remarks.

2 Representation of sparse relations

2.1 Assumptions on sparseness

Binary relations can be used to model numerous aspects of the real world. Depending on the case, the considered relations may display various formal mathematical properties, such as *associativity*, *reflexivity*, *transitivity* of one form or another and so on. The specification of each one of these properties is

an objective task, which makes it feasible to discriminate between the different types of relations and handle them in different ways.

On the other hand, some subjective properties are often important in real life relations. The most important among them is sparseness; sparse relations, although having the same objective properties as their dense counterparts, are "best" handled using totally different approaches. By "best" we do not refer to the validity of the methodologies applied, as their results are identical, but rather to their efficiency as far as storage and computational resources needed are concerned.

As sparse relations play an important role in various fields, specialized data models and corresponding algorithms have been developed just for them; one of the most important problem with such data models and algorithms is their evaluation. Traditionally, the efficiency of a data structure or algorithm is measured using its storage and computational complexity, which is closely related to its operation in the worst possible case. This is of course inapplicable for the case of models and algorithms that have been designed for the sparse case, as, by definition, the notion of *worst case* is contradictory to that of sparseness.

Thus, in order to make the evaluation possible, we typically refer to the performance in the *average*, rather than the worst case scenario. This, in turn, demands that we can formally define the statistical properties of the average case. In this work, driven by the statistical characteristics of sparse relations appearing in the field of ontologies, we define the average case for sparse relations as follows:

Let $n = |S|$ be the cardinality of the universe of discourse. A small and constant percentage p_r of the rows and p_c of the columns of an $n \times n$ *typical sparse relation* may be non zero. Thus, we have $O(n)$ non zero rows and $O(n)$ non zero columns. Additionally, the count of non zero elements contained in a non zero row or column is proportional to the logarithm of the count of all the elements in the relation. Thus we have $O(\log n)$ non zero elements in each non zero row and column. Overall, we have $O(n \log n)$ non zero elements in the relation.

2.2 Proposed sparse representation

A fuzzy binary relation defined on a set S containing n distinct elements can be represented using a square matrix of dimension $n \times n$. The physical storage of such an array requires the representation of n^2 different decimal values, which for large numbers of n is prohibitive for a practical implementation. On the other hand, in such a representation access to a specific element of the array

has a computational complexity of $O(1)$, as the position of the element in the array directly specifies its position in storage as well, with the utilization of a formula of the form $M = [(i-1)\cdot n+(j-1)]\cdot d+O$ where (i, j) is the position of the element in the matrix, M is its actual position in storage, O is an offset specifying the position of the first element of the array in storage and d is the space allocated for each element. As already explained, the utilization of such a representation is not always possible. In cases where the relation is known to be sparse, i.e. only a small subset of the elements of the corresponding array are non zero, then a sparse array implementation can be used to overcome the storage problem.

Conventional sparse array implementations utilize linked lists to represent elements, thus raising the computational complexity of accessing a specific element from $O(1)$ to $O(n)$. In this case, although the storage requirements are much smaller, the representation model remains inapplicable for time critical applications, where complex operations utilizing a binary relation have to be performed before the system response is determined, and number n is large.

The representation model proposed in order to overcome these limitations is as follows: a binary relation is represented using two *AVL trees*; an AVL tree is a binary, balanced and ordered tree that allows for access, insertion and deletion of a node in $O(\log m)$ time, where m is the count of nodes in the tree [1]. If $n \log n$ nodes exist in the tree, as will be the case for the typical sparse relation, then the access, insertion and deletion complexity is again $O(\log n)$ since $n < n \log n < n^2 \Rightarrow O(\log n) \leq O(\log(n \log n)) \leq O(\log n^2) = O(\log n)$.

In both trees, both row index i and column index j are utilized to sort the nodes lexicographically; however, the first tree, the row-tree, is sorted according to index i, and in case of common row positions i, column position j is utilized, and vice versa for the second tree, the column-tree. The following array illustrates this representation:

$$\begin{bmatrix} & (1,2) & & (1,5) & (1,6) \\ (2,1) & & (2,4) & (2,5) & \\ & (3,2) & & & \\ & & (4,4) & & (4,6) \\ (5,1) & & (5,4) & & \end{bmatrix}$$

For this array, elements can be ordered based first on index i and then on index j as follows:

[(1,2), (1,5), (1,6), (2,1), (2,4), (2,5), (3,2), (4,4), (4,6), (5,1), (5,4)]^T

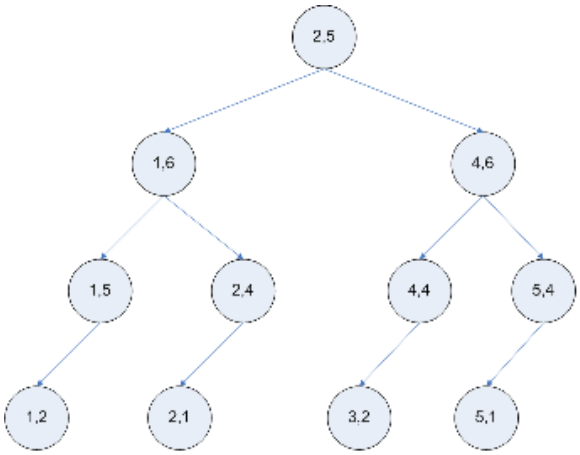


Fig. 1. Example of a row-tree, ordered by index i.

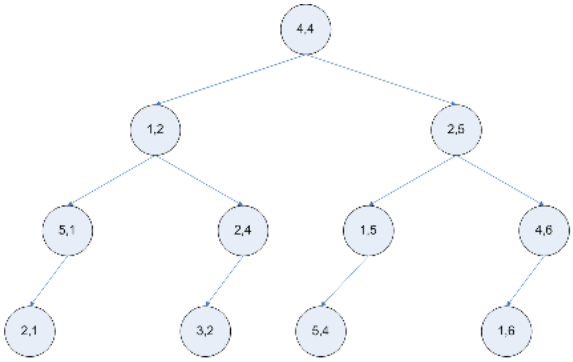


Fig. 2. Example of a column-tree, ordered by index j.

Similarly, elements can be ordered based first on index j and then on index i as follows:

$[(2,1), (5,1), (1,2), (3,2), (2,4), (4,4), (5,4), (1,5), (2,5), (1,6), (4,6)]^T$

The vectors above can then be represented as AVL trees, as shown in figures 1 and 2, depicting the corresponding row-tree and column-tree, respectively. Of course the trees are not unique, as more than one balanced binary trees can be created when the count of elements is not equal to $(2^k - 1)$ for some $k \in \mathcal{N}$.

This representation model preserves the storage merits of the conventional sparse matrix implementation using linked lists. Moreover, *access time* to a specific element, row or column of the relation has a computational complexity of $O(\log n)$, which is considerably lower than the linear complexity of the conventional linked lists approach. Finally, the complexity of *insertion* and *deletion* is also $O(\log n)$; note that both trees have to be kept up-to-date after each insert, delete or update operation.

3 Conventional $sup - t$ transitive closure of fuzzy binary relations

3.1 Conventional algorithm using complete representation

A transitive closure of a fuzzy binary relation can be performed with a complexity of $O(n^4)$, where $n = |S|$ is the dimension of the relation matrix, utilizing the methodology reported in [46], as described below.

In the general case, the transitive closure $Tr^t(R)$ of relation R, given some t-norm, can be calculated as

$$Tr^t(R) = \bigcup_{f=1}^{n-1} R^f \tag{1}$$

where R^f can be calculated recursively as

$$R^f = R^{f-1} \circ^t R \tag{2}$$

$$R^1 = R \tag{3}$$

As a special case, when relation R is reflexive, it is proven that the transitive closure is given by equation

$$Tr^t(R) = R^{n-1}$$

thus making the calculation of the sup of equation 1 unnecessary [25].

The following lemma provides the basis for the calculation of the computational complexities of relation operations using the complete representation model:

Lemma 1: When the complete representation model is utilized, the calculation of the sup $R_{sup} = \sup(R_1, R_2)$ and composition $R_{comp} = R_1 \circ^t R_2$ of two relations R_1 and R_2 of dimension $n \times n$ have computational complexities of $O(n^2)$ and $O(n^3)$, respectively.

Proof: An element $R_{sup}(i, j)$ of the sup is calculated as

$$R_{sup}(i, j) = \sup(R_1(i, j), R_2(i, j))$$

An element $R_{comp}(i,j)$ of the sup-t composition is calculated as

$$R_{comp}(i,j) = \bigcup_{c=1}^{n} R_2(i,c) \wedge_t R_1(c,j)$$

The former is an $O(1)$ operation and the latter an $O(n)$ operation. As the outputs of the sup and composition have a dimension of $n \times n$, n^2 such operations need to be performed, thus concluding in overall complexities of $O(n^2)$ and $O(n^3)$, respectively.

□

From this lemma, it is easy to conclude the following:

Lemma 2: In the case of complete representation, the calculation of the transitive closure of two relations following the methodology of [46] as described above has a computational complexity of $O(n^4)$

Proof: In the calculation of the transitive closure of a reflexive relation, R^{n-1} needs to be calculated. Using the methodology described by equations 2 and 3 this is performed in $n-1$ compositions. The previous lemma provides that each one of these compositions is performed with a computational complexity of $O(n^3)$, thus resulting in an overall complexity of $O(n^4)$ for the calculation of R^{n-1}, which is the transitive closure of relation R.

In the process of calculation of R^{n-1}, all $R^f, f \in \mathcal{N}_{n-1}$ are calculated as byproducts. Thus, for the calculation of the transitive closure in the general (not necessarily reflexive) case what is additionally needed is the calculation of the sup described by equation 1. According to the previous lemma, the sup of two relations is calculated in $O(n^2)$. In total, $n-2$ such calculations are required, resulting in a computational complexity of $O(n^3)$. Thus, the overall computational complexity for transitive closure using this methodology is $O(n^4) + O(n^3) = O(n^4)$.

□

Dunn has proposed a computationally enhanced algorithm to calculate the transitive closure of a fuzzy binary relation, with a complexity of $O(n^3 \log n)$ [20]. This algorithm relies on the *recursive self composition* of the relation matrix until transitivity is achieved.

Specifically, the transitive closure of a relation R is given as:

$$Tr^t(R) = \bigcup R^f_*, f \in \{1, 2, 4, 8, .., 2^F\}, F \geq n-1 \quad (4)$$

where

$$R_*^1 = R$$

$$R_*^f = (R_*^{f/2} \circ^t R_*^{f/2}) \cup R_*^{f/2}, f \in \{2, 4, 8, ..\} \quad (5)$$

The complexity of this approach is given by the following lemma:

Lemma 3: The transitive closure using the methodology of [20] as described above has a computational complexity of $O(n^3 \log n)$.

Proof: In order to calculate the sup described in equation 4 we need to perform $O(\log n)$ relation unions, each one having a complexity of $O(n^2)$, as shown earlier.

Calculating the additives of the sup recursively according to equation 5 starting from the ones corresponding to lower values of f, each one is calculated in one composition and one addition, thus with a complexity of

$$O(n^3) + O(n^2) = O(n^3)$$

This calculation will end when we have reached a value of f such that $F \geq n-1$ holds. Thus, $O(\log n)$ such calculations are adequate. Overall, this results in a complexity of

$$O(\log n) \cdot O(n^2) + O(\log n) \cdot O(n^3) = O(n^3 \log n)$$

□

Dunn's extension is considered as the classical approach to the sup-t transitive closure of fuzzy binary relations. In the remaining of the paper, term *conventional algorithm* for transitive closure will refer to Dunn's extension.

The sup-t transitive closure of fuzzy binary relations can be calculated in as little as $O(n^3)$ time using more efficient algorithm implementations [30]:

Algorithm $O(n^3)$ sup-t:

Parameters: R

Output: R

(1) for $i = 1 \ldots n$
  (a) for $j = 1 \ldots n$
    (i) for $k = 1 \ldots n$

$$R(j,k) \leftarrow \sup(R(j,k), R(j,i) \wedge_t R(i,k))$$

The original formulation of this algorithm can be found in [21] and is based on the use of selectors [8] and the star decomposition rule [6][7]. This algorithm, unlike Dunn's approach, cannot be extended to efficiently support a sparse representation model, and is thus inadequate to provide for meaningful comparisons against the methodology developed in this work. Therefore, in this work we shall consider the conventional approach in all comparative study experiments and comments.

3.2 Conventional algorithm using sparse representation

As we have already mentioned, the characteristics of the representation model utilized reflect on the characteristics of the applied algorithms as well. Thus, when it comes to the conventional algorithm of transitive closure, the following holds:

Theorem 1: The complete transitive closure is achieved with computational complexity $O(n^2 \log^2 n)$ and $O(n^3 \log n)$ in the average and worst case, respectively, when using the proposed sparse representation.

Proof: During a step of relation composition in the sparse representation case, if row i and column j exist in the relation, i.e. if they have at least one non zero element, they are retrieved. As already explained, retrieving a row or column has a complexity of $O(\log n)$. Continuing, the corresponding element $R_{comp}(i,j)$ is calculated as

$$R_{comp}(i,j) = \bigcup_{\substack{R(i,c) \in r_i \\ R(c,j) \in c_j}} R(i,c) \wedge_t R(c,j)$$

where r_i and c_j are the sets of non zero elements of the i-th row and j-th column, respectively. As the row and column are both available in a sorted by index form, this is an $O(|r_i| + |c_j|)$ operation, where $|r_i|$ is the count of non zero elements in row i and $|c_j|$ is the count of non zero elements in column j.

For a single composition $k_r \cdot k_c$ such operations will be performed, where k_r is the count of non zero rows and k_c is the count of non zero columns of the relation. Overall, the complexity for a single composition is

$$O(\log n) + O(\log n) + O(k_r \cdot k_j) \cdot [O(|r_i| + |c_j|) + O(\log n)]$$

where the last $O(\log n)$ factor corresponds to the complexity of the insertion of an element in the output relation.

In the average case for sparse relations, a small percentage of the rows and columns will be non zero. Of course, although this may affect the execution time required, it does not alter the complexity, as

$$O(k_r \cdot k_c) = O((p_r \cdot n) \cdot (p_c \cdot n)) = (p_r \cdot p_c) \cdot O(n^2)) = O(n^2)$$

where constants p_r and p_c is the percentage of non zero rows and columns respectively. As far as the count of elements contained in a non zero row or column is concerned, as has already been mentioned in subsection 2.1, this is assumed to be proportional to the logarithm of the count of all elements in the relation. Thus $O(k_r \cdot k_j) = O(\log n)$. Overall, this leads to a complexity of

$$O(\log n) + O(\log n) + O(n^2) \cdot [O(\log n) + O(\log n)] = O(n^2 \log n)$$

for the composition.

In the worst case, all the elements of the relation exist. In that case, $k_r = k_c = |r_i| = |c_j| = n$, and thus the complexity of the composition becomes

$$O(\log n) + O(\log n) + O(n^2) \cdot [O(n) + O(\log n)] = O(n^3)$$

Considering the $O(\log n)$ compositions required by the conventional methodology for the transitive closure, it is straightforward that in the average case the complete transitive closure has a computational complexity of $O(n^2 \log^2 n)$ and in the worst case $O(n^3 \log n)$.

□

3.3 Comparative study

As far as the computational complexity of the transitive closure algorithm is concerned, it is $O(n^3 \log n)$ for the complete representation and $O(n^2 \log^2 n)$

Table 1
Summary of computational complexities of conventional composition and transitive closure algorithms

Algorithm	Data model	Sparse relation	Dense relation
Composition	Complete	n^3	n^3
Composition	Sparse	$n^2 \log n$	n^3
Transitive Closure	Complete	$n^3 \log n$	$n^3 \log n$
Transitive Closure	Sparse	$n^2 \log^2 n$	$n^3 \log n$

and $O(n^3 \log n)$ for the sparse representation, in the average and worst case, respectively. It is worth noting that the proposed sparse representation achieves enhanced computational complexity in the sparse case, without loosing in efficiency in the dense case.

As far as storage requirements are concerned, using either of the two representation models, i.e. complete representation or the sparse representation proposed herein, the existence of two copies of the relation in memory during the execution of the transitive closure algorithm is required. The storage requirements of the two approaches, though, are fundamentally different in the case of sparse relations:

(1) In the conventional approach n^2 elements are represented for each copy of the relation, leading to the need for $2 \cdot n^2$ distinct elements being represented. This is an $O(n^2)$ space.
(2) In the proposed approach 2 trees are represented, each containing as many nodes as the relation at hand. In the average case of sparse relations this results in $2 \cdot O(n \log n)$ nodes, leading to an overall storage complexity of $4 \cdot O(n \log n) = O(n \log n)$ for the two trees. In the worst case $O(n^2)$ storage space is required, as in the conventional approach.

Table 1 summarizes the above conclusions on computational complexity of conventional composition and transitive closure algorithms, using complete and sparse representations. One can easily see that the proposed representation model is ideal for the case of sparse relations, as they have been defined in section 2.1, as it can lead to enhanced computational and storage complexities, even when using algorithms that have not been designed especially for the sparse case.

4 Incremental closure of fuzzy binary relations

A major disadvantage of the utilization of the conventional transitive closure methodologies is that when for some reason an element of the relation is

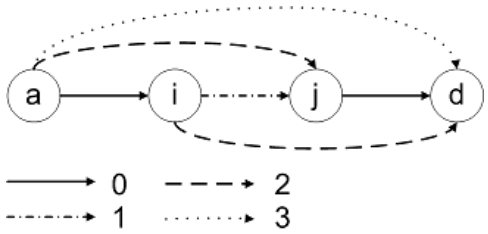


Fig. 3. Graphical representation of the incremental update of the transitive relation.

updated, or when a new entity is inserted to the global set S and linked to some existing entity, thus *locally disturbing* transitivity, a computationally demanding operation needs to be performed again in order to re-establish the transitivity property.

4.1 Conventional re-establishment of transitivity

Theorem 2: When a new element is inserted to the relation, or when an element of the relation is updated, and assuming the conventional approach to transitive closure, an $O(n^2 \log n)$ operation is adequate in order to assure that the relation remains transitive in the average case. In the worst case the complexity is $O(n^3 \log n)$.

Proof: Let R be a transitive relation. In Fig. 3 non zero elements of R are represented using continuous lines of type 0. Let as suppose that $R(i, j)$ is inserted in the relation, or that its value is augmented. In Fig. 3 the update is represented using dash - dot lines of type 1. Then, we can no longer assume that relation r is transitive.

After one self-composition, the ancestor a of i is linked to j and i is linked to the descendant d of j. In Fig. 3 this is represented using dashed lines of type 2. Finally, after one more self-composition, a is linked to b. In Fig. 3 this is represented using doted lines of type 3.

As R was initially assumed transitive, two compositions will always be enough to assure transitivity. Thus, the complexity of the operation that re-establishes transitivity can be as low as that of the composition: $O(n^2 \log n)$ in the average case and $O(n^3)$ in the worst case, as established in the theorem's proof in section 3.

□

4.2 The incremental transitive update (ITU) algorithm

Having observed in Fig. 3 the way in which transitivity is achieved after a single element has been altered, we can design an algorithm for incremental update of transitive relations that has a considerably smaller computational complexity. Specifically, the incremental transitive update (ITU) algorithm only focuses on the changes that the self composition brings upon the relation after the update of the single element.

For a given relation R, when the updated element is the one between entities i and j, this can be achieved with the following steps:

Algorithm ITU:

Parameters: $R\ i\ j$

Output: R

(1) Identify the fuzzy set A of ancestors of entity i in relation R. Degrees in A are determined as $A(s) = R(s, j), s \in S$.
(2) Identify the fuzzy set D of descendants of entity j in relation R. Degrees in D are determined as $D(s) = R(i, s), s \in S$.
(3) For each element s appearing in A assign

$$R(s, j) \leftarrow \sup(R(s, j), A(s) \wedge_t R(i, j))$$

(4) For each element s appearing in D assign

$$R(i, s) \leftarrow \sup(R(i, s), R(i, j) \wedge_t D(s))$$

(5) For each element s_1 appearing in A and s_2 appearing in D assign

$$R(s_1, s_2) \leftarrow \sup(R(s_1, s_2), A(s_1) \wedge_t R(i, j) \wedge_t D(s_2))$$

When the algorithm terminates we have $R = Tr^t(R)$.

If relation R is reflexive, then $R(i, i) = 1$ and $R(j, j) = 1$ and thus $A(i) = 1$ and $D(j) = 1$. In this case, the above process can be simplified by omitting steps (3) and (4), as they are included in step (5).

Theorem 3: The computational complexity of the incremental update algorithm is $O(\log^3 n)$ in the average case and $O(n^2 \log n)$ in the worst case, for both reflexive and non reflexive relations, assuming the proposed sparse representation model.

Proof: The complexity of steps (1) and (2) is $O(\log n)$, as computation of an

element's ancestors or descendants requires access to a column or row of the relation, respectively. The complexity of steps (3) and (4) is $O(|r_i|) \cdot O(\log n)$ and $O(|c_j|) \cdot O(\log n)$, respectively, and the complexity of step (5) is $O(|r_i| \cdot |c_j|) \cdot O(\log n)$.

In the average case $|r_i| = O(\log n)$ and $|c_j| = O(\log n)$, and thus the overall complexity is $2 \cdot O(\log n) + 2 \cdot O(\log^2 n) + O(\log^3 n) = O(\log^3 n)$. In the worst case $|r_i| = O(n)$ and $|c_j| = O(n)$, and thus the overall complexity is $2 \cdot O(\log n) + 2 \cdot O(n \log n) + O(n^2 \log n) = O(n^2 \log n)$. Ignoring the influence of steps (3) and (4) in the above calculations does not alter the overall complexity, as in all cases step (5) contributes more to it. Thus, the same complexity holds for the reflexive case as well.

□

Of course, it is easy to extend algorithm ITU in order to make it applicable to the case of complete representation as well:

Algorithm ITU for complete representation:

Parameters: R i j

Output: R

(1) For every element s_1 in column i
  (a) For every element s_2 in row j
    Assign:

$$R(s_1, s_2) \leftarrow \sup(R(s_1, s_2), R(s_1), i \wedge_t R(i, j) \wedge_t R(j, s_2)) \tag{6}$$

Theorem 4: The computational complexity of the incremental update algorithm is $O(n^2)$, assuming a complete representation model.

Proof: When using the complete representation model, the relation is represented as an $n \times n$ array. Thus, there are n elements in each row and each column. Since equation 6 describes simply two intersections, it is performed in $O(1)$ time. Overall, we have a complexity of

$$O(n) \cdot O(n) \cdot O(1) = O(n^2)$$

□

What remains to be established is that the output of the proposed ITU algorithms is indeed transitive. This proof is spit in two sections:

Lemma 4: If relation R is $\sup -t$ transitive on S, then it is also $\sup -t$

transitive on $S' \supseteq S$.

Proof: A fuzzy relation R defined on S is called $\sup -t$ transitive if

$$R(x,z) \geq \sup_{y \in S}\{t(R(x,y),R(y,z))\}, \forall (x,z) \in S^2$$

If $x \in (S' - S)$, then $R(x,z) = 0$. In this case

$$\sup_{y \in S'}\{t(R(x,y),R(y,z))\} = \sup_{y \in S'}\{0,R(y,z))\} = 0$$

and thus transitivity on S' holds. Similarly if $z \in (S' - S)$.

If $(x,z) \in S^2$, then

$$\sup_{y \in S'}\{t(R(x,y),R(y,z))\} =$$

$$= \sup(\sup_{y \in S}\{t(R(x,y),R(y,z))\}, \sup_{y \in (S'-S)}\{t(R(x,y),R(y,z))\})$$

$$= \sup(\sup_{y \in S}\{t(R(x,y),R(y,z))\}, 0)$$

$$= \sup_{y \in S}\{t(R(x,y),R(y,z))\}$$

Since R is transitive on S,

$$R(x,z) \geq \sup_{y \in S}\{t(R(x,y),R(y,z))\} = \sup_{y \in S'}\{t(R(x,y),R(y,z))\}$$

and thus transitivity holds.

□

With this lemma we have established that extending the universe with the addition of new elements, as the ITU does when needed, does not damage the property of transitivity for the already existing relations. We may now split the operation of ITU in two steps:

(1) insert new elements in the universe of discourse (if needed)
(2) augment the value of a link between existing elements

According to lemma 4, only the second one of these steps affects transitivity and needs to be considered. Thus, we can limit our examination of the validity

of the ITU algorithm to the case where an existing link of the relation is augmented (the initial link can have any value in the [0 1] range). With the next lemma we conclude the proof by establishing that ITU correctly re-establishes transitivity what an existing link in the input relation is augmented.

Lemma 5: Let R be a transitive relation on S. Let also $i, j \in S$. $\forall q \in [01]$, relation R_1 is defined as

$$R_1(x,z) = \begin{cases} \max(R(i,j),q), & \text{x=i and z=j;} \\ R(x,z), & \text{otherwise.} \end{cases}$$

Then relation R', calculated as $R' = ITU(R_1, i, j)$, is transitive.

Proof:

If $R(i,j) \geq q$ from construction $R' = R$ and thus transitivity holds. If $R(i,j) < q \Rightarrow R'(i,j) > R(i,j)$. We then we need to prove that

$$R'(x,z) \geq \sup_{y \in S} t(R'(x,y), R'(y,z)), \forall (x,z) \in S^2 \tag{7}$$

For y such that $R'(x,y) = R(x,y)$, easily

$$\sup_{y:R'(x,y)=R(x,y)} t(R'(x,y), R'(y,z)) =$$

$$= \sup_{y:R'(x,y)=R(x,y)} t(R(x,y), R(y,z))$$

$$\leq R(x,y)$$

$$\leq R'(x,y)$$

Focusing on the remaining cases of y such that $R'(x,y) > R(x,y)$, the proof is based on the observation that the ITU algorithm only affects specific elements of the relation.

Let $X = {}_{0+}A$. This is the strong 0-cut of fuzzy set A of ancestors of i and contains elements of S that participate in A to non zero degrees. Similarly, let $Z = {}_{0+}D$.

If $x \in X \cup \{i\}$ and $z \in Z \cup \{j\}$, then eq. 7 holds by construction.

If $x \notin X$ and $z \notin Z$, then by construction we have that $R'(x,y) = R(x,y)$ and $R'(y,z) = R(y,z)$. Thus

$$\sup_{y:R'(x,y)>R(x,y)} t(R'(x,y), R'(y,z)) =$$

$$= \sup_{y:R'(x,y)>R(x,y)} t(R(x,y), R(y,z))$$

$$\leq R(x,z)$$

$$= R'(x,z)$$

If $x \in X \cup \{i\}$ and $z \notin Z$, then by construction we have that $R'(y,z) = R(y,z)$ and that $sup_{y:R'(x,y)>R(x,y)} t(R(j,y), R(y,z)) = 0$. If $R'(x,y) = R(x,y)$ eq. 7 obviously holds. If $R'(x,y) > R(x,y)$, then from construction $R'(x,y) = t(R(x,i), R_1(i,j), R(j,y))$. Then

$$\sup_{y:R'(x,y)>R(x,y)} t(R'(x,y), R'(y,z)) =$$

$$= \sup_{y:R'(x,y)>R(x,y)} t(R(x,i), R_1(i,j), R(j,y), R(y,z))$$

$$\leq \sup_{y:R'(x,y)>R(x,y)} t(R(j,y), R(y,z))$$

$$= 0$$

because $z \notin Z$. Thus eq. 7 holds. Similarly if $x \notin X$ and $z \in Z$. If $x = i$ and $z \notin Z$ the proof follows closely the steps of the previous case. Similarly if $x \notin X$ and $z = j$, which concludes all cases. Thus R' is transitive.

□

4.3 Numerical example

In this subsection we provide a numerical example of the application of the ITU algorithm, in order to best explain its operation. As input we assume the transitive relation R_{input} of table 2. The assumed t-norm for the sup $-t$ transitivity is the bounded difference. The element added to the relation is (#9,#6,0.95). In the following we explain the effect that each one of the steps of algorithm ITU has on the initial relation R_{input} so that relation R_{output} is acquired.

(1) Fuzzy set A is the #9 column.

Table 2
Initial transitive relation R_{input}

	#1	#2	#3	#4	#7	#8	#9
#1	0.80	0.95	0.90	0.85	0.85	0.90	0.80
#2	0.85	0.80	0.95	0.90	0.90	0.95	0.85
#3	0.90	0.85	0.80	0.95	0.75	0.80	0.90
#4	0.95	0.90	0.85	0.80	0.80	0.85	0.95
#5	0.85	0.80	0.95	0.90	0.70	0.75	0.85
#6					0.95	0.90	
#7					0.90	0.95	
#8					0.95	0.90	

Table 3
Output relation R_{output} of ITU after the addition of element (#9,#6,0.95)

	#1	#2	#3	#4	#6	#7	#8	#9
#1	0.80	0.95	0.90	0.85	0.75	0.85	0.90	0.80
#2	0.85	0.80	0.95	0.90	0.80	0.90	0.95	0.85
#3	0.90	0.85	0.80	0.95	0.85	0.80	0.80	0.90
#4	0.95	0.90	0.85	0.80	0.90	0.85	0.85	0.95
#5	0.85	0.80	0.95	0.90	0.80	0.75	0.75	0.85
#6						0.95	0.90	
#7						0.90	0.95	
#8						0.95	0.90	
#9					0.95	0.90	0.80	

(2) Fuzzy set D is the #6 row.
(3) #6 column is created.
(4) #9 row is created.
(5) #3 → #7, #4 → #7 and #5 → #7 elements are updated.

4.4 Comparative study

In table 4 we present a summary of the computational complexities of the transitive closure re-establishment approaches mentioned herein.

In the case of the average sparse relation, the proposed ITU algorithm outperforms the conventional one, for both representation models. It is worth

Table 4
Summary of computational complexities of transitive closure re-establishment algorithms

Algorithm	Data model	Sparse relation	Dense relation
Conventional	Complete	n^3	n^3
Conventional	Sparse	$n^2 \log n$	n^3
ITU	Complete	n^2	n^2
ITU	Sparse	$\log^3 n$	$n^2 \log n$

mentioning, though, that only the combination of the proposed model and algorithm reaches a *sub-linear* complexity, when the conventional approach combined with a complete representation has a complexity of $O(n^3)$. In the case of dense relations, the combination of the proposed model and ITU algorithm reaches a complexity of $O(n^2 \log n)$, when the conventional approach combined with a complete representation has a complexity of $O(n^3)$.

Even in the case that the complete representation model is followed, the ITU algorithm outperforms the conventional one, having a complexity of $O(n^2)$, compared to a complexity of $O(n^3)$.

Finally, the proposed ITU algorithm is more efficient as far as the storage requirements are concerned when compared to the conventional approach, when using either representation model, due to the fact that it does not require the storage of two copies of the relation.

5 Complete transitive closure of fuzzy binary relations

The ITU algorithm for incremental update of transitive fuzzy binary relations presented in the previous section easily leads to the design of an algorithm for a complete transitive closure of a relation as well. The proposed incremental transitive closure (ITC) algorithm is described next, along with a theoretical study on its computational complexity and comparison to the conventional algorithm.

5.1 The incremental transitive closure (ITC) algorithm

Algorithm ITC:

Parameters: R

Output: R'

(1) Create an empty binary relation R'.
(2) For each non zero element $R(i,j)$ in the initial relation R
  (a) Assign

$$R'(i,j) \leftarrow \sup(R'(i,j), R(i,j))$$

  (b) Run the incremental update algorithm with parameters R' i j:

$$R' \leftarrow ITU(R', i, j)$$

When the algorithm terminates we have $R' = Tr^t(R)$.

Theorem 5: The computational complexity of proposed ITC algorithm is $O(n \log^4 n)$ in the average case and $O(n^4 \log n)$ in the worst case, assuming the proposed sparse representation model.

Proof: Step (1) is obviously completed in an $O(1)$ operation. Step (2a) is executed in an $O(\log n)$ operation, while the complexity of step (2b) is as described in the previous section. Step (2) is executed as many times, as is the count of elements in relation r.

In the average case, there are $O(n)$ rows in r, each one containing $O(\log n)$ elements, thus resulting in $O(n \log n)$ elements in r, and the complexity of step 2b is $O(\log^3 n)$. Thus, the overall complexity is

$$O(1) + O(n \log n) \cdot (O(\log n) + O(\log^3 n)) = O(n \log^4 n)$$

In the worst case, the count of elements in r is n^2, and step 2b has a complexity of $O(n^2 \log n)$, thus resulting in

$$O(1) + O(n^2) \cdot (O(\log n) + O(n^2 \log n)) = O(n^4 \log n)$$

□

5.2 Numerical example

In this subsection we provide a step-by-step demonstration of the operation of the ITC algorithm. The relation upon which the algorithms is be applied is presented in Fig. 4, where drawn links have a weight of 0.95, while all other links have a weight of 0. It is worth noting that in the provided sample relation loops exist (see #1 → #2 → #3 → #4 → #1 as well as #7 → #8 → #1).

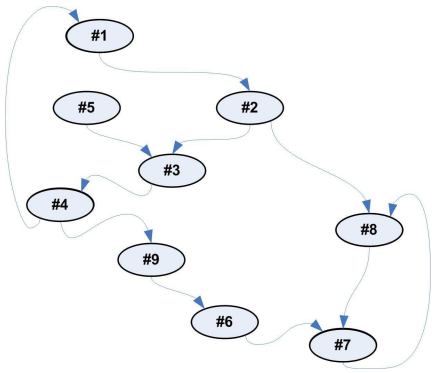
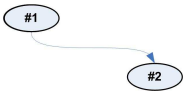
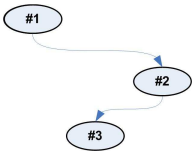


Fig. 4. The sample fuzzy relation



	#2
#1	0.95

Fig. 5. Added element (#1,#2,0.95)



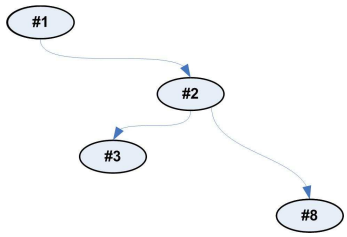
	#2	#3
#1	0.95	0.90
#2		0.95

Fig. 6. Added element (#2,#3,0.95)

Moreover, the relation is not a reflexive one, which is a requirement for most known transitive closure algorithms. Finally, the bounded difference t-norm is chosen for the property of transitivity.

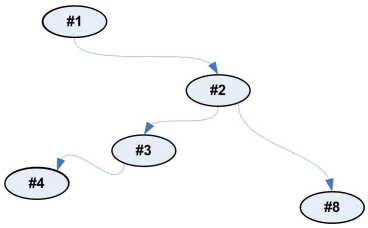
As explained in section 5, in order to construct the transitive closure of the provided relation we start from an empty relation and gradually built the closure by repeatedly applying the ITU algorithm. In the first step we add the link between elements #1 and #2 (see Fig. 5); in the figures of this subsection, on the left we present the links already presented to the ITC algorithm, while on the right we present the resulting temporary relation; this is the relation that will hold the transitive closure when the algorithm terminates.

In the next step the link between #2 and #3 is added (see Fig. 6), while ITU also adds the link between #1 and #3. The reader is invited to follow the application of ITC and ITU through figures 7, 8, 9, 10, 11, 12, 13, 14 and 15. Worthy of mentioning is the step when the link between element #4 and element #1 is added, where we can see how the algorithm successfully handles the generated loop.



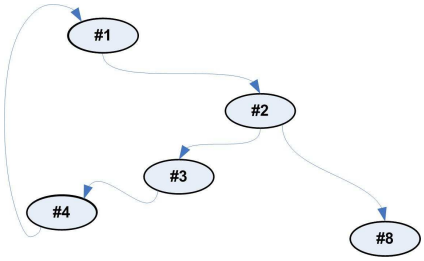
	#2	#3	#8
#1	0.95	0.90	0.90
#2		0.95	0.95

Fig. 7. Added element (#2,#8,0.95)



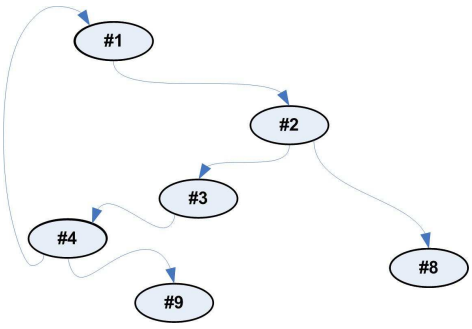
	#2	#3	#4	#8
#1	0.95	0.90	0.85	0.90
#2		0.95	0.90	0.95
#3			0.90	

Fig. 8. Added element (#3,#4,0.95)



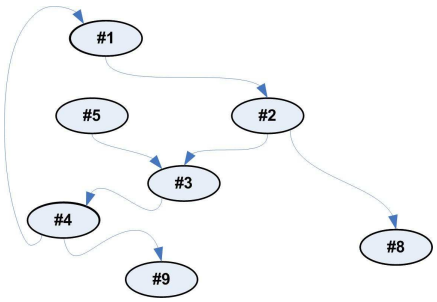
	#1	#2	#3	#4	#8
#1	0.80	0.95	0.90	0.85	0.90
#2	0.85	0.80	0.95	0.90	0.95
#3	0.90	0.85	0.80	0.95	0.80
#4	0.95	0.90	0.85	0.80	0.85

Fig. 9. Added element (#4,#1,0.95)



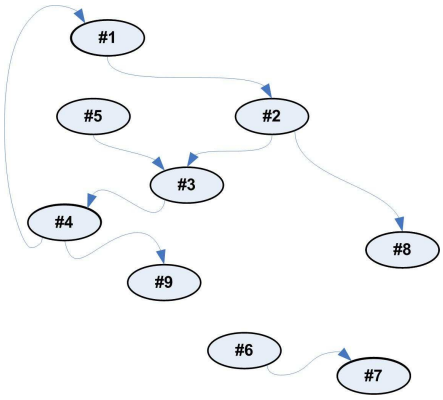
	#1	#2	#3	#4	#8	#9
#1	0.80	0.95	0.90	0.85	0.90	0.80
#2	0.85	0.80	0.95	0.90	0.95	0.85
#3	0.90	0.85	0.80	0.95	0.80	0.90
#4	0.95	0.90	0.85	0.80	0.85	0.95

Fig. 10. Added element (#4,#9,0.95)



	#1	#2	#3	#4	#8	#9
#1	0.80	0.95	0.90	0.85	0.90	0.80
#2	0.85	0.80	0.95	0.90	0.95	0.85
#3	0.90	0.85	0.80	0.95	0.80	0.90
#4	0.95	0.90	0.85	0.80	0.85	0.95
#5	0.85	0.80	0.95	0.90	0.75	0.85

Fig. 11. Added element (#5,#3,0.95)



	#1	#2	#3	#4	#7	#8	#9
#1	0.80	0.95	0.90	0.85		0.90	0.80
#2	0.85	0.80	0.95	0.90		0.95	0.85
#3	0.90	0.85	0.80	0.95		0.80	0.90
#4	0.95	0.90	0.85	0.80		0.85	0.95
#5	0.85	0.80	0.95	0.90		0.75	0.85
#6					0.95		

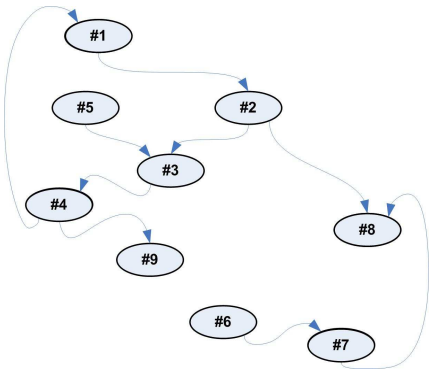
Fig. 12. Added element (#6,#7,0.95)

Table 5
Summary of computational complexities of complete transitive closure algorithms

Algorithm	Data model	Sparse relation	Dense relation
Conventional	Complete	$n^3 \log n$	$n^3 \log n$
Conventional	Sparse	$n^2 \log^2 n$	$n^3 \log n$
ITC	Complete	-	-
ITC	Sparse	$n \log^4 n$	$n^4 \log n$

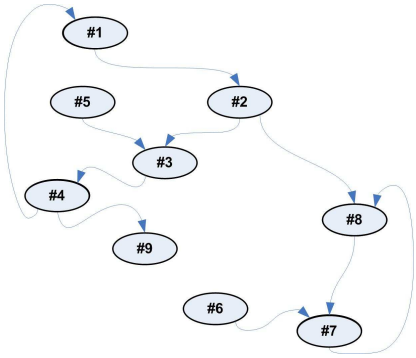
5.3 Comparative study

Table 5 summarizes the computational complexities of the conventional approach to transitive closure and the proposed ITC algorithm. In the case where



	#1	#2	#3	#4	#7	#8	#9
#1	0.80	0.95	0.90	0.85		0.90	0.80
#2	0.85	0.80	0.95	0.90		0.95	0.85
#3	0.90	0.85	0.80	0.95		0.80	0.90
#4	0.95	0.90	0.85	0.80		0.85	0.95
#5	0.85	0.80	0.95	0.90		0.75	0.85
#6					0.95	0.90	
#7						0.95	

Fig. 13. Added element (#7,#8,0.95)

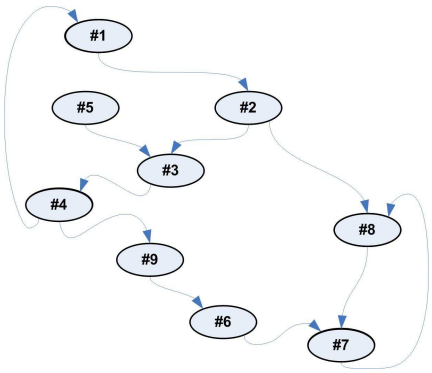


	#1	#2	#3	#4	#7	#8	#9
#1	0.80	0.95	0.90	0.85	0.85	0.90	0.80
#2	0.85	0.80	0.95	0.90	0.90	0.95	0.85
#3	0.90	0.85	0.80	0.95	0.75	0.80	0.90
#4	0.95	0.90	0.85	0.80	0.80	0.85	0.95
#5	0.85	0.80	0.95	0.90	0.70	0.75	0.85
#6					0.95	0.90	
#7					0.90	0.95	
#8					0.95	0.90	

Fig. 14. Added element (#8,#7,0.95)

the binary relation in question is dense, the conventional approach is better, having a complexity of $O(n^3 \log n)$, compared to a complexity of $O(n^4 \log n)$ for the ITC algorithm.

In the average case, on the other hand, the ITC algorithm is greatly superior, having a complexity of $O(n \log^4 n)$; the difference between this and the complexity of $O(n^3 \log n)$ for the conventional approach, or even the complexity of $O(n^2 \log^2 n)$ when exploiting the proposed sparse representation, is huge. It is worth mentioning that the complexity of the proposed approach in the case



	#1	#2	#3	#4	#6	#7	#8	#9
#1	0.80	0.95	0.90	0.85	0.75	0.85	0.90	0.80
#2	0.85	0.80	0.95	0.90	0.80	0.90	0.95	0.85
#3	0.90	0.85	0.80	0.95	0.85	0.80	0.80	0.90
#4	0.95	0.90	0.85	0.80	0.90	0.85	0.85	0.95
#5	0.85	0.80	0.95	0.90	0.80	0.75	0.75	0.85
#6						0.95	0.90	
#7						0.90	0.95	
#8						0.95	0.90	
#9					0.95	0.90	0.80	

Fig. 15. Added element (#9,#6,0.95)

of the average sparse binary relation is below quadratic complexity.

Since the conventional approach is better in the worst case, while the proposed ITC algorithm is considerably better in the average case, the value of the proposed ITC algorithm greatly depends on the validity of the assumptions made concerning the average case. In other words, in all cases where the assumptions stated in subsection 2.1 hold, the proposed approach is by far preferable. The main assumption stated in subsection 2.1 is that considerably less non zero elements exist in the average case; $O(n \log n)$ instead of $O(n^2)$. In many real life problems handled using fuzzy binary relations this assumption holds; ontological knowledge representations used in inferencing engines and information retrieval applications are such examples.

When it comes to storage requirements, the conventional approach requires that two copies of the relation exist in the memory at the same time, thus doubling the storage requirements of the algorithm. On the other hand, space for a single relation is enough for the execution of the ITC algorithm; two relations exist, but as elements are added in one, they are removed from the other.

In addition to the reduced computational complexity and storage requirements, the proposed ITC approach to complete transitive closure of fuzzy binary relations also has the following merits:

(1) In the conventional approach, a composition of the relation, and thus an operation of complexity $O(n^2 \log n)$ in the average case, has to be per-

formed before the decision to terminate the algorithm is taken. This is true even when the relation is initially transitive, thus requiring no adjustment. In the proposed approach, in the same situation the algorithm would terminate in $O(n \log^4 n)$ time.

(2) In the conventional approach, the relation is not transitive until the operation terminates. Thus, in the case of a very large number n, a very long time has to be spent before the relation becomes usable by algorithms that assume transitivity. On the contrary, relation R' is *transitive after each step* when utilizing the ITC algorithm, and thus algorithms that assume it to be transitive can be applied to it before the closure operation is completed.

(3) The termination point of the conventional approach greatly depends on the "depth" of the relation, i.e. on the count of vertices of the longest path. Specifically, it may take the process anywhere between 1 and $\log n$ iterations to terminate. On the contrary, the ITC algorithm assures transitivity in one pass and thus the count of required steps is known beforehand. Therefore, an *indication of progress* of the process is available on-line, if we require it, in the form of percentage completed.

(4) The proposed approach is not affected by cycles in the relation, i.e. it does not require that the relation is ordering. This is is not a property shared by all specialized transitive closure algorithms [37].

(5) The computational complexity for the complete transitive closure in the sparse case is very close to the computational complexity of loading the relation from a storage location such as a hard disk; $O(n \log^4 n)$ compared to $O(n \log n)$. Consequently, we have the option to store the relation in a non transitive form, thus saving in disk space, and calculate the transitive closure at the time of loading using the ITC algorithm, only slightly affecting the overall complexity and execution time.

6 Experimental results

In this section we provide experimental results from the application of the conventional methodologies and ITU and ITC algorithms for transitivity reestablishment and complete transitive closure on a synthetic data set, as well as on a real life data set from the field of knowledge based information retrieval [5][39]. Implementation of the proposed relation representation model, as well as of the conventional and proposed algorithms, has been done using the Java environment and execution has been performed on a PC (Centrino 1.6GHz, 256MB RAM) with Windows XP operating system. The source code of these experiments, modified as to provide detailed step by step output, is freely available at [48].

6.1 Data sets

In knowledge based information retrieval, the retrieval system needs to utilize all or a part of the available knowledge when processing e.g. a user query, a user profile or a document, in order to provide a meaningful response. In cases where a fuzzy relational knowledge representation model is followed, having the fuzzy relations readily available in a closed transitive form typically alleviates the need for recursion in the processing algorithms, thus greatly reducing computational complexity and processing time.

In this paper we utilize such a relation in order to experimentally validate our theory. Specifically, the universe of discourse $S_{90,000}$ is the set of all concepts defined in *WordNet* for the English language verbs and nouns [22]; there is a 1-1 mapping between elements of this set and the verb and noun synsets defined in WordNet. The cardinality of this set is slightly over 90,000 elements.

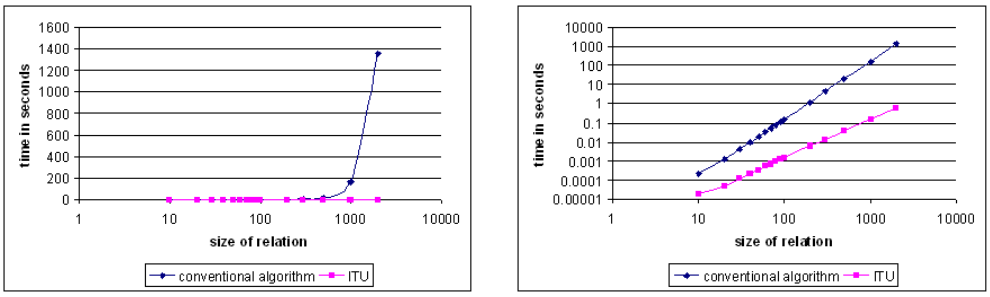
A fuzzy binary relation on $S_{90,000}$ is generated automatically, again using WordNet as a source. Two of the lexical relations, *hyponym* and *part meronym*, are used to specify the pairs of connected elements in the relation. As these relations are crisp in WordNet, degree 0.9 is assigned to all such pairs by default. Additionally, the relation is made reflexive. Overall, around 110,000 pairs are connected to a degree 0.9 and 90,000 more elements to degree 1 due to reflexivity. This forms fuzzy binary relation $R_{90,000}$.

By randomly selecting elements from set $S_{90,000}$ we form crisp set $S_{50,000}$. This set contains 50,000 elements. Keeping only the rows and columns of $R_{90,000}$ that correspond to elements in $S_{50,000}$ we construct relation $R_{50,000}$. This is similar in structure and content with fuzzy binary relation $R_{90,000}$, but smaller in size; the comparative study of algorithms' performance on such relations provides for more intuitive evaluation of their complexities. Recursively, $S_{20,000}$ is constructed from $S_{50,000}$, $S_{10,000}$ is constructed from $S_{20,000}$ and so on, thus resulting in a wide range of corresponding binary relations: R_n, $n \in \{90000, 50000, 20000, 10000, 5000, 3000, 2000, 1000, 500\}$.

Fuzzy binary relation R^t_{90000}, obtained using algorithm ITC, is the sup-t transitive closure of $R_{90,000}$, where t is the bounded sum t-norm. R^t_{90000} contains 760,000 non zero elements; 90,000 elements having degree 1 due to reflexivity, 110,000 elements with degree 0.9 also existing in the original $R_{90,000}$ and 560,000 elements with lesser degrees, produced during transitive closure. Similarly, we have constructed transitive relations R^t_n, all following the rule:

$$R^t_n = Tr^t(R_n) \tag{8}$$

The utilization of the complete representation model for $R_{90,000}$ or other re-



A B

Fig. 16. Execution times for 2 compositions and for ITU on R_n^d using complete representation.

lations having a universe of discourse of considerable dimension is not practically possible. For the case of $R_{90,000}$, for example, the relation dimension of $90,000 \times 90,000$ requires the representation of approximately 8 billion double precision numbers. With 8 bytes allocated for each such number, the required memory space just for one copy of the relation is 65Gb of RAM. Thus, in all subsequent experiments only the proposed sparse representation model is considered for this series of data sets.

In order to experimentally verify the efficiency of the proposed methodologies when dealing with dense relations, or when combined with the complete representation model, we have synthetically constructed a suitable series of data sets. Specifically, we have generated $n \times n$ relations for various values of n, with random degrees of relation between any pair of elements, and performed a transitive closure operation on them. The sizes n of relations constructed are $n \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 500, 1000, 2000\}$; the names of the corresponding relations are R_n^d.

6.2 Re-establishment of transitivity using the complete representation model

As shown in section 4, ITU algorithm has a complexity of $O(n^2)$, compared to a complexity of $O(n^3)$ of the conventional approach, assuming the complete representation model. In order to experimentally verify this, we have applied both to the R_n^d data set. In the cases where size n was too small to lead to a reliable measurement of time, the operations were executed 10, 100, 1,000 or 1,000,000 times, and the total time was divided by the count of repetitions. Table 6 summarizes the experimental times.

It is easy to see that algorithm ITU requires considerably less time in order to perform the same operation when compared to the conventional method-

Table 6
Execution times for 2 compositions and for ITU on R_n^d using complete representation

Size n	conventional algorithm (2 compositions)	ITU algorithm
10	0.000231s	0.00002s
20	0.001272s	0.00005s
30	0.004196s	0.00013s
40	0.009724s	0.000231s
50	0.018867s	0.00036s
60	0.032887s	0.000531s
70	0.052175s	0.000711s
80	0.077862s	0.000931s
90	0.11046s	0.001201s
100	0.15402s	0.001493s
200	1.20353s	0.00588s
300	4.496s	0.0136s
500	21.29s	0.039297s
1000	164.616s	0.1512s
2000	1353.796s	0.6039s

ology, for all sizes of relations. For example, when the size of the relation is 2000×2000 the conventional methodology takes more that 2,200 times as long as the proposed ITU algorithm in order to re-establish transitivity. More importantly, as can be seen more intuitively in figure 16, where the elevation of the graph for ITU is smaller that that for the conventional approach when plotted on a logarithmic scale, algorithm ITU is much more efficient in scaling with respect to size, as was also expected from the theoretically computed enhanced complexity; In figure 16.A the time axis is linear, while in figure 16.B it is logarithmic.

6.3 Re-establishment of transitivity for sparse relations using the sparse representation model

In section 4 we have shown that ITU algorithm has a sub-linear complexity, compared to a complexity of $O(n^2 \log n)$ of the conventional approach,

Table 7
Execution times for 2 compositions and for ITU on R_n using sparse representation.

Size n	conventional algorithm (2 compositions)	ITU algorithm (1,000,000 repetitions)
500	201.564s	7.64s
1000	1543.842s	8.95s
2000	13436s	9.74s
3000		10.29s
5000		10.45s
10000		10.95s
20000		12.40s
50000		13.53s
90000		17.80s

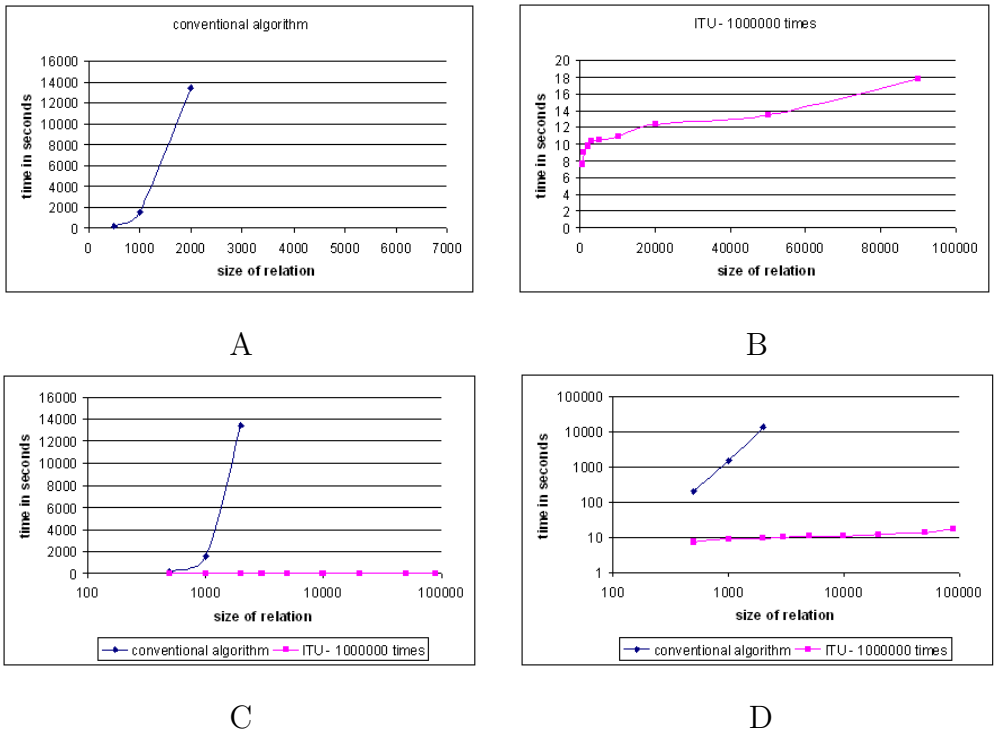
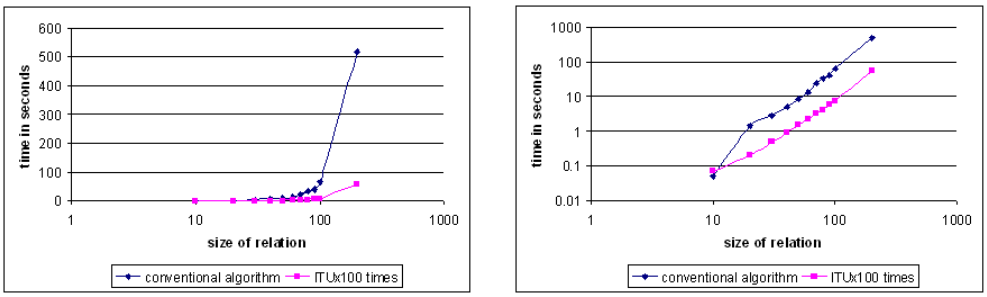


Fig. 17. Execution times for 2 compositions and for ITU on R_n using sparse representation.

assuming sparse relations and the proposed sparse representation model. In order to experimentally verify this, we have applied both to the R_n data set. Table 7 summarizes the results. In some cases execution time was too long and the experiment could not completed because of that. Therefore some values are missing from the table. For a more intuitive presentation, the values



A B

Fig. 18. Execution times for 2 compositions and for ITU on R_n^d using sparse representation.

are graphed in figure 17. Note that both in the table and in the graphs, the times reported for algorithm ITU correspond to 1,000,000 repetitions, while the times reported for the conventional approach correspond to a single application.

In figure 17.A we can observe that the calculated complexity of $O(n^2 \log n)$ is experimentally verified. Similarly, figure 17.B verifies the sub-linear complexity for algorithm ITU. Figures 17.C and 17.D contain measurements from both experiments to allow for a comparative study. We can easily see the superiority of the algorithm ITU both in execution time and in scaling with respect to the size of the relation.

6.4 Re-establishment of transitivity for dense relations using the sparse representation model

In section 4 we have shown that ITU algorithm has a complexity of $n^2 \log n$, compared to a complexity of $O(n^3)$ of the conventional approach, assuming dense relations and the proposed sparse representation model. In order to experimentally verify this, we have applied both to the R_n^d data set, which was loaded on the AVL trees of the proposed sparse representation model. Table 8 summarizes the results, and figure 18 presents the data in a more intuitive format. Note that both in the table and in the graphs, the times reported for algorithm ITU correspond to 100 repetitions, while the times reported for the conventional approach correspond to a single application.

Once more the superiority of the ITU algorithm in scaling that was proven theoretically is verified by the experimental results.

Table 8
Execution times for 2 compositions and for ITU on R_n^d using sparse representation.

Size n	conventional algorithm (2 compositions)	ITU algorithm (100 repetitions)
10	0.05s	0.07s
20	1.462s	0.211s
30	2.854s	0.481s
40	5.067s	0.911s
50	8.292s	1.522s
60	12.948s	2.283s
70	24.425s	3.205s
80	33.998s	4.286s
90	40.438s	5.618s
100	64.663s	7.181s
200	516.473s	55.83s

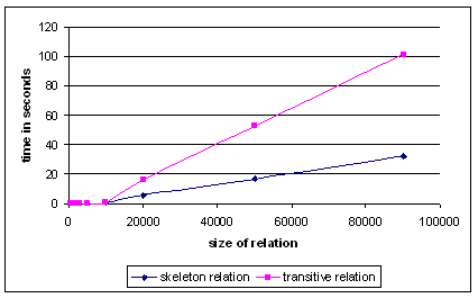


Fig. 19. Execution times for application of ITC on R_n and R_n^t.

6.5 Complete transitive closure for sparse relations using the sparse representation model

In section 5 we have proven that algorithm ITC has a complexity of $n \log^4 n$, compared to a complexity of $n^2 \log^2 n$ of the conventional approach, when considering sparse relations and the proposed sparse representation model. In order to verify the efficiency of algorithm ITC we have applied it to data sets R_n and R_n^t. Results are presented in table 9 and summarized in figure 19.

We can see that the algorithm scales almost linearly for both data sets. Moreover, the augmented density of the transitive relation has an effect on the overall execution time. Still, as can be seen in the figure, this is an effect only

Table 9
Execution times for application of ITC on R_n and R_n^t data sets.

size n	R_n data set	R_n^t data set
500	0.03s	0.03s
1000	0.05s	0.04s
2000	0.07s	0.06s
3000	0.101s	0.1s
5000	0.17s	0.16s
10000	0.591s	0.761s
20000	5.518s	15.832s
50000	16.473s	52.745s
90000	32.106s	101.396s

on the elevation of the graph, not on its form; in other words, the complexity remains close to linear and certainly below the quadratic when dealing with sparse relations that are already transitive, and thus contain more non zero elements.

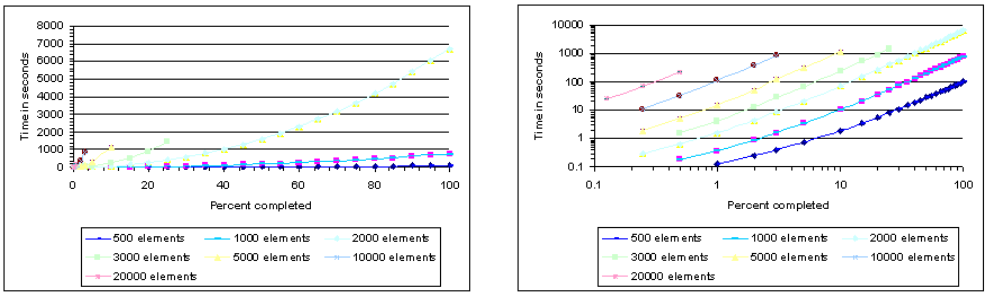
Still, what is most important is to compare the efficiency of the ITC algorithm with that of the conventional transitive closure approach. As was made obvious from the missing elements in table 7 and the form of figure 17, the execution time even of a single composition is prohibitive for the application of the conventional approach on the R_n or R_n^t data sets. Therefore, we have performed estimations of the time it would take to apply the algorithm on data set R_n, as follows:

We have edited the self-composition module, as to indicate the progress made and the time elapsed after each row of the output relation has been computed. The output of the first steps of the execution of the self-composition algorithm on data set R_{90000} is reported in Table 10.

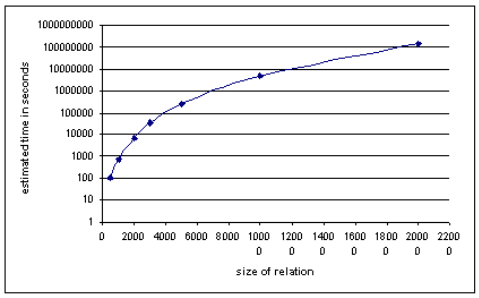
Figure 20.A graphically presents the computed times. There, we can see that the algorithm tends to slow down as the process progresses and the output relation is gradually augmented. Plotting the same points on a logarithmic scale, as in figure 20.B, we acquire much simpler curves, based on which we can estimate probable execution times for 100% of the self-composition in a more intuitive manner. This way, we have produced the estimations that are graphically presented in figure 20.C. In that figure, the first three points are the real measured execution times and the rest have been estimated; the first ones with a larger portion of the process already completed, and thus

Table 10
Execution times for self-composition algorithm on data set R_{90000}.

row	time for this step	total time
1	7.262s	7.262s
2	7.461s	14.723s
3	7.661s	22.384
4	7.871s	30.255s
5	8.202s	38.457s
6	8.182s	46.639s
7	8.301s	54.94s
8	9.134s	64.074s



A B

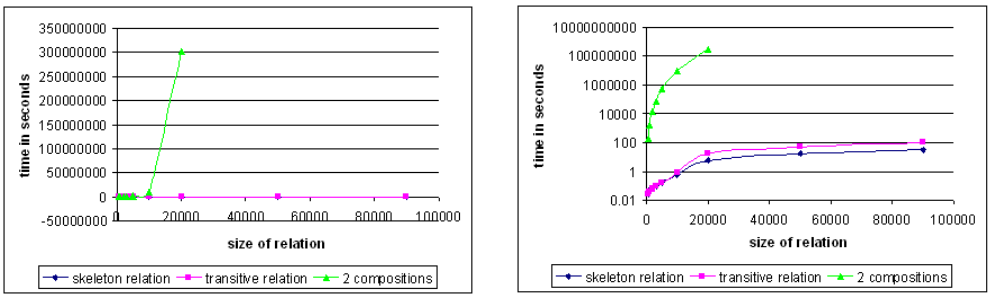


C

Fig. 20. Execution time for a single self composition on R_n

with greater confidence, the last ones with a smaller portion of the process completed and thus with lower confidence. In order to compensate for the lack of information for the execution times for larger relations, the estimations for large sizes n are more conservative. An observation that can be made in the figure is that the complexity of the self-composition algorithm is certainly far from linear.

Of course, even assuming that the estimated execution times for a single composition are correct, we still cannot estimate the overall execution time for the



A B

Fig. 21. Execution times for application of ITC on R_n and R_n^t.

transitive closure following the conventional approach, as it depends on the "depth" of the relations. For relation R_{90000}, for example, in theory the conventional approach might require anywhere between 2 and 17 self-compositions. In order to be able to make some sort of comparison we follow the best case scenario for the conventional approach, thus assuming that 2 self-compositions are adequate to guarantee transitivity. Comparative presentation of results, under these assumptions, are presented in figure 21. In the figure we can easily see, as was expected, that there is a great difference between both the execution time for the specific sizes and in the way that the algorithms scale, between the traditional approach and the proposed ITC algorithm.

7 Conclusions

In this paper we have dealt with the transitive closure of sparse fuzzy binary relations. We have started by formalizing the notion of sparseness and by proposing a compact representation model that allows for $O(\log n)$ access to a specific element, row or column in the relation, as well as $O(\log n)$ insertion time. Continuing, we have presented ITU, an algorithm for incremental update of transitive binary relations. This algorithm has important practical implications in fields were transitive relations are constructed using a trial and error approach. Extending this, we have described ITC, an algorithm for transitive closure of binary relations that relies on the above incremental methodology. The practical implications of this algorithm are found in fields were large and sparse transitive relational representations are meaningful. As most representative examples we can mention knowledge based systems, ontological representations and intelligent information and multimedia retrieval. Other fields and applications that require the transitive closure of binary relations, fuzzy or not, may benefit from the findings of this work, provided that the relations they need to handle are compliant with the given assumptions

on sparseness.

As further work, we have the intention to derive theoretical estimations of the computational complexities of algorithms ITU and ITC when assumptions on sparseness are variable, e.g. $k_r(n)$ non zero rows, $k_c(n)$ non zero columns, $r_i(n)$ non zero elements in row i, etc. Extending this work, we hope to reach a theoretical criterion determining which approach to complete transitive closure, the conventional one or ITC, is best for a given relation, assuming that some statistical knowledge on the structure of the relation is available. Finally, we intend to further investigate the practical implications of this work in diverse fields, such as the fields of context determination in ontologies, intelligent information retrieval, or even dynamic routing algorithms for computer networks.

References

[1] Adelson-Velskii G.M., Landis E.M., "An algorithm for the organization of information" Doklady Akademia Nauk SSSR, vol. 146, pp. 263-266, 1962; English translation in Soviet Math, vol. 3, pp. 1259-1263, 1962.

[2] Akrivas G. and Stamou G., "Fuzzy Semantic Association of Audiovisual Document Descriptions", International Workshop on Very Low Bitrate Video Coding (VLBV), Athens, Greece, October 2001.

[3] Akrivas G., Wallace M., Andreou G., Stamou G. and Kollias S., "Context - Sensitive Semantic Query Expansion", ICAIS, Divnomorskoe, Russia, 2002.

[4] Athanasiadis T., Avrithis Y., "Adding Semantics to Audiovisual Content: The FAETHON Project" in Enser P.,, Kompatsiaris Y., OConnor N.E., et al. (Eds) Image and Video Retrieval, LNCS 3115, pp. 665-673, 2004.

[5] Avrithis Y., Stamou G., Wallace M., Marques F., Salembier P., Giro X., Haas W., Vallant H.,Zufferey M., "Unified Access to Heterogeneous Audiovisual Archives" .Journal of Universal Computer Science vol. 9 no. 6, pp. 510-519, 2003

[6] Backhouse R.C."Calculating the Floyd-Warshall path algorithm", Eindhoven University of Technology, 1992.

[7] Backhouse R.C., van den Eijnde J.P.H.W., van Gasteren A.J.M., "Calculating path algorithms", Sci. Comput. Programming vol. 22(3), pp. 3-19, 1994.

[8] Backhouse R.C., van Gasteren A.J.M., "Calculating a Path Algorithm", Second International Conference on Mathematics of Program Construction, 1992, Lecture Notes in Computer Science vol. 669, pp. 32-44, 1993.

[9] Baker J.J., "A note on multiplying Boolean matrices", Comm. ACM vol. 5 no. 2, pp. 102, 1962.

[10] Bedek J.C., Biswas G. and Huang L.Y., "Transitive closures of fuzzy thesauri for information retrieval systems", International Journal of Man-Machine Studies vol. 25, pp. 343-356, 1986.

[11] Boixader D., Jacas J. and Recasens J.,"Transitive closure and betweenness relations", Fuzzy Sets & Systems vol. 120, pp. 415-422, 2001.

[12] Chen S.-M., Horng Y.-J. and Lee C.-H."Fuzzy information retrieval based on multi-relationship fuzzy concept networks", Fuzzy Sets & Systems vol. 140, pp. 183-205, 2003.

[13] Dasgupta M. and Deb R.,"Factoring fuzzy transitivity", Fuzzy Sets & Systems vol. 118, pp. 489-502, 2001.

[14] Dawyndt P., De Meyer H., De Baets B., "The complete linkage clustering algorithm revisited", Soft Computing, in press.

[15] De Baets B., De Meyer H., "On the existence and construction of T-transitive closures", Inform. Sc. vol. 152, pp. 167-179, 2003.

[16] De Baets B., De Meyer H., "Transitive approximation of fuzzy relations by alternating closures and openings", Soft Computing, pp. 210-219, 2003.

[17] De Baets B., De Meyer H., Naessens H., "A top-down algorithm for generating the Hasse tree of a fuzzy preorder closure", IEEE Trans. Fuzzy Systems, in press.

[18] Duan J.-S., "The transitive closure, convergence of powers and adjoint of generalized fuzzy matrices", Fuzzy Sets & Systems vol. 145, pp. 301-311, 2004.

[19] Dunn J.C., "A graph-theoretical analysis of pattern classification via Tamura's fuzzy relation", IEEE Trans. SMC vol. 5, pp. 310-313, 1974.

[20] Dunn J.C., "Some recent investigations of a new fuzzy partitioning algorithm and its applications to pattern classification problems", Journal of Cybernetics vol. 4, pp. 1-15, 1974.

[21] Feijs L.M.G., van Ommering R.C., "Abstract derivation of transitive closure algorithms", Information Processing Letters vol. 63, pp. 159-164, 1997.

[22] Fellbaum C. (Ed), "WordNet: An Electronic Lexical Database", MIT Press, 1998.

[23] Fodor, J. and Roubens M., "Structure of transitive valued binary relations", Mathematical Social Sciences vol. 30, pp. 71-94, 1995.

[24] Guu S.-M., Chen H.-H. and Pang C.-T., "Convergence of products of fuzzy matrices", Fuzzy Sets & Systems vol. 121, pp. 203-207, 2001.

[25] Klir G., Yuan B., "Fuzzy Sets and Fuzzy Logic: Theory and Applications", Prentice Hall, 1995.

[26] Kundu S.,"A representation theorem for min-transitive fuzzy relations", Fuzzy Sets & Systems vol. 109, pp. 453-457, 2000.

[27] Lee H.-S., "An optimal algorithm for computing the maxmin transitive closure of a fuzzy similarity matrix", Fuzzy Sets & Systems vol. 123, pp. 129-136, 2001.

[28] Maedche A.,Motik B., Silva N., Volz R.," MAFRA - An Ontology MApping FRAmework in the Context of the SemanticWeb". Workshop on Ontology Transformation, ECAI, 2002.

[29] De Meyer H., Naessens H., De Baets B., "Algorithms for computing the min-transitive closure and associated partition tree of a symmetric fuzzy relation", European J. Oper. Res. vol. 155, pp. 226-238, 2004.

[30] Naessens H., De Meyer H., De Baets B., "Algorithms for the computation of T-transitive closures", IEEE Trans. Fuzzy Systems vol. 10, pp. 541-551, 2002.

[31] Ovchinnikov S., "Numerical representation of transitive fuzzy relations", Fuzzy Sets & Systems vol. 126, pp. 225-232, 2002.

[32] Purdom P., "A transitive closure algorithm", BIT vol. 10 pp. 76-94, 1970.

[33] Seidel R., "On the All-Pairs-Shortest-Path problem in unweighted undirected graphs", J. Computer & System Sciences vol. 51, pp. 400-403, 1995.

[34] Stamou G., Avrithis Y., Kollias S., Marques F., Salembier P. "Semantic Unification of Heterogenous Multimedia Archives" Proc. of 4th European Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS), London, UK, April 9-11, 2003.

[35] Tan Y.-J., "On compositions of lattice matrices", Fuzzy Sets & Systems vol. 129, pp. 19-28, 2002.

[36] Thorelli L.-E., "An algorithm for computing all paths in a graph", BIT vol. 6 pp. 347-349, 1966.

[37] Ullman, J.D., Yannakakis M., "The Input/Output Complexity of Transitive Closure", Annals of Mathematics and Artificial Intelligence, vol. pp. 331-360, 1991.

[38] Wallace M., Akrivas G., Mylonas P., Avrithis Y., Kollias S. "Using context and fuzzy relations to interpret multimedia content", CBMI, IRISA, Rennes, France, 2003.

[39] Wallace M., Avrithis Y., Stamou G., Kollias S. "Knowledge-based Multimedia Content Indexing and Retrieval" Multimedia Content and Semantic Web: Methods, Standards and Tools, Stamou G., Kollias S. (Editors), Wiley, 2004.

[40] Wallace M., Akrivas G. and Stamou G., "Automatic Thematic Categorization of Documents Using a Fuzzy Taxonomy and Fuzzy Hierarchical Clustering", FUZZ-IEEE, St. Louis, MO, USA, 2003.

[41] Wallace M., Akrivas G., Stamou G. and Kollias S., "Representation of user preferences and adaptation to context in multimedia content – based retrieval", Workshop on Multimedia Semantics, SOFSEM, Milovy, Czech Republic, 2002.

[42] Wallace M., Kollias S., "Computationally efficient incremental transitive closure of sparse fuzzy binary relations", FUZZ-IEEE, July 2004.

[43] Warren H.S., "A modification of Warshall's algorithm for the transitive closure of binary relations", Comm. ACM, vol. 18, no. 4, pp. 218-220, 1975.

[44] Warshall S., "A theorem on Boolean matrices", J. ACM vol. 9 no. 1, pp. 11-12, 1962.

[45] Yoshida Y.,"A limit theorem in dynamic fuzzy systems with transitive fuzzy relations", Fuzzy Sets & Systems, vol. 109, pp. 371-378, 2000.

[46] Zadeh L.A., "Fuzzy Sets", Information and Control, vol. 3, pp. 338-353, 1965.

[47] Zadeh L.A., "Similarity relations and fuzzy orderings", Information Sciences, vol. 3, pp. 177-200, 1971.

[48] `http://image.ntua.gr/~wallace/java/transitive`