Allowing Each Node to Communicate Only Once in a Distributed System: Shared Whiteboard Models

Florent Becker
LIFO, Université d'Orléans
florent.becker@univ-orleans.fr

Adrian Kosowski
LaBRI, INRIA
Bordeaux Sud-Ouest
kosowski@labri.fr

Nicolas Nisse
CNRS / Université Nice
Sophia-Antipolis
nicolas.nisse@sophia.inria.fr

Ivan Rapaport
DIM-CMM (UMI 2807 CNRS)
Universidad de Chile
rapaport@dim.uchile.cl

Karol Suchan
UAI (Chile) and U of Science
and Technology (Poland)
karol.suchan@uai.cl

ABSTRACT

In this paper we study distributed algorithms on massive graphs where links represent a particular relationship between nodes (for instance, nodes may represent phone numbers and links may indicate telephone calls). Since such graphs are massive they need to be processed in a distributed and streaming way. When computing graph-theoretic properties, nodes become natural units for distributed computation. Links do not necessarily represent communication channels between the computing units and therefore do not restrict the communication flow. Our goal is to model and analyze the computational power of such distributed systems where one computing unit is assigned to each node. Communication takes place on a whiteboard where each node is allowed to write at most one message. Every node can read the contents of the whiteboard and, when activated, can write one small message based on its local knowledge. When the protocol terminates its output is computed from the final contents of the whiteboard. We describe four synchronization models for accessing the whiteboard. We show that message size and synchronization power constitute two orthogonal hierarchies for these systems. We exhibit problems that *separate* these models, i.e., that can be solved in one model but not in a weaker one, even with increased message size. These problems are related to maximal independent set and connectivity. We also exhibit problems that require a given message size independently of the synchronization model.

Categories and Subject Descriptors

F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*Parallelism and Concurrency*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SPAA'12, June 25–27, 2012, Pittsburgh, Pennsylvania, USA.
Copyright 2012 ACM 978-1-4503-1213-4/12/06 ...$10.00.

General Terms

Algorithms, Theory

Keywords

Distributed computing, local computation, graph properties, bounded communication

1. INTRODUCTION

A distributed system is typically represented by a graph where links correspond to a particular relationship between nodes. For instance, nodes may represent phone numbers and links may indicate telephone calls. A classical approach is to view each node as a processor. Since nodes lack global knowledge, new algorithmic and complexity notions arise. In contrast with classical algorithmic theory – where the Turing machine is the consensus formal model of algorithm – in distributed systems many different models are considered. Under the paradigm that communication is much slower and more costly than local computations, complexity analysis of distributed algorithms mainly focuses on message passing. That is, an important performance measure is the number and the size of messages that are sent by nodes for performing some computation. Theoretical models were conceived for studying particular aspects of protocols such as fault-tolerance, synchronism, locality, congestion, etc.

The particularity of this work lies in the fact that links between nodes do not necessarily represent communication channels between the computing units and therefore do not restrict the communication flow. In that sense our setting is similar to the "mud" (massive, unordered, distributed) model, where the authors tackle the problem of performing a computation when the data is distributed among many machines [4]. Roughly, in such mud algorithms, pieces of data are processed independently in parallel and pairs of messages are aggregated in any order. Only one message is created by each node because in truly massive database "a common approach for dealing with large datasets is to stream over the input in one pass" [4].

The problem we intend to model here is less general than the one addressed in [4]. In fact, in our setting *there exists an underlying graph and the information each node possesses is nothing but its neighborhood.* The computation the nodes need to perform collectively is related to some property of

the graph. In [2] the first simple model for studying such scenario was introduced. In that model, the total amount of local information that each node was allowed to provide was bounded by $O(\log n)$ bits. Each node transmitted its message to a central authority, the referee, that collected and used them in order to give the correct output.

The main question was whether this small amount of local information provided by each node was sufficient for the referee to decide some basic structural properties of the graph G. For instance, simple questions like "Does G contain a square (cycle of length 4)?" or "Is the diameter of G at most 3?" cannot be solved. On the other hand, the referee can decode the messages in order to have full knowledge of G when G belongs to one of many graph classes such as planar graphs, bounded treewidth graphs and, more generally, bounded degeneracy graphs.

In this paper we define extensions of the model in [2] and investigate their computational power. It is interesting to point out that despite being extremely natural, these models of computation have never been studied before.

Communication using a shared whiteboard. The computational model in [2] can be stated equivalently in the following form. Given a question about the graph, every node writes *simultaneously* one message (computed from its local knowledge) on a global zone of shared memory, a *whiteboard*, and then one must be able to answer the question using only the contents of the whiteboard.

In this paper we intend to give more power to the initial model of [2]. For this purpose, we relax the simultaneity constraint in different ways. Roughly, messages may be written sequentially on the whiteboard. This allows nodes to compute their messages taking into account the contents of the whiteboard, i.e., the messages, that have previously been written. In other words, in the new models we propose, nodes have more sophisticated ways to share information. Basically, the four models we now present aim at describing how the nodes can access the shared medium, in particular, differentiating synchronous and asynchronous networks.

We define a framework for synchronization without using a global clock. Instead, time is divided into *rounds* corresponding to observable events, i.e., whiteboard modifications. More precisely, a round terminates when a node writes a message on the whiteboard. Along the evolution of the system, the nodes may be in three states: *awake*, *active* or *terminated*. Initially, all nodes are awake. A node becoming active means that this node would like to write a message on the whiteboard. Metaphorically speaking, it "rises its hand to speak". To model the worst-case behavior, an *adversary* chooses, among the set of active nodes, the particular node which is going to write a message on the whiteboard. Afterwards, this node enters the terminated state. Therefore, a node is in state terminated when its message has been written on the whiteboard. In one round, several awake nodes may become active but exactly one active node becomes terminated. Note that a node may become active and terminated in one round. In our model, if a node is active in round i and it does not write a message then it must stay active in round $i+1$. In other words, once a node raises its hand it cannot "change its mind" later. After the last round, when all nodes are terminated, all of them must be able to answer the question by using *only* the information stored on the whiteboard.

In this setting, we propose several scenarios leading to the definition of four computational models. A computational model is said *simultaneous* if all nodes become active (raise their hands) at the beginning of the process. On the other hand, the model is said *free* if, in every round, any awake node may decide to become active based on its knowledge and on its own protocol. The other criterion we use to distinguish models is the state-transition during which a node must create the message it will eventually write on the whiteboard. In the *asynchronous* scenario, the nodes must create their message *as soon as they become active*. In the *synchronous* scenario, every node is allowed to create its message later, when the adversary chooses it to write the message on the whiteboard. Thus, in the asynchronous case, there may be some delay between the creation of a message and the step when it is written. In particular, the order in which the messages are created and the order in which they are actually available on the whiteboard may differ. In this way, we can model real-world asynchronous systems where there are no guarantees on the time of communications.

Our results. In this work we define four families of systems, namely SimAsync$[f(n)]$, SimSync$[f(n)]$, FreeAsync$[f(n)]$ and FreeSync$[f(n)]$, which correspond to the four possible free/simultaneous, asynchronous/synchronous scenarios, parametrized by the amount $f(n)$ of data (in bits) each node is allowed to write on the whiteboard. We show that these classes form a hierarchy from the point of view of message size as well as from the point of view of the synchronization mechanism. More precisely, for any $f(n) = o(n)$, we show that SimAsync$[f(n)] \subsetneq$ SimSync$[f(n)] \subsetneq$ FreeAsync$[f(n)] \subseteq$ FreeSync$[f(n)]$; the strictness of the last inclusion is left as an open problem. On the other hand, we also prove that when $g(n) = o(f(n))$, FreeSync$[g(n)] \subsetneq$ SimAsync$[f(n)]$. This means that message size and synchronization mechanisms are two orthogonal parameters with respect to the power of each instance of our model.

Connectivity problems in general, and breadth-first search (BFS) in particular, are classical problem in distributed computing, and we examine their positions in our hierarchy. We show that BFS is in the class FreeSync$[\log n]$, and that for the bipartite case, it is in FreeAsync$[\log n]$. We also show that for all $f(n) = o(n)$, BFS is not in the class SimSync$[f(n)]$ even in the bipartite case.

Related work. The two main aspects of our approach, the *locality* and the fact that the nodes are allowed to send *only one short message* have been tackled before. In the classical model $\mathcal{CONGEST}$ [8], where a network is represented by a graph whose nodes correspond to network processors and edges to inter-processor links, the n processors can send in each round a message of size $O(\log n)$ bits through each of its outgoing links. A restriction of the $\mathcal{CONGEST}$ model has been proposed by Grumbach and Wu to study *frugal* computation [5]. In this model, where the total amount of information traversing each link is bounded by $O(\log n)$ bits, they showed that any first order logic formula can be evaluated in any planar or bounded degree network [5]. Many variations to the $\mathcal{CONGEST}$ model have been proposed in order to focus on different aspects of distributed computing. In a seminal paper, Linial introduced the $\mathcal{LOCAL}$ model [7, 8]. In the $\mathcal{LOCAL}$ model, the restriction on the size of messages is removed so that every vertex is allowed to send unbounded size messages in every round. This model fo-

cuses on the issue of locality in distributed systems, and more precisely on the question "*What cannot be computed locally?*" [6]. Difficult problems like minimum vertex cover and minimum dominating set cannot be well approximated when processors can locally exchange arbitrary long messages during a bounded number of rounds [6].

The idea of abstracting away from the cost of *transmitting* data throughout the network and to look at *how much local information must be shared* in order to compute some property is present in the the Simultaneous Message Model defined in [1]. In such model the communication is *global*: n players must evaluate a function $f(x_1, \ldots, x_n)$ in which player i knows the whole input except x_i. Each player directly transmits *one message* to a central authority, the referee, that collects and uses them in order to compute $f(x_1, \ldots, x_n)$. The Simulateouus Message Model is a variant of the more general Multiparty Communication model, where the n players communicate by writing messages on a common whiteboard [3].

2. COMMUNICATION MODELS

Our protocols work on simple undirected connected n-node graphs. In $G = (V, E)$, each node $v \in V$ has a unique identifier $ID(v)$ between 1 and n. Typically, $V = \{v_1, \ldots, v_n\}$, where v_i is such that $ID(v_i) = i$. Throughout the paper, a graph should be understood as a labeled graph. At each node $v \in V$ there is an independent processing unit that knows its own identifier, the identifier of each of its neighbors and the total number of nodes n. Each node is in one of three *states*: awake, active or terminated. Initially, they are all awake.

All nodes execute the same algorithm. Roughly, if they are in the awake state, they must decide whether to become active, and if they are active, what to write on the whiteboard. Each node is allowed to write exactly one message on the whiteboard. Once they write the message they enter the terminated state. The size of these messages, in bits, is some $f(n) = o(n)$, typically $O(\log n)$.

Let $W_{n,s}$ be the set of possible configurations of the *whiteboard* with at most n messages of size at most s bits each.

We first define synchronous protocols, then asynchronous protocols. Synchronous protocols rely on some external synchronization primitives to ensure that messages are delivered one by one, whereas asynchronous protocols have to deal with concurrent messages, which means that messages are created as soon as the nodes become active. We also distinguish between simultaneous and free protocols. In simultaneous protocols, nodes must be ready to speak at any time, whereas in free protocols, they can decide when to become active. We get the following four family of models, with $f(n)$ representing the message size:

	message created when node becomes active
all nodes initially active	SIMASYNC$[f(n)]$
no node initially active	FREEASYNC$[f(n)]$

	message created when node is chosen
all nodes initially active	SIMSYNC$[f(n)]$
no node initially active	FREESYNC$[f(n)]$

2.1 Synchronous protocols

DEFINITION 1. *Let n be a positive integer. Let $[1, n] = \{1, \ldots, n\}$. A* synchronous protocol *with output set O and message size $f(n)$ is a triplet $\mathcal{A} = (act_n, msg_n, out_n)$ where:*

- *act_n: $[1, n] \times 2^{[1,n]} \times W_{n,f(n)} \to \{awake, active\}$ is the activation function, which depending on the node's identifier, its neighborhood and the contents of the whiteboard decides whether to become active or stay awake.*
- *msg_n: $[1, n] \times 2^{[1,n]} \times W_{n,f(n)} \to \{0, 1\}^{f(n)}$ is the message function, which depending on the node's identifier, its neighborhood and the contents of the whiteboard decides what to write on the whiteboard.*
- *out_n: $W_{n,f(n)} \to O$ is the output function.*

Let FREESYNC$[f(n)]$ be the set of all synchronous protocols with message size at most $O(f(n))$. A *configuration* of a protocol corresponds to a configuration of the whiteboard in $W_{n,f(n)}$ together with a state in $\{\text{awake, active, terminated}\}^n$ (which must be interpreted as the state of each node). In the *initial configuration*, all nodes are awake and the whiteboard is empty.

A *round* corresponds to an observable transition of the protocol, which in practice occurs when a message is written on the whiteboard. In a round, awake nodes may decide to become active and one active node will write its message.

DEFINITION 2. *Consider the synchronous protocol $\mathcal{A} = (act_n, msg_n, out_n)$. Let $G = (V, E)$ be an n-node graph. A* round *goes from configuration C to configuration C' if:*

- *Any terminated node in C is also terminated in C'.*
- *For any node $v_i \in V$ which is awake in C, its state in C' is $act_n(i, N(v_i), W_C)$, where $N(v_i)$ is the set of identifiers of v_i's neighbors and W_C is the content of the whiteboard in configuration C.*
- *The configuration of the whiteboard $W_{C'}$ is the same as W_C but where the message $msg_n(j, N(v_j), W_C)$ is attached provided that there exists at least one active node v_j in C. In C' the node v_j enters the terminated state. Every other active node stays active.*

DEFINITION 3. *An* execution *of a protocol $\mathcal{A}$ is a (finite) sequence of configurations starting from the* initial configuration *where transitions are determined by rounds. The execution is* successful *if in the last configuration, all nodes are terminated. We say that the execution ends in a* deadlock *when we end up in a situation where there are no active nodes but the set of awake nodes is not empty. For a successful execution, where the last whiteboard configuration is W, we define the* output *to be $out_n(W)$.*

DEFINITION 4. *For a function F_n defined from the set of n-node graphs to an output set O, we say that $\mathcal{A}$ computes F_n if for all G, all maximal executions of $\mathcal{A}$ are successful and output $F_n(G)$. Therefore, we can assume that there is an adversary that chooses in each round which active node writes a message on the whiteboard.*

DEFINITION 5. *A protocol is* simultaneous *if all nodes are active from the beginning (the activation function is uniformly active). We note* SIMSYNC$[f(n)]$ *the set of simultaneous protocols. In this subclass of protocols there will never be deadlocks.*

2.2 Asynchronous protocols

In asynchronous protocols nodes create their messages as soon as they become active. Therefore, if two nodes become active simultaneously, then the first message written on the whiteboard does not affect the second message.

DEFINITION 6. *Let n be a positive integer. Let $[1,n] = \{1,\ldots,n\}$. An* asynchronous protocol *with output set O and message size $f(n)$ is a pair $\mathcal{A} = (act/msg_n, out_n)$ where:*

- *act/msg_n: $[1,n] \times 2^{[1,n]} \times W_{n,f(n)} \to \{awake, active\} \times \{0,1\}^{f(n)}$ is the activation/message function. Note that this transition is such that a (nonempty) message is created only when the node enters the active state.*
- *out_n: $W_{n,f(n)} \to O$ is the output function.*

Let FREEASYNC$[f(n)]$ be the set of asynchronous protocols with message size $O(f(n))$.

The definition of configuration is similar to the synchronous case, except that since active nodes create their messages (following the *act/msg* function) before being chosen by the adversary for writing on the whiteboard (eventually), these messages are part of the configuration, despite the fact that they do not appear on the whiteboard.

We define rounds, executions, deadlocks, successful executions and computations as in the synchronous case. We also define the set SIMASYNC$[f(n)]$ of simultaneous asynchronous protocols.

This paper aims at deciding what kind of problems can be solved in each of these models. For instance, [2] proves that deciding if a graph has degeneracy k, $k \geq 1$, can be solved in SIMASYNC$[\log n]$. On the negative side, deciding whether a graph contains a triangle as a subgraph and deciding whether a graph has diameter at most 3 cannot be solved in SIMASYNC$[\log n]$ [2].

3. A COMPUTING POWER LATTICE

First of all, we prove the following lemma that extends a result of [2]. Let BUILD be the problem that consists in computing the adjacency matrix of a graph.

LEMMA 1. *Let $\mathcal{G}$ be a family of n-node graphs, and $g(n)$ be the number of graphs in $\mathcal{G}$. Let $f(n) = o(n)$, and $\mathcal{C} \in \{$SIMASYNC, SIMSYNC, FREEASYNC, FREESYNC$\}$. If the problem* BUILD*, when the input graphs are restricted to the class $\mathcal{G}$, can be solved in the model $\mathcal{C}[f(n)]$ then $\log g(n) = O(n(f(n) + \log n))$.*

PROOF. Consider any algorithm in one of the four considered models. In any model, at the end of the communication process, n messages of size $O(f(n))$ bits are written on $\mathcal{B}$. Hence, at the end, accounting for the order of the messages, a total of $O(nf(n) + \log n)$ bits are available on the whiteboard. For the output function to distinguish two different graphs in $\mathcal{G}$, we must have $\log g(n) = O(n(f(n)+\log n))$. □

For the ease of descriptions, in what follows we will not define explicitly the functions for activation, message creation and decision. Nevertheless, they always will be clear from the context.

In this section we intend to show that these models form a lattice in which the computational power grows strictly whenever either the syncronization model is enriched or the message size is increased. On the other hand, when one resource is increased but the other restricted then the resulting class is incomparable with the original. (neither is included in the other) The main result of this section is the following theorem:

THEOREM 1. *For all $\Omega(\log n) = f(n) = o(n)$,* SIMASYNC$[f(n)] \subsetneq$ SIMSYNC$[f(n)] \subsetneq$ FREEASYNC$[f(n)] \subseteq$ FREESYNC$[f(n)]$.

We start with the following weaker result:

LEMMA 2. *For all $f(n)$,* SIMASYNC$[f(n)] \subseteq$ SIMSYNC$[f(n)] \subseteq$ FREEASYNC$[f(n)] \subseteq$ FREESYNC$[f(n)]$.

PROOF. **SimAsync**$[f(n)] \subseteq$ **SimSync**$[f(n)]$**.** In the SIMSYNC model, any node applies directly the protocol of the SIMASYNC model. Nodes create their message initially, ignoring the messages present on the whiteboard when they write their own.

SimSync$[f(n)] \subseteq$ **FreeAsync**$[f(n)]$**.** Recall that a problem is solved in the SIMSYNC model if the nodes compute the output *no matter* the order chosen by the adversary. So we can translate a SIMSYNC protocol into a FREEASYNC one if we fix an order (for instance $v_1,\ldots,v_n$) and use this order for a sequential activation of the nodes.

FreeAsync$[f(n)] \subseteq$ **FreeSync**$[f(n)]$**.** The situation is that of the first inclusion. It suffices to force the protocols in FREESYNC to create their messages based only on what was known at the moment when they became active. □

3.1 SIMASYNC vs. SIMSYNC

We consider here a "rooted" version of the INCLUSION MAXIMAL INDEPENDENT SET problem. This problem, denoted by MIS(x), takes as input an n-node graph $G = (V,E)$ together with an identifier $ID(x)$, $x \in V$, and the desired output is any maximal (by inclusion) independent set containing x.

THEOREM 2. MIS(x) *can be solved in the* SIMSYNC$[\log n]$ *model.*

PROOF. Recall that in the SIMSYNC model, all nodes are initially active and that the adversary chooses the ordering in which the nodes write their messages. Hence, an algorithm in this model must specify the message created by a node v, according to the local knowledge of v and the messages written on the whiteboard before v is chosen by the adversary.

The protocol is trivial (it is the greedy one). When node v is chosen by the adversary, the message of v is either its own ID (meaning that v belongs to the final independent set) or v writes "no" (otherwise). The choice of the message is done as follows. The message is $ID(v)$ either if $v = x$ or if $v \notin N(x)$ and $ID(y)$ does not appear on the whiteboard for any $y \in N(v)$. Otherwise, the message of v is "no".

Clearly, at the end, the set of vertices with their IDs on the whiteboard consists of an inclusion maximal independent set containing x. □

THEOREM 3. *For any $f(n) = o(n)$,* MIS(x) *cannot be solved in the* SIMASYNC$[f(n)]$ *model.*

PROOF. Let $f(n) = o(n)$. We proceed by contradiction. Let us assume that there exists a protocol $\mathcal{A}$ for solving MIS(x) in the SIMASYNC$[f(n)]$ model. Then we show how to design an algorithm $\mathcal{A}'$ to solve the BUILD Problem for any graph in this model, contradicting Lemma 1.

Let $G = (V, E)$ be a graph with $V = \{v_1, \ldots, v_n\}$. For any $1 \leq i < j \leq n$, let $G^{(x)}_{i,j}$ be obtained from G by adding a vertex x adjacent to every vertex in V with the exception of v_i and v_j. Note that $\{x, v_i, v_j\}$ is the only inclusion maximal independent set containing x in $G^{(x)}_{i,j}$ if and only if $\{v_i, v_j\} \notin E$. Indeed, if $\{v_i, v_j\} \in E$, there are two inclusion maximal independent sets containing x: $\{x, v_i\}$ and $\{x, v_j\}$.

Recall that, in the SIMASYNC model, all nodes must create their message initially, i.e., while the whiteboard is still empty. Hence, the message created by a node only depends on its local knowledge

Notice that, for a given k, the node v_k only sees two different neighborhoods for all the possible $G^{(x)}_{i,j}$, depending on whether $k \in \{i, j\}$ or $k \notin \{i, j\}$. Therefore, we call m_k the message that v_k generates when $k \in \{i, j\}$ (i.e., x and v_k are not neighbors) and m'_k the message v_k generates when $k \notin \{i, j\}$ (i.e., x and v_k are neighbors).

From the previous protocol $\mathcal{A}$ we are going to define another protocol $\mathcal{A}'$ in the SIMASYNC$[f(n)]$ model which solves the BUILD Problem for any graph. Protocol $\mathcal{A}'$ works as follows. Every node v_k generates the pair (m_k, m'_k) of the two messages v_k would send in $\mathcal{A}$ when it is adjacent to x and when it is not. Clearly, this consists of $O(f(n))$ bits.

Now let us prove that any node can reconstruct $G = (V, E)$ from the messages generated by $\mathcal{A}'$. More precisely, for any $1 \leq s < t \leq n$, any node can decide whether $\{v_s, v_t\} \in E$ or not. It is enough for any node to simulate de decision function of $\mathcal{A}$ in $G^{(x)}_{s,t}$ by using messages m_s, m_t and $\{m'_k : k \in \{1, \cdots, n\} \setminus \{s, t\}\}$. Since the output of $\mathcal{A}$ is $\{x, v_s, v_t\}$ if and only if $\{v_s, v_t\} \notin E$, the results follows. This would mean that from $O(nf(n))$ bits we can solve BUILD in the class of all graphs, a contradiction. □

COROLLARY 1. *For all* $\Omega(\log n) = f(n) = o(n)$, SIMASYNC$[f(n)] \subsetneq$ SIMSYNC$[f(n)]$.

We discuss now another problem that could possibly separate the two models. Given an $(n-1)$-regular $2n$-node graph G, the 2-CLIQUES problem consists in deciding whether G is the disjoint union of two complete graphs with n vertices.

It is easy to show that 2-CLIQUES can be solved in the SIMSYNC$[\log n]$ model. Indeed, a trivial protocol can partition the vertices into two cliques numbered 0 and 1 if the input consists of two cliques, or otherwise indicate that it is not the case. The first vertex u to be chosen by the adversary writes $(ID(u), 0)$ on $\mathcal{B}$. Then, each time a vertex v is chosen, it writes $(ID(v), 0)$ if it "believes" to be in the same clique as u, and $(ID(v), 1)$ otherwise. More precisely, let S_v be the subset of neighbors of v that have already written a message on the whiteboard. If $S_v = \emptyset$ then v writes 1. If all nodes in S_v have written that they belong to the the same clique $c \in \{0, 1\}$ then v writes c, and v writes "no" otherwise. Clearly, G is the disjoint union of two cliques if and only if there is no message "no" on the whiteboard at the end of the communication process.

Proving that the problem 2-CLIQUES cannot be solved in the SIMASYNC$[f(n)]$ model (either for $f(n) = \log n$ or for any other $f(n)$) is an interesting question because it would allow us to show that CONNECTIVITY (deciding whether a graph is connected or not) cannot be solved in the SIMASYNC model. Indeed, it is easy to show that an $(n-1)$ regular $2n$-node graph is the disjoint union of two cliques if and only if it is not connected. We leave this as an open question:

OPEN PROBLEM 1. *For which* $f(n)$ *can* 2-CLIQUES *be solved in the* SIMASYNC$[f(n)]$ *model?*

3.2 SIMSYNC vs. FREEASYNC

We say that a graph is *even-odd-bipartite* if there are no edges between nodes having identifiers with the same parity. For separating models SIMSYNC and FREEASYNC, the problem we are going to introduce is EOB-BFS. In this problem, the input is an arbitrary n-node graph G and the output is a BFS-tree (or BFS-forest) if G is even-odd bipartite, and a negative answer otherwise. The root of the BFS-tree in each connected component of G will be the node with the smallest identifier in the respective component.

THEOREM 4. *The problem* EOB-BFS *can be solved in the* FREEASYNC$[\log n]$ *model.*

PROOF. Let G be the input graph. All nodes detecting that they have a neighbor with the same parity become active and create a message saying that this is an "invalid" graph. So we are going to define our algorithm assuming that G is indeed even-odd-bipartite.

The protocol will activate the nodes layer by layer in the BFS-forest. The first node to become active is v_1, then all its neighbors, then all nodes at distance 2, and so on. When all nodes in layer k have written their messages, then the information appearing in the whiteboard will be sufficient to compute the number of edges crossing between layer k and layer $k+1$ (if such number is 0 then that would mean that another connected component must be activated).

Initially, only v_1 is active. Let N^*_v be the set of neighbors of v *that have already written a message* on the whiteboard. When node v becomes active it creates the message $(i(v), l(v), p(v), d_{-1}(v), d_{+1}(v))$ where:

$i(v)$	is its ID
$l(v)$	$= min_{w \in N^*_v} l(w) + 1$
$p(v)$	is the node in N^*_v with minimum ID, or ROOT if N^*_v is empty
$d_{-1}(v)$	$= \lvert N^*_v \rvert$
$d_{+1}(v)$	$= d(v) - \lvert N^*_v \rvert$, with $d(v)$ its degree

$l(v)$ represents the level of v, $d_{-1}(v)$ its degree towards the previous level, $d_{+1}(v)$ its degree towards the next level and $p(v)$ its parent in the BFS-forest. The message created by v_1 at the beginning is $(1, 0, \text{ROOT}, 0, d(v_1))$. Since v_1 is the only active node the adversary is forced to choose it and v_1 writes its message on the whiteboard. Then all the neighbors of v_1 become active and, since we want all nodes of the same layer to become active simultaneously, the protocol works as follows. An arbitrary node v becomes active (and computes its message) if the next two conditions are satisfied:

1. A neighbor w of v has already written its message on the whiteboard.
2. $\Sigma_{u \in L_{l(w)}} d_{-1}(u) = \Sigma_{u \in L_{l(w)-1}} d_{+1}(u)$, where L_k is the set of nodes in layer k that have already written a message.

The key argument is to see that the second condition for activation ensures that all edges from layer $k-1$ to layer k have been written their messages before layer $k+1$ is activated.

Previous protocol works correctly if the graph has only one connected component. In order to avoid any deadlock we have to add another condition for becoming activated. The idea is to verify that a component has already been covered. More precisely, v becomes activated if the last message was written by a non-neighbor node w of v and the following three conditions are satisfied:

1. $\Sigma_{u \in L_{l(w)}} d_{+1}(u) = 0$.
2. $\Sigma_{u \in L_{l(w)}} d_{-1}(u) = \Sigma_{u \in L_{l(w)-1}} d_{+1}(u)$.
3. The ID of v is the minimum among the nodes that have not written a message yet.

These condition ensure that when the active connected component changes, exactly one node is activated. In the end, the output function corresponds to the forest indicated by the $p(v)$ from each message. □

THEOREM 5. *For any $f(n) = o(n)$,* EOB-BFS *cannot be solved in the* SIMSYNC$[f(n)]$ *model.*

PROOF. We proceed by contradiction. Let us assume that there exists a protocol $\mathcal{A}$ for solving EOB-BFS in SIMSYNC$[f(n)]$ for some $f(n) = o(n)$. The idea is to construct a protocol $\mathcal{A}'$ for solving the BUILD problem for even-odd-bipartite graphs in SIMASYNC$[f(n)]$, in contradiction with Lemma 1. Note that there are $2^{O(n^2)}$ even-odd-bipartite graphs with n vertices.

Let $G = (V, E)$ be an even-odd-bipartite graph with $V = \{v_1, \ldots, v_{n-1}\}$. Assume (w.l.o.g) that n is odd, and renumber the vertices so that $V = \{v_2, \ldots, v_n\}$.

Let $V' = \{v_1, v_{n+1}, v_{n+2}, \ldots, v_{2n-1}\}$. Let $3 \leq i \leq n$ be odd. We are going to define the auxiliar even-odd-bipartite graph $G_i = (V \cup V', E \cup E_i)$ where the edges E_i are defined as follows: connect v_1 with v_{i+n-2}, v_j with v_{j+n-2} for every $3 \leq j \leq n$ odd and v_j with v_{j+n} for every $2 \leq j \leq n-1$ even.

Suppose now that we run $\mathcal{A}$ on G_i. A node v_j is at level 3 of the BFS-tree rooted in v_1 if and only if v_i and v_j are neighbors in G. Thus, if we simulate $\mathcal{A}$ on every G_i (i.e., for all $3 \leq i \leq n$ odd) at once, then we would solve BUILD in SIMSYNC$[f(n)]$.

Note that if we run $\mathcal{A}$ on each of the G_i's with the nodes activated in order $(v_2, v_3, \ldots, v_{2n-1}, v_1)$ then the messages written by the nodes in $V = \{v_2, \ldots, v_n\}$ will not depend on the choice of i. In fact, the neighbourhood of all of these nodes is the same in every G_i, and their messages can only depend on such neighborhoods and the previous messages.

We then define $\mathcal{A}'$ to be the protocol in which each node in G sends the message it would send in any of the G_i's when running $\mathcal{A}$. Once all these messages have been collected, $\mathcal{A}'$ simulates $\mathcal{A}$ for every G_i in order to compute the neighbourhood of v_i. Thus, EOB-BFS is not in FREEASYNC$[f(n)]$. □

COROLLARY 2. *For all $\Omega(\log n) = f(n) = o(n)$,* SIMSYNC$[f(n)] \subsetneq$ FREEASYNC$[f(n)]$.

3.3 Message size

Obviously, by increasing the size of the messages we make the system more powerful. What is more interesting is that this resource is orthogonal (independent) to the synchronization power. We have already seen in previous section that MIS(x) $\in$ SIMSYNC$[\log n]$ but MIS(x) $\notin$ SIMASYNC$[o(n)]$. In other words, there are problems that can not be solved if we go down in the sinchronization hierarchy *no matter the extra length given to the size of the messages.* Now we are going to prove a more general result.

THEOREM 6. *Let $f(n) = o(n)$.* SUBGRAPH$_f$ *is the problem where the input is an n-node graph $G = (V, E)$ and the output is the subgraph obtained by removing all edges between nodes in $\{v_{f(n)+1}, \ldots, v_n\} \subseteq V$. Let $g(n) = o(f(n))$. It follows that* SUBGRAPH$_f \in$ SIMASYNC$[f(n)]$ *but* SUBGRAPH$_f \notin$ FREESYNC$[g(n)]$.

PROOF. It is obvious that SUBGRAPH$_f \in$ SIMASYNC$[f(n)]$. In fact, each node sends a vector consisting of the $f(n)$ first bits of its line in the adjacency matrix of the graph. Let $g(n) = o(f(n))$. SUBGRAPH$_f$ cannot be in FREESYNC$[g(n)]$, since that would allow us to solve BUILD for graphs of size n where $\{v_{f(n+1)}, \ldots, v_n\}$ are isolated nodes. This contradicts Lemma 1 because these graphs need $2^{O(nf(n))}$ bits to be defined. □

4. CONNECTIVITY AND RELATED PROBLEMS

One of the main questions arising in distributed environments concerns connectivity. For instance, one important task in wireless networks consists in computing a connected spanning subgraph (e.g., a spanning tree) since the links of such subgraph are used for communication.

In Section 3 we proved that, in the FREEASYNC$[\log n]$ model, it is possible to compute a BFS-forest for even-odd-bipartite graphs (i.e., bipartite graphs where the bipartition is *fully known* to every node). In such model it is in fact possible to get a protocol which outputs a BFS-forest *for all bipartite graphs without knowledge of the bipartition.* In the case of a non-bipartite graph though, running this protocol can result in a deadlock: at some point, no more nodes are activated. With synchronzation, as we are going to see in the next theorem, we do not need the graph to be bipartite and BFS *can be solved in the general case*, for arbitrary input graphs. Formally, the input of problem BFS is an arbitrary n-node graph G and the output is a BFS-tree (or BFS-forest). The root of the BFS-tree in each connected component of G will be the node with the smallest identifier in the respective component.

THEOREM 7. BFS *can be solved in the* FREESYNC$[\log n]$ *model.*

PROOF. The protocol is very similar to the one we used for EOB-BFS, but we need to keep track of edges within a level (these edges do not exist in the bipartite case).

Initially, only v_1 is active. Let N_v^* be the set of neighbors of v *that have already written a message* on the whiteboard. When node v becomes active it creates the message $(i(v), l(v), p(v), d_{-1}(v), d_0(v), d_{+1}(v))$ where:

$i(v)$ is its ID
$l(v)$ $= min_{w \in N_v^*} l(w) + 1$
$p(v)$ is the node in N_v^* with minimum ID, or ROOT if N_v^* is empty
$d_{-1}(v)$ $= |\{w \in N_v^* : l(w) = l(v) - 1\}|$
$d_0(v)$ $= |\{w \in N_v^* : l(w) = l(v)\}|$
$d_{+1}(v)$ $= d(v) - d_{-1}(v)$, with $d(v)$ its degree

Consider nodes v at distance al least 2 from the ROOT. These nodes v become active if the condition $(1 \wedge 2) \vee 3$ is satisfied, where

1. A neighbor w of v has already written its message on the whiteboard.
2. $$\sum_{u \in L_{l(w)}} d_{-1}(u) = \sum_{u \in L_{l(w)-1}} d_{+1}(u) - 2 \sum_{u \in L_{l(w)-1}} d_0(u),$$
   where L_k is the set of nodes in layer k that have already written a message on the whiteboard.
3. v is the node with the smallest ID that has not written a message on the whiteboard, the last message was written by a non-neighbor w,
   $$\sum_{u \in L_{l(w)}} d_{-1}(u) = \sum_{u \in L_{l(w)-1}} d_{+1}(u) - 2 \sum_{u \in L_{l(w)-1}} d_0(u),$$
   and
   $$\sum_{u \in L_{l(w)}} d_{+1}(u) - 2 \sum_{u \in L_{l(w)}} d_0(u) = 0.$$

Condition 2, by counting the edges crossing from layer $l(v) - 1$ to layer $l(v) - 2$, ensures that all the nodes in layer $l(v) - 1$ have sent their messages and the nodes of layer $l(v)$ may become active. Condition 3 ensures that, when the active connected component changes (because there are no edges "going outside" the last layer), exactly one node is activated. □

COROLLARY 3. *There exists a protocol in* FREEASYNC[log] *which, on any bipartite graph G, outputs a BFS-forest of G.*

PROOF. In a bipartite graph there are no edges between nodes in the same layer. In other words, we need to apply the protocol for the general case without computing $d_0(v)$. Thanks to this, all the information the nodes in layer k need to compute is available when layer k is activated. □

OPEN PROBLEM 2. *Is it possible to solve* SPANNING-TREE *or even* CONNECTIVITY *in the* FREEASYNC$[f(n)]$ *model? For which $f(n)$?*

OPEN PROBLEM 3. *Is it true that for all (or some) $f(n)$,* FREEASYNC$[f(n)] \subsetneq$ FREESYNC$[f(n)]$*? We conjecture that this is the case and that in fact* BFS *cannot be solved in the* FREEASYNC$[f(n)]$ *model for $f = o(n)$.*

5. ACKNOWLEDGMENTS

Partially supported by programs Fondap and Basal-CMM (I.R., K.S.), Fondecyt 1090156 (I.R.), Anillo ACT88 (K.S.), Fondecyt 11090390 (K.S) and FP7 STREP EULER (N.N.).

6. REFERENCES

[1] L. Babai, A. Gál, P. G. Kimmel, and S. V. Lokam. Communication complexity of simultaneous messages. *SIAM J. Comput.*, 33:137–166, 2004.

[2] F. Becker, M. Matamala, N. Nisse, I. Rapaport, K. Suchan, and I. Todinca. Adding a referee to an interconnection network: What can(not) be computed in one round. In *Parallel and Distributed Processing Symposium, International*, pages 508–514. IEEE Computer Society, 2011.

[3] A. K. Chandra, M. L. Furst, and R. J. Lipton. Multi-party protocols. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, STOC '83, pages 94–99. ACM, 1983.

[4] J. Feldman, S. Muthukrishnan, A. Sidiropoulos, C. Stein, and Z. Svitkina. On distributing symmetric streaming computations. *ACM Trans. Algorithms*, 6:66:1–66:19, 2010.

[5] S. Grumbach and Z. Wu. Logical locality entails frugal distributed computation over graphs. In *Proceedings of 35th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 5911 of *Lecture Notes in Computer Science*, pages 154–165, 2009.

[6] F. Kuhn, T. Moscibroda, and R. Wattenhofer. What cannot be computed locally! In *Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 300–309. ACM, 2004.

[7] N. Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

[8] D. Peleg. *Distributed computing: a locality-sensitive approach.* SIAM Monographs on Discrete Mathematics and Applications, 2000.