Computationally Efficient Incremental Transitive Closure of Sparse Fuzzy Binary Relations

Manolis Wallace
School of Electrical and
Computer Engineering
National Technical University of Athens
Zographou, Greece
E-mail: wallace@image.ntua.gr

Stefanos Kollias
School of Electrical and
Computer Engineering
National Technical University of Athens
Zographou, Greece
E-mail: stefanos@cs.ntua.gr

***Abstract*— Existing literature in the field of transitive relations focuses mainly on dense, Boolean, undirected relations. With the emergence of a new area of intelligent retrieval, where sparse transitive fuzzy ordering relations are utilized, existing theory and methodologies need to be extended, as to cover the new needs. This paper discusses the incremental update of such fuzzy binary relations, while focusing on both storage and computational complexity issues. Moreover, it proposes a novel transitive closure algorithm that has a remarkably low computational complexity (below $O(n^2)$) for the average sparse relation; such are the relations encountered in intelligent retrieval.**

I. Introduction

Transitivity of binary relations is a property that has been closely related to the study of graphs. In that framework, transitive closure of a relation is equivalent to detecting the pairs of vertices that are either directly connected or connected via some path. The majority of existing literature on transitive closure is quite old and has focused mainly on the cases of unweighted [1][2][3][4][5] and usually undirected graphs [6], while paying little attention to sparse matrix representation issues. Works that focus on the I/O complexity of the algorithms also refer to the Boolean case [7]. As less attention has been given to the closure of sparse, directed and weighted graphs, there is a theoretical gap when it comes to the computationally efficient transitive closure of such relations.

Quite recently, in parallel with the developments of ontological representation and storage of knowledge [8], the EU funded FAETHON project has investigated the utilization of fuzzy binary relations towards more intelligent and semantics based information and multimedia retrieval [9]. In this work, sparse partial ordering fuzzy binary relations have been found to be the most efficient in describing an application domain. More information on the properties of these relations and the algorithms that utilize them can be acquired from [10][11][12][13][14].

Within the framework of the FAETHON project, in the transition from basic research results to actual prototype implementation, the handling of the relations was proven to be the most challenging task. Specifically, the memory required for the storage of an $n \times n$ is prohibitive, as n is the count of concepts / semantic entities that are known to the system and is close to $n \simeq 70000$.

We have managed to overcome this problem with the utilization of a more compact representation format, since utilized relations were found to be very sparse; this has had a small expense of computational time required to access a specific element. Still, the two following issues, related to the theory and practice of transitive closure of fuzzy ordering binary relations have to be tackled:

- Most developed retrieval and document analysis algorithms assume that the fuzzy binary relation they accept as input is transitive in some form. Thus, before the system is made operable for the first time, a transitive closure of the $n \times n$ relation has to be computed.
- The relation is constructed using a trial and error strategy, which means that small adjustments are often made, thus locally disturbing the transitivity of the relation. A new time consuming transitive closure after each adjustment would make the application of the trial and error strategy impossible, thus calling for a method for incremental adjustment of the property of transitivity.

Both of these issues are tackled using the methodologies presented in this paper. Specifically, in section II we start by discussing the compact representation model chosen for the relations, as the properties of this data model directly affect the properties of the algorithms to follow. In the same section we also discuss the properties of the classical approach to transitive closure. Continuing, in section III we present our approach for incremental update of a binary relation and discuss its storage and computational merits. In the same section we extend our discussion to explain how this can be utilized to simplify the complete transitive closure of a relation as well. Section IV lists a few indicative experimental results and section V lists our concluding remarks.

II. Preliminaries

A. Data Model

A fuzzy binary relation defined on a set S containing n distinct elements is equivalent to a square matrix of dimension n. The representation of such an array requires the representation of n^2 different values, which for large numbers of n is prohibitive for practical implementation. On the other

hand, in such a representation access time has a computational complexity of $O(1)$, as the position of the element directly specifies its position in memory as well, with the utilization of a formula of the form

$$M = (i-1) * n + j \tag{1}$$

where (i, j) is the position of the element in the matrix and M is its actual position in memory.

As explained in section I, the utilization of such a representation is not always possible. In the cases that 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. A classical sparse array implementation utilizes linked lists to represent elements, thus raising the computational complexity of accessing a specific element to $O(n)$. In this case, although the storage requirements are much smaller, the representation model remains inapplicable for algorithms such as the ones in the FAETHON project, where numerous operations utilizing a binary relation have to be performed for a query to be processed processed, 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 sorted vectors. The first vector is sorted according to index i, and in case of identical row positions, column position j is utilized, and vice versa for the second vector.

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

For example, the above array would be represented using the following vectors:

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

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

This representation model preserves the storage merits of the classical sparse matrix implementation. Moreover, access time for a specific element, row or column has a computational complexity of $O(\log n)$, utilizing a divide and conquer approach.

B. Classical Transitive Closure

A complete transitive closure of a fuzzy binary relation is performed utilizing one of the following methodologies [15]:

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

$$Tr^t(r) = \bigcup_{f=1}^{\infty} r^f \tag{2}$$

$$r^{f+1} = r^f \circ^t r \tag{3}$$

$$r^1 = r \tag{4}$$

In (2), as well as in all subsequent equations in this paper, the classical fuzzy co-norm $\max$ is assumed. Given that the set is finite ($|S| = n$), (2) can be rewritten as :

$$Tr^t(r) = \bigcup_{f=1}^{n-1} r^f \tag{5}$$

Moreover, in order to avoid some steps and lower the computational complexity, (3) can be rewritten as [15]:

$$r^{2f} = r^f \circ^t r^f \tag{6}$$

Finally, in order to avoid performing the costly operation of relation composition more times than needed, the process may stop before reaching $f \geq n-1$, if it is found that $r^{2f} = r^f$, as it is then easily proven that $Tr^t(r) = r^f$.

As a special case, when relation r is reflective, it is proven that the transitive closure is given by

$$Tr^t(r) = r^{n-1} \tag{7}$$

In the case of complete representation, the calculation of the composition of two relations has a complexity of $O(n^3)$[1].

Proof: An element r^2_{ij} of the composition is calculated as

$$r^2_{ij} = \bigcup_{c=1}^{n} r_{ic} \cap_t r_{cj} \tag{8}$$

This is an $O(n)$ operation. As the output of the composition has a dimension of $n \times n$, n^2 such operations need to be performed, thus concluding in an overall complexity of $O(n^3)$. ■

Thus, the transitive closure has a computational complexity of $O(n^3 \log n)$, for both reflective and non reflective binary relations.

Proof: In the reflective case, (7) can be used to calculate the transitive closure of the relation. Utilizing (6) we can calculate r^{n-1} in $O(\log n)$ compositions. Each one of them, as proven above, has a complexity of $O(n^3)$, thus summing up to $O(n^3 \log n)$

In the non reflective case, combining (5) and (6) we can see that we need $O(\log n)$ additions and compositions. An addition has a complexity of $O(n^2)$ and a composition, as shown above $O(n^3)$. Thus, the overall complexity is again

$$O(\log n) \times (O(n^3) + O(n^2)) = O(n^3 \log n) \tag{9}$$

■

In the case of the proposed compact representation, with the utilization of the sorted vectors, the computational complexity is better in the best case and the average case and equal to the one of the complete representation in the worst case.

Proof: In the compact representation case, if row i and column j exist in the relation, they are retrieved. As already explained, retrieving a row or column has a complexity of

[1]Relation composition can be considered as a generalized matrix multiplication operation. Having that in mind, enhanced matrix multiplication algorithms may be utilized, resulting in a computational complexity of $O(n^{2.376})$

$O(\log n)$. Continuing, the corresponding element r^2_{ij} is calculated as

$$r^2_{ij} = \bigcup_{r_{ic} \in row \vee r_{cj} \in col} r_{ic} \cap_t r_{cj} \tag{10}$$

As the row and column are both available in a sorted by index form, this is an $O(\max(k_i^{row}, k_j^{col}))$ operation, where k_i^{row} is the count of elements in row i and k_j^{col} is the count of elements in column j. $a \times b$ such operations will be performed, where a is the count of non zero rows and b is the count of non zero columns of the relation. Overall, the complexity for a single composition is

$$2 \cdot O(\log n) + O(ab) \cdot O(\max(k_i^{row}, k_j^{col}))$$

In the best case the relation is empty. Thus $a = b = 0$. Moreover, in the compact representation, n is the count of elements that actually participate in the relation, and thus $n = 0$ also holds. Thus, the closure operation terminates in a single operation. In the average case for sparse relations a small percentage of the rows and columns will be non zero. Of course, although this affects greatly the execution time required, it does not alter the complexity, as

$$O(ab) = O((p_a n)(p_b n)) = O(n^2) \tag{11}$$

where p_a and p_b 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, this is proportional to the logarithm of the count of all the elements in the relation. Thus $O(\max(k_i^{row}, k_j^{col}))$ is replaced by $O(\log n)$. Overall, this leads to a complexity of

$$2 \cdot O(\log n) + O(n^2) \cdot O(\log n) = O(n^2 \log n) \tag{12}$$

for the composition. In the worst case, all the elements of the relation exist. In that case, $a = b = k_i^{row} = k_j^{col} = n$, and thus the complexity of the composition becomes

$$2 \cdot O(\log n) + O(n^2) \cdot O(n) = O(n^3) \tag{13}$$

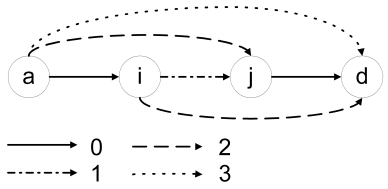
Considering the $O(\log n)$ compositions are required for the transitive closure, it is easy to calculate that in the best case we need just $O(1)$ operations, in the average case $O(n^2 \log^2 n)$ operations and in the worst case $O(n^3 \log n)$. Proving that the complexity in the average case is better using a compact representation, i.e. proving that $O(n^2 \log^2 n) < O(n^3 \log n)$, is trivial and can be achieved, for example, by showing that

$$\lim_{x \to +\infty} \frac{x^3 \log x}{x^2 \log^2 x} = +\infty \tag{14}$$

■

As far as the storage requirements are concerned, the existence of two copies of the relation during the execution of the algorithm is required, for both compact and complete representations; the effect of this is of course much more intense in the complete representation case, as much more space is required.

Fig. 1. Graphical representation of the incremental update of the transitive relation.



III. Incremental Closure of Fuzzy Binary Relations

A. Incremental Update

A major disadvantage of the utilization of the classical transitive closure methodologies is that when for some reason an element of the relation is updated, or when a new element is inserted to the global set S, thus locally disturbing transitivity, an $O(n^3)$ operation has to be performed in order to assure that the relation is still transitive.

Proof: Let r be a transitive relation. In Fig. 1 r is represented using lines of type 0. Let us suppose that r_{ij} is inserted in the relation, or that its value is augmented. In Fig. 1 the update is represented using 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. 1 this is represented using lines of type 2. Finally, after one more self-composition, a is linked to b. In Fig. 1 this is represented using lines of type 3. We always need exactly two operations of complexity $O(n^3)$, so the overall operation has a complexity of $O(n^3)$. ■

Having observed in Fig. 1 the way 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, we need to design an algorithm that will only focus on the changes that the complete transitive closure brings upon the relation after the update of the single element.

For any relation r this can be achieved with the following steps:

1) Identify the fuzzy set A of ancestors of i. Degrees in A are determined as

$$A(s) = r(s, i), s \in S \tag{15}$$

2) Identify the fuzzy set D of descendants of j. Degrees in A are determined as

$$A(s) = r(j, s), s \in S \tag{16}$$

3) For each element s appearing in A assign

$$r(s, j) = r(s, j) \cup (r(s, i) \cap_t r(i, j)) \tag{17}$$

4) For each element s appearing in D assign

$$r(i, s) = r(i, s) \cup (r(i, j) \cap_t r(j, s)) \tag{18}$$

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

$$r(s_1, s_2) = r(s_1, s_2) \cup (r(s_1, j) \cap_t r(i, j) \cap_t r(j, s_2)) \tag{19}$$

If relation r is reflective, then $A(i) = 1$ and $D(j) = 1$. Thus, the above process can be simplified by omitting steps 3 and 4.

The computational complexity of the proposed algorithm is smaller than that of performing a complete transitive closure. Specifically, the complexity is $O(n^2)$, while in the average case a complexity of $O(\log^2 n)$ is expected and in the best case the algorithm terminates in $O(\log n)$ operations, for both reflective and other relations.

Proof: As already explained, the complexity of steps 1 and 2 is $O(\log n)$. The complexity of steps 3 and 4 is $O(k_i^{row})$ and $O(k_j^{col})$, respectively, and the complexity of step 5 is $O(k_i^{row} \cdot k_j^{col})$.

In the average case $k_j^{col} = O(\log n)$, and thus the overall complexity is

$$4 \cdot O(\log n) + O(\log^2 n) = O(\log^2 n) \tag{20}$$

In the worst case $k_j^{col} = n$, and thus the overall complexity is

$$2 \cdot O(\log n) + 2 \cdot O(n) + O(n^2) = O(n^2) \tag{21}$$

In the base case, $k_i^{row} = k_j^{col} = 1$, (or $k_i^{row} = k_j^{col} = 0$, depending on whether the relation is reflective or not). Either way, the overall complexity is

$$2 \cdot O(\log n) + 3 \cdot O(1) = O(\log n) \tag{22}$$

Ignoring the influence of steps 3 and 4 in the above calculations does not alter the overall complexity, as in all cases some other step contributes more to it. Thus, the same complexity holds for the reflective case as well. ■

B. Application to Complete Closure

The above algorithm for incremental update of transitive fuzzy binary relations easily leads to the design of an algorithm for a complete transitive closure of a relation as well. The steps of the algorithm follow:

1) Create an empty binary relation r'.
2) For each element r_{ij} in the initial relation r
   a) Assign $r'(i, j) = r'(i, j) \cup r(i, j)$
   b) Run the incremental update algorithm on relation r' with parameters i and j.

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

The computational complexity of this algorithm is $O(n \log^3 n)$ in the average case and $O(n^4)$ in the worst case.

Proof: Step 1 is completed in an $O(1)$ operation. Step 2a is executed in an $O(1)$ 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 worst case, the count of elements in r is n^2, and step 2b has a complexity of $O(n^2)$, thus resulting in

$$2 \cdot O(1) + O(n^2) \cdot (O(1) + O(n^2)) = O(n^4) \tag{23}$$

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 $O(\log^2 n)$. Thus, the overall complexity is

$$2 \cdot O(1) + O(n \log n) \cdot (O(1) + O(\log^2 n)) = O(n \log^3 n) \tag{24}$$

■

C. Comparisons

Comparing the traditional and the incremental approaches to assuring transitivity, in the case of a relation that is already transitive and is undergoing a small change is easy. In all cases the incremental approach presented herein is superior; it has a complexity of $O(\log^2 n)$ in the average case and $O(n^2)$ in the worst case, compared to a complexity of $O(n^3)$ in all cases for the traditional approach. Moreover, the proposed approach is more efficient in means of storage requirements as well, as no copy of the relation needs to be created in memory.

When it comes to a complete transitive closure, the comparison is a little more complicated. It is clear that in the dense/worst case, the traditional approach is better, having a complexity of $O(n^3 \log n)$, compared to a complexity of $O(n^4)$ for the proposed approach. In the average case, on the other hand, the proposed approach is greatly superior, having a complexity of $O(n \log^3 n)$; proving that

$$O(n^3 \log n) > O(n^2) > O(n \log^3 n)$$

is again trivial and can be achieved by proving that

$$\lim_{x \to +\infty} \frac{x^2}{x \log^3 x} = +\infty \tag{25}$$

Thus, comparison of the two approaches depends on the validity of the assumptions made concerning the average case. The main assumption made is that considerably less non zero elements exist in the average case; $O(n \log n)$ instead of $O(n^2)$. This assumption is valid, not just because it is verified through the practical experience of the FAETHON project, but also because it is not feasible for it to be invalid: If the relation had proportionally less elements, or, in other words, if $O(n^2)$ non zero elements existed in relation r, then it would not be possible to store the relation in main memory in real life applications, where n is very large.

On top of the enhanced computational complexity, the proposed approach to complete transitive closure of fuzzy binary relations also has the following merits:

- In the traditional approach, a composition of the relation, so an operation of complexity $O(n^2 \log n)$ in the average case, has to be performed before the decision to stop 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 $O(n \log^3 n)$.

- In the traditional 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 proposed approach, and thus algorithms that assume it to be transitive can be applied to it before the whole operation is completed.
- Due to the recursive form of the traditional approach, the termination point greatly depends on the "depth" of the relation, i.e. on the count of vertices of the longest path. On the contrary, the proposed approach assures transitivity in one – pass and the count of required steps is known beforehand. Thus, the progress of the process can be monitored on-line.
- The traditional 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 proposed approach; two relations exist, but as elements are added in one, they are removed from the other.
- The proposed approach is not affected by cycles in the relation, i.e. it does not require the relation to be ordering. This is is not a property shared by all transitive closure algorithms [7].

IV. Experimental Results

The proposed methodology for transitive closure, as well as the traditional approach, have been implemented using the Java programming language in the framework of the FAETHON project. Moreover, a fuzzy binary relation representing all concepts and semantic entities in the English language has been prepared, utilizing WordNet as a source. This relation has a dimension of $n \simeq 70000$ and contains approximately 90000 non zero elements.

The complete (not compact) representation of the relation was not been possible, as it would require storing approximately 5 billion double precision numbers; with 8 bytes allocated for each double precision number in Java, this means that 40Gb of RAM would be required just for the storage of one copy of the relation.

Utilizing the proposed compact representation form, a transitive closure of the relation has been attempted, using both the proposed and the traditional approach; the t-norm used for the transitive closure was the $\min$. The execution time for the proposed approach was 6.45 seconds. It took the traditional approach a little over 5 days to complete the same operation. Execution of both algorithms was performed in identical environments; the same computer was utilized in both cases (PIII 900 running W2K as operating system), and the computer was in both cases dedicated to the execution of the transitive closure algorithm.

V. Conclusions

In this paper, after proposing a compact representation format for sparse binary relations that allows for logarithmic access a row, a column or an element, we presented a method for incremental update of transitive binary relations; we proved that this method outperforms the traditional approach of performing a complete transitive closure in both the worst and the average case.

Continuing, we have described a method for transitive closure of any binary relation that relies on the above incremental methodology. We have proven that this method greatly outperforms the classical approach for the average sparse relation, achieving a computational complexity that is below $O(n^2)$. Finally, we have implemented the traditional and proposed methodologies and we have tested then on a real life fuzzy binary relation. The difference in execution time was immense, proving the efficiency of the proposed algorithm.

Acknowledgment

This work has been partially funded by the EU IST-1999-20502 FAETHON project.

References

[1] Warshall, S., "A theorem on Boolean matrices", J. ACM vol. 9 no. 1, pp. 11-12, 1962.
[2] 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.
[3] Purdom, P., "A transitive closure algorithm", BIT vol. 10 pp. 76-94, 1970.
[4] Thorelli, L.-E., "An algorithm for computing all paths in a graph", BIT vol. 6 pp. 347-349, 1966.
[5] Baker, J.J., "A note on multiplying Boolean matrices", Comm. ACM vol. 5 no. 2, pp. 102, 1962.
[6] Seidel, R., "On the All-Pairs-Shortest-Path problem in unweighted undirected graphs", J. Computer & System Sciences vol. 51, pp. 400-403, 1995.
[7] Ullman, J.D., Yannakakis, M., "The Input/Output Complexity of Transitive Closure", Annals of Mathematics and Artificial Intelligence, vol. pp. 331-360, 1991.
[8] 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.
[9] Avrithis Y. and Stamou G.: "FAETHON: Unified Intelligent Access to Heterogeneous Audiovisual Content", VLBV, Athens, Greece, 2001.
[10] Akrivas, G., Wallace, M., Andreou, G., Stamou, G. and Kollias, S., "Context - Sensitive Semantic Query Expansion", ICAIS, Divnomorskoe, Russia, 2002.
[11] 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.
[12] Wallace, M., Akrivas, G., Mylonas, P., Avrithis, Y., Kollias, S. "Using context and fuzzy relations to interpret multimedia content", CBMI, IRISA, Rennes, France, 2003.
[13] 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.
[14] Avrithis, Y., Stamou, G, Wallace, M., Marques, F., Salembier, P., Giro, X., Haas, W., Vallant, H. and Zufferey, M., "Unified Access to Heterogeneous Audiovisual Archives", I-KNOW, Graz, Austria, 2003.
[15] Klir, G., Yuan, B., "Fuzzy Sets and Fuzzy Logic: Theory and Applications", Prentice Hall, 1995.