# Optimal Computation of Global Sensitive Functions in Fast Networks

**Israel Cidon, Inder Gopal, and Shay Kutten**

*IBM T. J. Watson Research Center*
*Yorktown Heights, NY 10598*

## Abstract

The common practice in distributed algorithms research was to assume that the computation time is zero, and assign a cost only to the communication. This was justified at the times that the communication was the bottleneck of a network. However, lately there has been a dramatic increase in the capacity of communication links so that we can no longer make this assumption.

In this paper we find optimal algorithms for distributed computation of "global sensitive" functions in the case that the computation time is some given P, and the communication time is some given C. (Roughly speaking, a global sensitive function is one that depends on each of its inputs.)

## 1. Introduction

The common practice in distributed algorithms research was to assume that the computation time is zero, and assign a cost only to the communication. (See e.g. [GHS83]) This was justified at the times that the communication was the bottleneck of a network. However, lately there has been a dramatic increase in the capacity of communication links so that we can no longer make this assumption.

In [CGK88] we have investigated the complement assumption, i.e. that of zero communication cost, with a charge only for the computation. However, if the network covers long distances, then propagation delays may still be significant. In addition, the speed of the computation may increase in the future. Thus it makes sense to analyze distributed computations when the ratio between the computation time and the communication time is given as a parameter.

In this paper we make the first step, by analyzing the distributed computation of "global sensitive" functions in the case that the computing time is some given P, and the communication time is some given C. Roughly speaking, a global sensitive function is one that depends on every one of its inputs, though a more formal definition will be given later. Most "natural" functions are global sensitive. In addition to their theoretical importance they arise in many applications, e.g. call set-up [ACGKK90], directory search, parallel computing and more.

An optimal solution is presented for the above problem.

## 2. Model and Problem Definition

The communication network is represented by a graph $(V, E)$ where $V$ is the set of nodes and $E$ is the set of bidirectional communication links. We denote $|V|$ by $n$, and assume, without loss of generality, that the nodes are numbered $1, 2, ..., n$. In this paper we assume that the graph is complete.

Messages are assumed to arrive within finite but unbounded time. For purposes of time complexity analysis, we count computation and communication delays separately. (In the terms of [CGK88] The computation delay is the delay of the software, or *system call complexity* and the communication delay is the delay of the switching hardware including the links' propagation delays.) In particular, we assume that the communication delays are upper bounded by C time units and the computation delays are upper bounded by P time units. (Note that these upper bounds are introduced for the purpose of the time complexity analysis and the algorithm operates correctly even if delays are unbounded.) Thus we define

- Time Complexity - The time complexity is then the maximal time which may elapse from the algorithm starting point to its end under the assumption that the hardware and software delays are upper bounded as described above.

We assume that each node $i$ maintains some *independent* input value $I_i$ which is drawn from some finite alphabet. A distributed algorithm is triggered at time 0 at all nodes. The algorithm terminates eventually at node 1. Upon termination, node 1 should have computed the correct value of some function $f(I_1, I_2,..., I_n)$. $f$ is defined for all vectors of length equal or smaller than $n$.

The function $f$ is defined to be:

1. Associative - $f(X, Y, Z) = f(X,f(Y,Z)) = f(f(X,Y),Z)$
2. Commutative - $f(X,Y) = f(Y,X)$

In addition $f$ is **global sensitive in the weak sense.** This means that there exit at least one input vector $\bar{I} = (I_1, I_2,..., I_n)$ (which is called **a global sensitive input vector**) such that for all $j$, $(1 \leq j \leq n)$, if we define $\bar{I}_j$ to be equal to $\bar{I}$ except for the single input value $I_j$, then there exists some selection of $I_j$ such that $f(\bar{I}) \neq f(\bar{I}_j)$

A global sensitive function **in the strong sense** was defined [ALSY90]. There, the function must be different for **any** two input vectors that differ in one entry value. Our definition of global sensitive in the weak sense captures a larger class of functions including the basic Boolean functions (OR, AND) and cases in which some input values are not distinguishable.

## 3. Properties of Optimal Protocols

During the execution of the algorithm messages are sent. We distinguish between messages that can affect the final result and messages that cannot. A *Causal Message* is defined in the following recursive way. It is either received by node 1 before it terminates or it is received by some other node before that node sends a causal message. Any other message is defined to be a non-causal message. It is clear from the definition that a causal message sent over a link cannot be preceded by a non-causal message as we assume FIFO reception. If a message is received at a node before a causal message has been received then, this previous message must be causal as well. Lemma 1 shows that we can essentially ignore or suppress any non-causal message. We assume the operation of an arbitrary correct algorithm that computes the function $f$.

*Lemma 1*: Consider any arbitrary execution of the algorithm then, we can delay any non-causal message at a node or a link for any arbitrary amount of time without changing the algorithm's results or its termination time.

*Proof*: The proof follows immediately from the fact that non-causal messages cannot be received before a causal one at any node. Since causal messages cannot be sent after the reception of a non-causal message and since node 1 makes its final decision only due to reception of causal messages the lemma holds.

*Lemma 2*: Assume that a function $f$ is computed. Then, there exist at least one execution of the (correct) distributed algorithm in which every node sent at least one causal message whose information depends on its local input value.

*Proof*: Let $\bar{I}$ be a global sensitive input vector and assume an execution under this input vector. Define the termination time as T. By using lemma 1 we construct a new feasible execution of the algorithm by delaying all non-causal messages that have been sent before T, to be received after time T. Since we are dealing with an asynchronous network, this execution is still a feasible one. We now focus on the new execution.

Let A be the set of nodes that have sent a causal message (we include node 1 in A as well), and let $B = A^c$. If B is empty then the proof is completed. If B is not empty than no message sent from B is received at any node in A before T. This also means that the input values of nodes in B cannot affect the computation results before T. Now assume the operation of the algorithm with a different input value for a single node $j \in B$. Since input values of nodes in A are exactly as before, and no message from nodes in B are received until time T, repeating the exact execution of the algorithm at these node is a feasible execution. This means that no matter how the input value $I_j$ is chosen, node 1 terminates with the same result. This contradicts the correctness of the algorithm.

Following the results of lemma 2, for every correct algorithm (in particular optimal ones) there are executions in which each node must send at least one causal message. From now on we will focus on these particular executions.

Let us further assume that message delays do not depend on their contents. Then the following Lemma 3 can be applied.

*Definition*: A *tree based algorithm* is an algorithm where the function is computed over a predefined rooted tree (the same tree for all possible input vectors). The computation is done in the following: At initialization, all tree leaves send their input values to their parents. Each node (which is not a leaf) waits until it has received a message from all its children and then computes the partial function of its subtree and forward the result to its parent.

Since we assumed that message delays do not depend on their contents, a particular tree based algorithm will have the same worst case time-cost for all functions $f$ and all possible input vectors. We are looking for optimal algorithms in the worst case (over all executions and possible input vectors). Therefore, in order to prove the optimality of a tree based algorithm, it is sufficient to show that for any arbitrary correct algorithm that compute an arbitrary function $f$ there exists an input vector and a tree based protocol which is at least efficient as the original algorithm (only for this particular input vector).

*Lemma 3*: For any function $f$ and any correct algorithm that computes $f$ there exist a tree-based algorithm with worst case time and message delays better or equal to that algorithm.

*Proof*: Consider some execution of the algorithm for a global sensitive input vector. Consider the last causal message that is sent by each node. Since this is the last message then, the causal path of these messages define a tree rooted at node 1. (From the definition of causal messages this paths must define a connected structure. Since we are dealing with the last causal message, this structure cannot have any cycles.) Similarly, each node must wait for all its children last causal messages before it sends its message. Otherwise, some of these messages will be non-causal. This implies that this last causal messages process is equivalent to the operation of a tree based algorithm.

Now let us execute the tree-based algorithm over the above defined tree (for all input vectors). This algorithm is equivalent in its structure to the last causal messages sent by the original algorithm for a specific input vector. Therefore, the tree based algorithm just introduced, has worst case message and time cost which is bounded by a particular execution of the original algorithm (not necessarily the worst one). This implies that its time and message delays are bounded by the worst case delay of the original one.

*Corollary 1*: There exists a single tree-based algorithm which is worst-case optimal for all functions *f*.

## 4. Optimal Tree-based Algorithm

We assume that there are two worst-case delays associated with each message. We define C to be the worst case communication delay and P the worst case system call reception delay. The communication cost, reflects the time elapsed from starting the message transmission until it is received at the other hand. (This can be further split into transmission and propagation delays.) The system call delay is the time it takes to process the received message and accomplish the computation steps that are needed. We also assume that the initialization of the algorithm at some node (if it is not via a message reception) takes one system call - P.

We define an optimal (t,P,C) tree, as the rooted tree with the maximum number of nodes over which a tree-based algorithm can compute any function *f* and terminates no later than t. Fixing P and C, we denote this optimal tree by $OT(t)$ and the number of nodes in $OT(t)$ by $S(t)$. It is clear that the size of the optimal tree is a non-decreasing function of t. C and P were defined as worst case times. It is easy to prove that increasing any message delay in the tree-based algorithm never decrease the overall time to complete an execution. Therefore, we can assume that in the worst case situation messages encounter exactly the maximum allowable delays (P and C).

Let us consider an optimal tree OT(t). The root of this tree is receiving causal messages from its children. It is able to process all of them until time t. Let us observe the last message it processes. This message must be received no later than (t-P) otherwise it cannot terminate at t. This in turn implies that the message must have been sent before (t-P-C). In addition, assuming that messages are being processed in FIFO order, the root must have received and processed all other messages by that time (t-P). It is also clear that:

$$S(t) = \begin{cases} 0 & t < P \\ 1 & t < 2P + C \end{cases} \qquad (1)$$

Let $X$ and $Y$ be rooted trees. We define the operation $\overleftarrow{\bigcup}$ as an operation that get as input two rooted trees and output a new tree in the following way. The rooted tree $Z$ which is the result of $Z = X \overleftarrow{\bigcup} Y$ is constructed by augmenting the root of Y (we add a directed edge) as a child of the root X.

Since $S(t)$ is a non decreasing function of $t$, this lead us to the following recursive structure of the tree ( $t \geq 2P + C$).

$$OT(t) = OT(t-P) \overleftarrow{\bigcup} OT(t-C-P) \tag{2}$$

This also translates to the numerical equation:

$$S(t) = S(t-P) + S(t-C-P) \tag{3}$$

Both the tree structure and size can be solved using the recursions defined by (2) and (3) respectively and by the initial conditions (1). One has to compute $OT(t - i(C+P) - jP)$ for all $i,j$ such that $t - i(C+P) - jP \geq 2P + C$. The simplest way to do it is by computing the values of t, to be considered, ordering them from low to high and computing the structures and the values according to this order. Basically we can restrict the computation to discrete values of time that are in the form $iP + jC$. Other times $t$ can be truncated to the highest value that still satisfied $t \geq iP + jC$.

## 5. Examples

1) New model - C = 0, P = 1. We can assume only integer values of time. The equations take the following simple structure:

$$S(t) = \begin{cases} 0 & t < 1 \\ 1 & t < 2 \end{cases} \tag{4}$$

$$OT(k) = OT(k-1) \overleftarrow{\bigcup} OT(k-1) \tag{5}$$

This structure is a binomial tree [SCH81]. Solving for the tree size results in

$$S(k) = 2S(k-1) = 2^{k-1} \tag{6}$$

2) Old model - C = 1, P = 0. It is easy to see that the recursion is blowing up. This is because using a star configuration we can add any number of nodes to the structure in order to get any tree size for t = 1.

3)C = 1, P = 1

$$S(k) = \begin{cases} 0 & k < 1 \\ 1 & k < 3 \end{cases} \tag{7}$$

$$OT(k) = OT(k-1) \overleftarrow{\bigcup} OT(k-2) \tag{8}$$

This structure is growing slower than a binomial tree. Solving for the tree size results in

$$S(k) = S(k-1) + S(k-2) \tag{9}$$

Via Z transform it is easy to derive the equation:

$$S(Z) = \sum_{k=1}^{\infty} S(k)Z^k = \frac{-Z}{Z^2+Z-1} \tag{10}$$

Taking the inverse transform results in:

$$S(k) = \frac{\left((1+\sqrt{5})^k - (1-\sqrt{5})^k\right)}{\sqrt{5}\,(2)^k} \tag{11}$$

Note that these are the well known Fibonacci numbers.

### *Computing the optimal time for a given size*

Since the recursion is applied in the order of increasing value of time, one should just computes it for the discrete values of time until $S(t) \geq n$. As was mentioned before we should use only values of time that are in the form of $iP + jC$ where $i$ and $j$ are integers. Since in the tree-based algorithm there are only $n-1$ messages sent clearly: $i \leq n$, $j \leq n$. Therefore, there are at most $n^2$ possible number of points that should be computed.

## 6. Conclusion

Our model is a generalization of both the model in [CGK88] and the model of telephone networks of [SCH81].

This paper also clarifies a certain point regarding the models for fast networks [CGK88]. In that paper we assumed that nodes do not know in advance the path to some remote nodes. However, if node $v$ does know (or learned) the route to some other node $u$ (and no topological change has disconnected that route) then $v$ could have sent a message that reached node $u$ costing only one system call (i.e. P). This might lead to the (wrong) conclusion that once each node knows how to directly reach all other nodes, the difference between the old model and the new class of models diminishes. The current paper demonstrates that even in complete networks, charging for the computation does make the model different than the traditional model.

An interesting open problem is to generalized the problem investigated here for other graphs.

## References


**[ACGKK90]**, A.Awerbuch, I. Cidon, I.S. Gopal, M. Kaplan and S. Kutten, "Distributed Control for PARIS," to appear in *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing*, Quebec City, Canada, August 1990.

**[ALSY90]** Y. Afek, G.M. Landau, B. Schieber and M. Yung, "The Power of Multimedia: Combining Point-to-Point and Multiaccess Networks", *Information and Computation* vol. 84 no. 1 january 1990, pp. 97-118.

**[CGK88]** I.Cidon, I.S.Gopal, S.Kutten, "New Models and Algorithms for Future Networks," *Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing* Toronto, Ontario, Canada, August 1988, pp. 75-89.

**[SCH81]** Slater, P.J., E.J. Cockayne, and S.T. Hedetniemi, "Information dissimination in Trees," *SIAM J. on Computing* 10 (1981), pp. 692-701.

**[GHS83]** Gallager, R.G., P.M. Humblet and P.M. Spira, "A Distributed Algorithm for Minimum- Weight Spanning Trees", ACM Transactions on Programming Languages and Systems, January 1983, Vol. 5, No. 1, pp. 67-77.