FAST PARALLEL ALGORITHMS THAT COMPUTE TRANSITIVE CLOSURE OF A FUZZY RELATION

Vladik Kreinovich
Computer Science Department, University of Texas at El Paso
El Paso, TX 79968, email vladik@cs.ep.utexas.edu

Abstract. The notion of as transitive closure of a fuzzy relation is very useful for clustering in pattern recognition, for fuzzy databases, etc. The original algorithm proposed by L. Zadeh (1971) requires the computation time $O(n^4)$, where n is the number of elements in the relation. In 1974, J. C. Dunn proposed a $O(n^2)$ algorithm. Since we must compute $n(n-1)/2$ different values $s(a,b)$ $(a \neq b)$ that represent the fuzzy relation, and we need at least one computational step to compute each of these values, we cannot compute all of them in less than $O(n^2)$ steps. So, Dunn's algorithm is in this sense optimal. For small n, it is OK. However, for big n (e.g., for big databases), it is still a lot, so it would be desirable to decrease the computation time (this problem was formulated by J. Bezdek). Since this decrease cannot be done on a sequential computer, the only way to do it is to use a computer with several processors working in parallel.

We show that on a parallel computer, transitive closure can be computed in time $O((\log_2 n)^2)$.

1. FORMULATION OF THE PROBLEM

Crisp similarity: the notion of equivalence. Suppose that have several objects, and we need to group them into clusters, so that similar objects fall into the same group. In some cases, this is a perfectly well defined (crisp) task, and for every two objects we know exactly whether they belong to one class or not. In this case, the relation $a \sim b$ (meaning that a and b belong to one and the same class) satisfies the following natural properties: $a \sim a$; $a \sim b \leftrightarrow b \sim a$, and $(a \sim b \ \& \ b \sim c) \rightarrow a \sim c$. The relation with such properties is called an *equivalence relation*. It is well known that such a relation divides the set of objects into non-intersecting classes such that $a \sim b$ if and only if a and b belong to one and the same class.

It is not necessary to compare all objects with all the others to get the similarity relation: it is sufficient to have the results of comparing some pairs (and for big n, it is often simply impossible to ask the user to compare all pairs). If by K we denote the relation that represents our knowledge (i.e., aKb if we know that a is similar to b), then we must find the equivalence relation $\sim$ with the property that $aRb \rightarrow a \sim b$. The only natural restriction on K is that aKa for all a (this we know for sure), and $aKb \leftrightarrow bKa$. There may be several equivalence relations $\sim$ with this property; one of them is in which $a \sim b$ for all a and b. We would like to conclude that $a \sim b$ only if we are forced to conclude that by the knowledge that we have. So, we would like to choose as $\sim$ the "smallest" of all possible relations with these properties (a relation is defined in mathematics as a set of pairs; the smallest relation means the relation that is contained in all other relations). Such smallest relation always exists, and is called a *transitive closure* K^* of the initial relation K.

General case: fuzzy similarity. In many real-life situations, for some pairs of objects, we are unsure of whether they are similar or not. If we describe our degree of belief that a and b are similar, by the number $s(a,b) \in [0,1]$, then we get a *fuzzy relation*. The formulas that define an equivalence relation can be used to describe a fuzzy *similarity* relation. To do that, we must explain what $\rightarrow$ and & mean in fuzzy case. The statement $A \rightarrow B$ is natural to interpret as saying that our degree of belief in B is greater than or equal to our degree of belief in A; and we will denote by $f_\&$ the function from $[0,1] \times [0,1]$ to [0,1] that corresponds to & (it is supposed to be a $t-$*norm*, i.e., a symmetric associative operation, with the additional properties $f_\&(a,0) = 0$ and $f_\&(a,1) = a$; the most widely used $t-$norms are min and product).

Thus, we arrive at the following definition that was initially proposed by L. Zadeh himself [Z71] (in this and the following definitions, we will assume that some $t-$norm $f_\&$ is fixed).

Definition 1. A *fuzzy relation* on a set X is a function $s : [0,1] \times [0,1] \rightarrow [0,1]$. It is called a *similarity relation* if for all $a,b,c \in X$, $s(a,a) = 1$, $s(a,b) = s(b,a)$, and $s(a,c) \geq f_\&(s(a,b), s(b,c))$.

Definition of a fuzzy transitive closure. Just like in a crisp case, we may not have all the information about the similarity of the objects. This partial knowledge can be represented by assigning a number $k(a,b) \in [0,1]$ to describe to what extent wee believe that a is similar to be. Evidently, a is similar to a, and if a is similar to b, then b is similar to a, so $k(a,a) = 1$, and $k(a,b) = k(b,a)$. From this knowledge, we must find the transitive similarity relation $s(a,b)$. The natural conditions on s are as follows:

- if we know that a and b are similar, then they are similar (i.e., $s(a,b) \geq k(a,b)$);
- we state that $s(a,b)$ only when we are forced to do it, i.e., s must be the smallest (= weakest) similarity relation that follows from our knowledge.

These ideas, when formalized, lead to the following definitions (proposed by Zadeh):

Definition 2. A fuzzy relation k is called *symmetric* if $k(a,b) = k(b,a)$ for all a and b, and *reflexive* if $k(a,a) = 1$ for all a. We say that a relation k is *weaker* than the relation s if $k(a,b) \leq s(a,b)$ for all a,b. In this case, we also say that s *follows from* k. If we are given a family of fuzzy relations, then the weakest among them (if it exists) will also be called the *smallest*.

Definition 3. For a given symmetric reflexive fuzzy relation $k(a,b) = k(b,a)$, by its *transitive closure* we mean the smallest of all similarity relations that follow from k. We will denote the transitive closure of the relation k by k^*.

Zadeh proved that every symmetric reflexive relation has a transitive closure.

Why is it important to compute fuzzy transitive closure? Zadeh showed [Z71] that this transitive closure allows us to form a reasonable clustering of the objects: namely, if we use $f_\& = \min$, then the relation $k^*(a,b) \geq \alpha$ is an equivalence relation for all α, and so the equivalence classes that correspond to different α, form a hierarchic clustering (see also [BH78] and [BS92]).

TR1-3

Another application of fuzzy transitive closure is in fuzzy databases, when we are given only partial (and fuzzy) information about who is similar to whom, and we want to be able to answer queries about the similarity of other pairs as well. In this case, we want to be able, given some relation $k(a,b)$, to compute $k^*(a,b)$ for all a and b (this idea was proposed in [BBH86]).

How to compute fuzzy transitive closure? The first algorithm to compute k^* from k (for $f_{\&} = \min$) was proposed by L. Zadeh in [Z71]. It requires $O(n^4)$ computational steps, where $n = |X|$ is the total number of objects in X. Faster algorithms were proposed in [THT71] ($O(n^3 \log_2 n)$), [KY74] ($O(n^3)$), [D74], [L90] ($O(n^2)$ for min), and [LY90] ($O(n^2 \log_2 n)$ for an arbitrary $f_{\&}$).

Formulation of the main problem: for applications, we need a faster algorithm. In we are classifying a few objects (like 10), then $n^2 \approx 100$ is a reasonable computation time. However, for huge databases, where n can be big, an algorithm that requires the computation time $\sim n^2$ may be too slow. Therefore, it is desirable to find an algorithm that computes the transitive closure faster. This problem was formulated by J. Bezdek during an annual 1992 meeting of the North American Fuzzy Information Processing Society (NAFIPS).

On a sequential computer, Dunn's algorithm is the fastest possible. Since our goal is to compute $k^*(a,b)$ for $n(n-1)/2$ different pairs $a \neq b$, and computing each value requires at least one computational step, there is no way to compute $k^*(a,b)$ on a sequential computer faster than in $n(n-1)/2 = O(n^2)$ computational steps. So, Dunn's algorithm is asymptotically the fastest (in the sense that no algorithm with better time asymptotic is possible).

So, we need a parallel computer. Since we cannot decrease the computation time on a sequential computer, the only way to do it is to use several processors working in parallel, i.e., to use a parallel computer. In the current paper, we are presenting algorithms that compute the fuzzy transitive closure on a parallel computer.

2. MAIN RESULTS

In this paper, we will consider the two most frequently used models of parallel computations:

- The first one is called Concurrent Read Exclusive Write (CREW) Parallel Random Access Machine (PRAM). In this model, all the processors have access to the same memory. Several processors can read from the same memory location concurrently, but only one processor at a time can write to a given memory location.
- Another possible model is common Concurrent Read Concurrent Write (common CRCW) PRAM, where several processors are allowed to send the "write" commands simultaneously to the same memory location. If all these "write" commands require to write the same value, then this value is written; else, nothing is written into the memory.

THEOREM 1. *On CREW PRAM, the transitive closure of a fuzzy symmetric relation can be computed in $O((\log_2 n)^2)$ time using $O(n^3)$ processors. On common CRCW PRAM,*

TR1-4

it can be computed in $O(\log_2 n)$ time using $O(n^4)$ processors, or in $O(\log_2 n \cdot \log_2(\log_2 n))$ time using $O(n^3)$ processors.

(All the algorithms are described in Sections 3 and 4.)

The estimates for common CRCW essentially use the fact that all the processors share the same memory. The estimates for CREW PRAM remain the same if we assume that the processors do not have the shared memory. In particular, we can consider one of the real-life parallel architectures: a *hypercube*. A hypercube is a computer consisting of 2^d processors. To each of them, a number from 0 to $2^d - 1$ is assigned. In binary codes, these numbers are just all possible $d-$digit binary numbers. Two processors are connected if and only if the binary representations of their numbers differ exactly in 1 bit.

THEOREM 2. *On a hypercube of size $\geq n^3$, the transitive closure of a fuzzy relation can be computed in $O((\log_2 n)^2)$ time.*

The number of processors can be made somewhat smaller if we take into consideration the fact that the values $k(a, b)$ are defined only approximately. For example, one can ask an expert to estimate his degree of belief that a and b are similar on a scale from 0 to 10, and then divide the resulting number by 10. Hardly anyone can make distinction between more than 10 of his different degrees of belief. So, even if we use a more sophisticated technique to obtain the values $k(a, b)$, the values that differ by less than, say, 0.1, represent more or less the same degree of belief (this argument, in application to a different problem, appeared, e.g., in [NKLT92]). In this case, it makes no big sense to waste time and processors on computing the values $k^*(a, b)$ precisely: it is sufficient to compute them with some precision ε (≈ 0.1).

Definition 4. We say that an algorithm computes $k^*(a, b)$ *with precision* ε, if it computes the values $k_\varepsilon(a, b)$ such that for all a and b, $|k^*(a, b) - k_\varepsilon(a, b)| \leq \varepsilon$.

THEOREM 3. *On CREW PRAM, the transitive closure of a fuzzy relation can be computed with precision ε in $O((\log_2 n)^2 + \log_2(1/\varepsilon))$ time, using $O(n^{2.376}/\varepsilon)$ processors. On common CRCW PRAM, it can be computed with precision ε in $O(\log_2 n + \log_2(1/\varepsilon))$ time using $O(n^3/\varepsilon)$ processors.*

For $n \gg 10$ and $\varepsilon \approx 0.1$, $n^3\varepsilon \ll n^4$, so these algorithms are actually faster than the ones that compute $k^*(a, b)$ precisely.

3. ALGORITHMS THAT COMPUTE THE TRANSITIVE CLOSURE PRECISELY

The main idea. For our algorithms, we will use the following result that was first proved and used in [THT71] (actually, it was proved for $f_\&$ = min, but one can easily see that it is true for an arbitrary $t-$norm $f_\&$; for a compact exposition of these results, see [D74]). Namely, it turns out that $k^*(a, b) = k_l(a, b)$, where $l = \lceil \log_2(n-1) \rceil$, and the the sequence of fuzzy relations $k_i(a, b), 1 \leq i \leq l$ is defined recursively as follows: $k_1(a, b) = k(a, b)$, and $k_{i+1}(a, b) = \max_c(f_\&(k_i(a, c), k_i(b, c)))$, where c runs over all elements of X. The algorithm from [THT71] consists of consequently computing $k_2, k_3, ..., k_l$. We will use the same idea in our parallel algorithms.

TR1-5

As noted in [THT71] and [D74], computing k_{i+1} from k_i is similar to computing the product of two identical matrices; the only difference is that here, we have $f_\&$ instead of the product, and max instead of the sum. The standard method of computing the product $c_{ik} = \sum_k a_{ij} b_{jk}$ of the two matrices a_{ij} and b_{jk} in parallel (see, e.g., [ACS90], [J92, Ch. 1]) is as follows: we take n^3 processors corresponding to all possible triples (i, j, k); on each processor, we compute $a_{ij} b_{jk}$, and then add the results that correspond to different k.

We will use the same idea in our case, by taking n^3 processors that correspond to all possible triples (a, b, c), and letting each processor compute the value $f_\&(k_i(a, c), k_i(b, c))$. Now, to get $k_{i+1}(a, b)$, we must compute the maximum of n values that correspond to different c.

How to compute the maximum of n numbers in parallel: a brief survey. The problem of computing the maximum of n numbers in parallel is well-analyzed, and optimal algorithms are known [J92]. Those algorithms depends on the type of parallel computer that we are using.

For CREW, the optimal algorithm that computes the maximum of n numbers requires $O(\log_2 n)$ computation time and n processors (see, e.g., [J92], Section 2.6). Its idea is very simple: we divide the list of n elements into two halves, use separate computers to compute the maxima of each half (concurrently), and then compute the maximum of the two resulting maxima in one computational step. Computing the maxima of each half can be also done by this same algorithm. So, if we start with $n = 2^k$ elements, then the time that is required to find their maximum (we will denote this time by $t(k)$) is equal to the time $t(k-1)$ that is required to find a maximum of 2^{k-1} numbers, plus 1. Hence, $t(k) = t(k-1) + 1$, and $t(1) = 0$, thence $t(k) = k - 1 \approx \log_2(n)$. It is known that for CREW, this algorithm is optimal [J92, p. 71].

For common CRCW, there exist two algorithms that compute the maximum of n numbers:

- the first one (see, e.g., [J92, Section 2.6.1]) computes the maximum in $O(1)$ computation time using $O(n^2)$ processors;
- the second one (proposed in [SV81]; see, e.g., [J92, Section 2.6.2]) computes the same maximum in $O(\log_2(\log_2 n))$ computation time using $O(n/\log_2(\log_2 n))$ processors. It is known [V85; J92, Section 4.6.3] that this algorithm cannot be improved: namely, if we use $O(n)$ processors, then computing the maximum of n elements requires at least $\log_2(\log_2 n)$ computational steps.

Now, we are ready to describe the algorithms that compute fuzzy transitive closure.

Description of the algorithms. We propose three algorithms, one for CREW, and two others for common CRCW PRAM. In all of them, we compute first k_2 from k_1, then k_3 from k_2, etc, until we finally get $k_l = k^*(a, b)$. So, each of these algorithms consists of $l = O(\log_2 n)$ iterations.

On each iteration, to compute k_{i+1} from k_i, we first use n^3 processors that correspond to all triples (a, b, c), to compute the values $f_\&(k_i(a, c), k_i(b, c))$. This computation takes

TR1-6

exactly the time that is necessary to compute $f_{\&}(p,q)$ for known p and q, so it takes $O(1)$ time.

Next, we compute the maximum of the resulting n elements: in Algorithm 1 (for CREW), we use n processors and $\log_2 n$ computation time; in Algorithm 2, we use $O(n^2)$ processors and $O(1)$ computation time; in Algorithm 3, we use $O(n)$ processors, and $O(\log_2(\log_2 n))$ computation time.

Estimates. Let us now estimate the computation time and the number of processors for these algorithms.

The computation time can be obtained by adding $O(1)$ and the time for finding the maximum (thus, we get the time of one iteration), and multiplying the resulting sum by the number of iterations $O(\log_2 n)$. So, for Algorithm 1, we need $O((\log_2 n)^2)$ time, for Algorithm 2 we need $O(\log_2 n)$, and for Algorithm 3, we need $O(\log_2 n \cdot \log_2(\log_2 n))$.

As for the number of processors, in algorithms 1 and 3, we need $O(n)$ processors for each of n^2 pairs (a,b), so $O(n^3)$ processors are sufficient for these algorithms. For algorithm 2, we need n^2 processors for every pair, so, totally, we need $O(n^4)$ processors.

The hypercube case. In this case (see Section 1.3 of [J92]), we can also compute $k_{i+1}(a,b)$ from $k_i(a,b)$ in $O(\log_2 n)$ time using n^3 processors: namely, we need $\log_2 n$ time to communicate the values $k_i(a,c)$ and $k_i(b,c)$ to the node that corresponds to the triple (a,b,c), $O(1)$ time to compute $f_{\&}(k_i(a,c),k_i(c,a))$, and then $\log_2 n$ time to compute the maximum of the resulting values. Multiplying this time by the number of iterations ($O(\log_2 n)$), we conclude that the total computation time is $\le O((\log_2 n)^2)$.

4. ALGORITHMS THAT COMPUTE THE TRANSITIVE CLOSURE APPROXIMATELY

These algorithms are based on the following fact (discovered by L. Zadeh in his pioneer paper [Z71]): for every α, the crisp equivalence relation $k^*(a,b) \ge \alpha$ is a transitive closure of the crisp relation $k(a,b) \ge \alpha$. Therefore, we take $O(1/\varepsilon)$ different values $\alpha = \varepsilon, 2\varepsilon, 3\varepsilon, \ldots$, and for each of these values compute the crisp transitive closure of the relation $k(a,b) \ge \alpha$. Computations that correspond to different values of α, can be done in parallel. As a result, for every pair (a,b), and for each of these α, we know whether the inequality $k^*(a,b) \ge \alpha$ is true or not. To find the value of $k^*(a,b)$ with precision ε, we must find $i \le 1/\varepsilon$ for which $i\varepsilon \le k^*(a,b) < (i+1)\varepsilon$, i.e., for which the inequality $k^*(a,b) \ge i\varepsilon$ if true, and the inequality $k^*(a,b) \ge (i+1)\varepsilon$ is false. This value i can be found by a binary search method, that requires $O(\log_2(1/\varepsilon))$ computational steps. Computations of the approximate values that correspond to each pair (a,b) will be done concurrently.

To estimate the time and the number of processors that are required for the resulting algorithm, we can use the known parallel algorithms for computing the transitive closure of a crisp relation. Such methods are described in Section 5.5.2 of [J92]. For CREW, the best known method requires $O((\log_2 n)^2)$ time and uses $M(n)$ processors, where $M(n) = n^{2.376}$ is the best known sequential bound for multiplying two $n \times n$ matrices [CW90], [ACS90]. For common CRCW, the best known method requires $O(\log_2 n)$ time and n^3 processors.

TR1-7

To get the estimates for our case, we must multiply the number of processors by $O(1/\varepsilon)$ (since we are computing $O(1/\varepsilon)$ crisp transitive closures concurrently), and add $\log_2(1/\varepsilon)$ to the computation time.

5. CONCLUSIONS

In many areas (pattern recognition, databases, etc), it is important to compute the transitive closure for a fuzzy relation, and to compute it fast. The best existing algorithms for the sequential machine require $O(n^2)$ computation time, which can be too slow for some applications. So, J. Bezdek formulated a problem of finding faster parallel algorithms. In this paper, we propose three parallel algorithms. Algorithm 1 is for the case when for every memory location, and for every moment of time, only one processor is allowed to write into it (so called CREW PRAM). This algorithm requires n^3 processors and takes $O((\log_2 n)^2)$ computation time. We also propose two faster algorithms for the case when several processors can simultaneously try to write into the same memory location (the write will be done only if all of them want to write the same value; this case is called common CRCW PRAM). Algorithm 2 requires $O(n^4)$ processors and takes $O(\log_2 n)$ computation time; Algorithm 3 takes $O(n^3)$ processors, and takes $O(\log_2 n \cdot \log_2(\log_2 n))$ computation time.

The number of processors can be made smaller if we look for algorithms that compute the transitive closure with a given precision ε.

Acknowledgements. This work was partially supported by NSF Grant No. CDA-9015006, NASA Grant No. 9–482, and a grant from the Institute for Manufacturing and Materials Management. The author is greatly thankful to J. Bezdek, A. Ramer, R. Yager, and M. Zemankova for valuable discussions.

REFERENCES

[ACS90] A. Aggarwal, A. K. Chandra, and M. Snir. "Communication complexity of PRAMs", *Theoretical Computer Science*, 1990, Vol. 71, No. 1, pp. 3–28.

[BBH86] J. Bezdek, G. Biswas, and L. Y. Huang. "Transitive closures of fuzzy tesauri for information retrieval systems," *International Journal of Man-Machine Studies*, 1986, Vol. 25, pp. 343–356.

[BH78] J. C. Bezdek, J. D. Harris. "Fuzzy partitions and relations; an axiomatic basis for clustering", *Fuzzy Sets and Systems*, 1978, Vol. 1, pp. 111–127; reprinted in [BP92], pp. 181–194.

[BP92] J. C. Bezdek, S. K. Pal (eds.) *Fuzzy models for pattern recognition*, IEEE Press, N.Y., 1992.

[CW90] D. Coppersmith and S. Winograd. "Matrix multiplication via arithmetic progressions", *Journal of Symbolic Computations*, 1990, Vol. 9, No. 3, pp. 251–280.

[D74] J. C. Dunn. "A graph theoretic analysis of pattern classification via Tamura's fuzzy relation", *IEEE Transactions on Systems, Man, and Cybernetics*, 1974, Vol. SMC-4, No. 3, pp. 310–313; reprinted in [BP92], pp. 175–177.

TR1-8

[J92] J. JáJá. *An introduction to parallel algorithms*, Addison-Wesley, Reading, NA, 1992.

[KY74] A. Kandel and L. Yelowitz. "Fuzzy chains", *IEEE Transactions on Systems, Man, and Cybernetics*, 1974, Vol. SMC-4, No. 5, pp. 472–475; reprinted in [BP92], pp. 178–180.

[LY90] H. Larsen and R. Yager. "Efficient computation of transitive closures", *Fuzzy Sets and Systems*, 1990, Vol. 38, pp. 81–90.

[L90] S. Y. Li. "The simplest method of ascending value to find fuzzy transitive closures", *Fuzzy Sets and Systems*, 1990, Vol. 38, pp. 91–96.

[NKLT92] H. T. Nguyen, V. Kreinovich, R. Lea, and D. Tolbert. "How to control if even experts are not sure: robust fuzzy control", *Proceedings of the Second International Workshop on Industrial Applications of Fuzzy Control and Intelligent Systems*, College Station, TX, 1992, pp. 153–162.

[SV81] Y. Shiloach and U. Vishkin. "Finding the maximum, merging, and sorting in a parallel computation model", *Journal of Algorithms*, 1981, Vol. 2, No. 1, pp. 88–102.

[THT71] S. Tamura, S. Higuchi, K. Tanaka. "Pattern classification based on fuzzy relations", *IEEE Transactions on Systems, Man, and Cybernetics*, 1971, Vol. SMC-1, No. 1, pp. 61–66; reprinted in [BP92], pp. 169–174.

[V85] L. G. Valiant. "Parallelism in comparison problems", *SIAM Journal of Computing*, 1985, Vol. 4, No. 3, pp. 348–355.

[Z71] L. A. Zadeh. "Similarity relations and fuzzy orderings", *Inform. Sci.*, 1971, Vol. 3, pp. 177–200; reprinted in [BP92], pp. 151–168.

TR1-9