Distrib. Comput. (2015) 28:189–200
DOI 10.1007/s00446-014-0221-8

Allowing each node to communicate only once in a distributed system: shared whiteboard models

Florent Becker · Adrian Kosowski · Martin Matamala · Nicolas Nisse · Ivan Rapaport · Karol Suchan · Ioan Todinca

Received: 25 February 2013 / Accepted: 15 May 2014 / Published online: 4 June 2014

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 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.

Keywords Distributed computing · Local computation · Graph properties · Bounded communication

This paper is the union of two preliminary versions appeared in the proceedings of SPAA 2012 (Allowing each node to communicate only once in a distributed system: shared whiteboard models) and IPDPS 2011 (Adding a referee to an interconnection network: What can (not) be computed in one round?). It has been partially supported by programs Fondap and Basal-CMM (M.M., I.R., K.S.), Fondecyt 1100192 (M.M.), 1130061 (I.R.), FP7 STREP EULER (N.N.), ANR AGAPE (I.T.), ANR Displexity (A.K.), NCN under contract DEC-2011/02/A/ST6/00201 (A.K.), Ecos-Conicyt C09E04 (M.M., I.R., I.T.), Ecos-Sud Chili C12E03 (N.N, K.S.) and Associated Team Inria AlDyNet (N.N, K.S.).

F. Becker · I. Todinca
LIFO, Université d'Orléans, Orléans, France
e-mail: florent.becker@univ-orleans.fr

I. Todinca
e-mail: ioan.todinca@univ-orleans.fr

A. Kosowski
Inria Paris - LIAFA, Université Paris Diderot, Paris, France
e-mail: adrian.kosowski@inria.fr

M. Matamala (✉) · I. Rapaport
DIM-CMM (UMI 2807 CNRS), Universidad de Chile, Santiago, Chile
e-mail: mmatamal@dim.uchile.cl

I. Rapaport
e-mail: rapaport@dim.uchile.cl

N. Nisse
CNRS, I3S, Sophia-Antipolis, Inria & Univ. Nice Sophia-Antipolis, Nice, France
e-mail: nicolas.nisse@sophia.inria.fr

K. Suchan
Facultad de Ingeniería y Ciencias, Universidad Adolfo Ibáñez, Santiago, Chile
e-mail: karol.suchan@uai.cl

K. Suchan
Faculty of Applied Mathematics, AGH - University of Science and Technology, Kraków, Poland

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].

1.1 Communication using a shared whiteboard

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 G 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 order to communicate, each node will write some information on a shared memory called *whiteboard*. The questions we address are basic structural properties of the graph G, and one must be able to answer the question using only the contents of the whiteboard.

Clearly, if every node communicates its whole neighborhood (which can be done with $O(n)$ bits, where n is the number of nodes), the whole graph is described on the whiteboard; therefore, any question can be easily answered (assuming unlimited computational power). Nevertheless, our goal is to design communication protocols in which each node is only allowed to provide a small amount of information (e.g. $O(\log n)$ or $o(n)$) or to show that no such low communication protocol exists for the problem.

We start with a very simple model in which every node writes *simultaneously* one message (computed from its local knowledge) on the whiteboard. We show that there is a protocol using $O(\log n)$ bits per node, such that by reading the whiteboard, one gets 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. On the other hand, questions like "Does G contain a square?" or "Is the diameter of G at most 3?" cannot be solved by a protocol using $o(n)$ bits.

Then we investigate more powerful models. 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, 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 new message appears on the whiteboard.

Along the evolution of the system, nodes may be in one of three states: *awake*, *active* or *terminated*. Initially, all nodes are awake. A node becoming active means that this node computes a message which is stored in its local memory. Metaphorically speaking, it "rises its hand to speak" and writes the message it wants to share in its local memory. 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. This node enters the terminated state when it *sees* its message on the whiteboard.

In each round after the first one, several awake nodes may become active but exactly one active node becomes terminated. Note that a node can not become active and terminated in one round. As the round ends when a new message is written on the whiteboard, the node whose message was written at the end of round i will become terminated in round $i + 1$.

Depending on whether or not an active node can overwrite the value of its local memory, we differentiate synchronous models and asynchronous ones. In the former, once a node raises its hand it can "change its mind" later. That is, it can update the value stored in its local memory according to the current state of the system. On the other hand, in the asynchronous model, once a node raises its hand it cannot "change its mind" later. That is, the first message stored in the local

memory will be the one that eventually will be written in 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 the whiteboard. In particular, the order in which 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.

The last modification of the whiteboard leaves at most one node active. If the remaining nodes are all terminated, then this last active node computes the output of the system by using *only* the information stored on the whiteboard.

In this setting, we propose two more scenarios leading to the definition of four computational models. A computational model is said *simultaneous* if all nodes become active (raise their hands) after the first round. On the other hand, the model is said *free* if, in every round, any awake node may decide to become active.

1.2 Our results

We define four families of systems or protocols, SIMASYNC$[f(n)]$, SIMSYNC$[f(n)]$, ASYNC$[f(n)]$ and SYNC$[f(n)]$, which correspond to the four free/simultaneous, asynchronous/synchronous scenarios, parametrized by the amount $f(n)$ of data each node is allowed to write on the whiteboard. We show that the four classes of problems which can be solved by these models, PSIMASYNC$[f(n)]$, PSIMSYNC $[f(n)]$, PASYNC$[f(n)]$ and PSYNC$[f(n)]$, 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 PSIMASYNC$[f(n)] \subsetneq$ PSIMSYNC$[f(n)] \subsetneq$ PASYNC$[f(n)] \subseteq$ PSYNC$[f(n)]$; the strictness of the last inclusion is left as an open problem. On the other hand, we also prove that when $g(n) = o(f(n))$, PSIMASYNC$[f(n)] \not\subseteq$ PSYNC$[g(n)]$. 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 problems in distributed computing, and we examine their positions in our hierarchy. We show that BFS is in the class PSYNC$[\log n]$, and that for the bipartite case, it is in PASYNC$[\log n]$. We also show that for all $f(n) = o(n)$, BFS is not in the class PSIMSYNC$[f(n)]$, even in the bipartite case.

It is interesting to point out that these models of computations, quite natural for us, have never been studied before.

1.3 Related work

The two main aspects of our approach, the *locality* and the fact that nodes are allowed to send *only one short message* have been tackled before. In the classical model $\mathcal{CONGEST}$ [11], where a network is represented by a graph whose nodes correspond to network processors and links 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 [6]. In this model, where the total amount of information traversing each link is bounded by $O(\log n)$ bits, the authors showed that any first order logic formula can be evaluated in any planar or bounded degree network [6]. 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 [9,11]. In the $\mathcal{LOCAL}$ model, the restriction on the size of messages is removed so that every node is allowed to send unbounded size messages in every round. This model focuses on the issue of locality in distributed systems, and more precisely on the question "*What cannot be computed locally?*" [8]. Difficult problems like minimum node cover and minimum dominating set cannot be well approximated when processors can locally exchange arbitrary long messages during a bounded number of rounds [8].

It is worth to mention that a model where the number of rounds is limited to one and the message of each node is 0 or 1, has been considered in [5]. There a *complexity theory for local distributed algorithms* has been initiated. As in our model, the message of each node is based only on the information that it can collect among its neighbors. On the other hand, Naor and Stockmeyer in the context of *locally checkable labelings* study a model where each node computes its message based only on the information of nodes within a given radius from it [10]. They showed that a non trivial problem so-called *weak c-coloring* can be solved in this model but only in a restricted class of graphs and when the given radius is *large*. Since in our model each node must build its message considering only the information of its neighbors, some negative results obtained in [10] remain valid in it.

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 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 last referee, that collects and uses them in order to compute $f(x_1, \ldots, x_n)$. The Simultaneous 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 a graph $G = (V, E)$, each node $v \in V$ has a unique identifier $ID(v)$ between 1 and n. In what follows we shall assume that $V = \{v_1, \ldots, v_n\}$, where v_i is such that $ID(v_i) = i$. To ease the presentation we shall denote by $N(i)$ the set of neighbors of node v_i, for each $i = 1, \ldots, n$. 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. It also has a local memory. Each node is in one of three *states*: awake, active or terminated.

All nodes execute continuously the same algorithm.

Roughly, a node which is in the awake state must decide whether to become active. Once it is active, a node stores in its local memory a message that it wants to write on the whiteboard. Each node is allowed to write exactly one message on the whiteboard. Once a node writes the message it enters the terminated state. The size of these messages is some $f(n) = o(n)$, typically $O(\log n)$.

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, and, moreover, these messages do not change, even if they are not yet written on the whiteborad.

We also distinguish between simultaneous and free protocols. In simultaneous protocols, nodes must be ready to speak at any time, whereas in free protocol, they can decide when to become active. This gives raise to four families of models described in Table 1.

2.1 Synchronous protocols and subclasses

A synchronous protocol with message size M for graphs on n nodes is an algorithm which runs at each node of the graph by using a local memory and having access to a shared memory, which we call the *whiteboard*. At each round of the algorithm, each node is in one of the states of the set $S := \{\text{awake, active, terminated}\}$. At each round, each node v_i computes its new state and the new value of its local memory based on its current state, s_i, the state of its local memory, m_i, and the state of the whiteboard, denoted by W, according to two functions act and msg satisfying the following constraints.

1. If node v_i is not active it creates an empty message. More precisely, if $s_i \neq$ active, then $msg(v_i, N(i), W, s_i, m_i) = \epsilon$, where ϵ is the empty word.
2. If any message of node v_i appears on W, i.e., if node v_i *sees* one of its messages on the whiteboard, then $act(v_i, N(i), W, \text{active}) = \text{terminated}$.
3. $act(v_i, N(i), W, \text{awake}) \in \{\text{awake, active}\}$.

Table 1 Four families of protocols, where $f(n)$ represents the message size

	Message created when node becomes active	No restriction
All nodes active after the first round	SIMASYNC$[f(n)]$	SIMSYNC$[f(n)]$
No restriction	ASYNC$[f(n)]$	SYNC$[f(n)]$

A *configuration* of a synchronous protocol is the tuple (s, m, W) where s is the global state of the nodes, an element in S^n, m is the state of the local memories, a binary word of length at most nM, and W is the state of the whiteboard, a binary word of size at most nM.

A configuration $C' = (s', m', W')$ is a *valid successor* of a configuration $C = (s, m, W)$ for the synchronous protocol if for each node v_i, we have $s'_i = act(v_i, N(i), W, s_i)$, $m'_i = msg(v_i, N(i), W, s_i, m_i)$ is a binary word of length at most M, and W' has exactly one more message $m_j \neq \epsilon$, than W, where m_j is the message of some node v_j such that $s_j =$ active, and no message of node v_j appears on W. This definition can be interpreted as follows. An external entity, an *adversary*, adds to the whiteboard the message of the local memory of some active node. Notice that if a message m_i corresponding to node v_i is written on the whiteboard at the end of step t, then according to the function act, node v_i will be in state terminated at the end of step $t + 1$.

In the *initial configuration*, all nodes are awake, all local memories, as well as the whiteboard, are empty.

We say that a non-initial configuration is a *final* configuration if it has no active nodes. A final configuration is a *successful* configuration if all nodes are in the state terminated. Otherwise, it is a *corrupted* configuration.

An *execution* of a protocol is a (finite) sequence of configurations starting from the *initial configuration* and such that each configuration is a valid successor of the previous one. The execution is *successful* if the last configuration is successful. We say that the execution ends in a *deadlock* when the last configuration is corrupted.

The last ingredient of a synchronous protocol is its output function out. For a successful execution, where the last whiteboard configuration is W, we define the *output* of the protocol to be $out(W)$. This value is computed by the last node entering the terminated state. Recall that in this case, the state of the whiteboard W contains the messages from all the nodes.

Let SYNC[$f(n)$] be the set of all synchronous protocols with message size $O(f(n))$. We denote by SIMSYNC[$f(n)$] the subclass of SYNC[$f(n)$] protocols whose function act satisfies $act(v_i, N(i), \emptyset, \text{awake}) = \text{active}$, for each node v_i. It is clear that in this subclass of synchronous protocols there will never be deadlocks.

2.1.1 Asynchronous protocols

In asynchronous protocols nodes create their final 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. That is, in an asynchronous protocol each node modifies the value of its local memory just once. To implement this idea, we impose that, if $m_i \neq \epsilon$, then $msg(v_i, N(i), W, \text{active}, m_i) = m_i$. That is, the local memory may be modified only once.

Let ASYNC[$f(n)$] and SIMASYNC[$f(n)$] be the subsets of SYNC[$f(n)$] and SIMSYNC[$f(n)$] whose function msg satisfies previous restriction, respectively. Notice that for protocols in SIMASYNC[$f(n)$] all nodes are in the state active at the end of the second step and they have already computed their final messages in their local memory. Hence, we may assume that these protocols have just one step, where they compute $message_i := msg(v_i, N(i), \epsilon, \text{active}, \epsilon)$, for each $i = 1, \ldots, n$.

This paper aims at deciding what kind of problems can be solved in each of these models. Therefore, rather than classes of protocols we are interested in graph problems which can be solved by protocols. Notice that a protocol works on graphs of a given size n. However, in most cases the description of the protocol is parametric in n which implicitly defines a family of protocols indexed by n. When we say that a graph problem is solved by a protocol we means that for each size n a protocol working on graphs of size n solves the problem for all graphs of size n.

In the sequel we shall denote by PSIMASYNC[$f(n)$], PSIMSYNC[$f(n)$], PASYNC[$f(n)$], and PSYNC[$f(n)$] the class of problems which can be solved by the associated protocols using message size $O(f(n))$ on n-nodes graphs.

It is clear from the definitions that PSIMASYNC[$f(n)$] $\subseteq$ PSIMSYNC[$f(n)$] $\subseteq$ PSYNC[$f(n)$] and that PSIMASYNC[$f(n)$] $\subseteq$ PASYNC[$f(n)$] $\subseteq$ PSYNC[$f(n)$]. Less obvious, we shall prove in Section 5 that PSIMSYNC[$f(n)$] $\subseteq$ PASYNC[$f(n)$], and that three parts of this hierarchy are strict: PSIMASYNC[$f(n)$] $\subsetneq$ PSIMSYNC[$f(n)$] $\subsetneq$ PASYNC[$f(n)$].

3 Reconstruction of forests and bounded degeneracy graphs in SIMASYNC[log n]

Let BUILD be the problem that consists in computing the adjacency matrix of a graph.

We say that G is of degeneracy k if there is a node r of degree at most k in G that we can remove, and then proceed recursively on the resulting graph $G' = G \setminus r$, until we obtain an empty graph. Let us denote by r_i the i-th node removed from the graph.

Definition 1 $G = (V, E)$ is of degeneracy k if there exists a permutation $(r_1, \ldots, r_n)$ of V such that for all i, $1 \leq i \leq n$, r_i is of degree at most k in G_i, where G_i is the subgraph of G induced by $\{r_i, \ldots, r_n\}$.

In this section, we show that there is a SIMASYNC[log n] protocol solving BUILD on graphs of bounded degeneracy. Moreover, the protocol is robust in the sense that it detects if the graph is not in the class.

3.1 Case of forests ($k = 1$)

To give the flavor of the synchronous protocol for bounded degeneracy graphs, we start with the graphs with degeneracy 1, which is exactly the class of forests. Let T be a forest, and let $N_S(v)$ denote the neighborhood of $v \in V(S)$ in the sub-forest S of T.

In this case, each node v stores in its local memory a triple of integers. This triple consists of its identifier, its degree in T, $d_T(v)$, and the sum of the identifiers of its neighbors (this clearly can be encoded using less than $4 \log n$ bits):

$$m_v := \left(ID(v), d_T(v), \sum_{w \in N_T(v)} ID(w) \right).$$

Disregarding of the order used to fill the whiteboard, at a successful configuration the whiteboard has all messages $(m_v)_{v \in V}$. To build the graph using this information, the algorithm computing the output function chooses a leaf v (one of the nodes with degree at most 1). Intuitively, it prunes this leaf from the forest. By doing this recursively, it gets all the information concerning the forest. More precisely, the triple of v contains the identifier of the unique neighbor w of v since the sum of the IDs of the neighbors of v is exactly $ID(w)$. The output algorithm can replace the triple of w by

$$\left(ID(w), d_T(w) - 1, \sum_{z \in N_T(w)} ID(z) - ID(v) \right)$$

which is exactly

$$\left(ID(w), d_{T \setminus v}(w), \sum_{z \in N_{T \setminus v}(w)} ID(z) \right).$$

By induction on the number of nodes, the output algorithm is able to decode this information and rebuild the whole forest (or decide whether the graph contains a cycle).

In the following, we generalize the idea of "pruning" a node v (of degree at most k) from the graph G in such a way

that the information of the pruned graph $G \setminus v$ can be obtained from the information of all nodes of G (by modifying the information of the neighbors of v).

3.2 Generalization for any $k \geq 1$

Each node needs to know the value of k (recall that G is of degeneracy at most k). Moreover, some data structures that allow working on graphs of degeneracy upper bounded by k have to be present at all nodes. Nevertheless, no elimination order (see Definition 1) is known a priori: it will be discovered during the execution of calculations in the last node.

The information that each node v writes on the whiteboard is the following $k+2$-tuple:

- its identifier $ID(v)$.
- its degree $d_G(v)$ in G.
- for each integer $p, 1 \leq p \leq k$, the quantity $\sum_{w \in N_G(v)} (ID(w))^p$ (i.e., the sum of p's powers of the identifiers of the neighbors).

Note that for $k = 1$ this is the construction described in the case of forests. We shall see that, with this information, for any node v of degree at most k, the algorithm for the output function can retrieve the identifiers of the neighbors of v. Eventually, like in the case of forests, this output algorithm simulates the removal of node v from the graph and iterates the process until obtaining the empty graph.

To describe the encoding and decoding of the neighborhood information we need to recall some results from algebra and number theory. We will use the following matrix, very similar to the well-known Vandermonde matrix.

Definition 2 Define the matrix A by $A_{p,i} = i^p$, for $i = 1, \ldots, n$ and $p = 1, \ldots, k$. To express explicitly the dimensions we will write $A(k, n)$.

3.3 Neighborhood encoding

Let us recall that $(v_1, \ldots, v_n)$ denotes the nodes ordered by their identifiers. To encode its neighborhood, each node x uses the matrix $A(k, n)$ and the incidence vector of its neighborhood $\mathbf{x}$, i.e., the binary vector with 1 on the i-th coordinate if v_i is a neighbor of x, and 0 otherwise.

The message generated by node x is the tuple $(ID(x), d_G(x), \mathbf{b}(x))$, where $\mathbf{b}(x) = A(k, n)\mathbf{x}$.

Lemma 1 *The size of the message generated by each node is $O(\log n)$ bits – more precisely, $O(k^2 \log n)$ bits. The computation of the message can be performed in $O(n)$ local time.*

Proof For computing $\mathbf{b}$, the algorithm sums up at most n columns of the matrix $A(k, n)$. The result is a k element vector with a sum of some elements from the i-th row of $A(k, n)$ at position i. The coefficients in $A(k, n)$ are at most n^k, so the sum is at most n^{k+1}. It can be encoded using $(k+1)\log n$ bits, so the whole vector $\mathbf{b}$ requires $k(k+1)\log n$ bits at most. Altogether, the message associated to each node x is of size at most $O(k^2 \log n)$.

For the time complexity it is enough to notice that we sum up $O(n)$ values encoded using $O(\log n)$ bits each. □

3.4 Neighborhood decoding

We will use the following classical result from number theory.

Theorem 1 [14] *Let $j_1, j_2, \ldots, j_k$ be arbitrary positive integers. Let $i_1, i_2, \ldots, i_k$ be k unknowns. In integers, if the following system of simultaneous equations has non-trivial solutions, then $(i_1, i_2, \ldots, i_k) = (j_1, j_2, \ldots, j_k)$ up to a permutation.*

$$i_1^p + i_2^p + \cdots + i_k^p = j_1^p + j_2^p + \cdots + j_k^p \quad \text{for all } 1 \leq p \leq k$$

Notice that Theorem 1 covers also the case where some variables are 0, so we have the following corollary.

Corollary 1 *Given the message $(ID(x), d_G(x), \mathbf{b}(x))$ written on the whiteboard by a node of degree at most k, there exists only one binary vector $\mathbf{x}$ such that $A(k, n)\mathbf{x} = \mathbf{b}(x)$.*

In case a fast decoding of neighborhoods is needed, we can perform a preprocessing step to enumerate all k-subsets of $\{1, \ldots, n\}$ and compute the values $\mathbf{b} = A(k, n)\mathbf{x}$, where $\mathbf{x}$ is an incidence vector of such a subset, and store them in a table N that assigns to each value vector $\mathbf{b}$ the corresponding $\mathbf{x}$. The size of N is $O(n^k)$ and, by sorting it according to the lexicographic order on value vectors, we can perform a neighborhood look-up in time $k \log n$. Thus we have the following.

Lemma 2 *Let k, n be integers. There exists a function that, given the message $(ID(x), d_G(x), \mathbf{b}(x))$ of a node x of degree at most k, allows to compute the neighborhood of x in time $O(\log n)$.*

Using such a look-up table we can perform Algorithm 1, which reconstructs graph G in $O(n^2)$ time.

We conclude:

Theorem 2 *There exists a* SIMASYNC$[\log n]$ *protocol for* BUILD *for graphs of bounded degeneracy.*

Note that our protocol can also be turned into a recognition protocol for these graphs. By applying the same output algorithm as above, we just have to add one test in the output function, which rejects the graph if, during the pruning process, we find no node of degree at most k.

Many graph classes are known to be of bounded degeneracy. Planar graphs are of degeneracy at most 5, graphs of

Algorithm 1 Output function

Require: the whiteboard $W = \{(ID(x), d_G(x), \mathbf{b}(x)) \mid x \in V\}$, the look-up table N (as in Lemma 2)
Ensure: $H = (V, E)$ – a reconstruction of G
  while there is a message on the whiteboard W **do**
    take an element $(ID(x), d_G(x), \mathbf{b}(x))$ from W s.t. $d_G(x) \leq k$
    look-up in N the neighborhood $\mathbf{x}$ of x
    for all $v_i \in V$ s.t. $\mathbf{x}(i) = 1$ **do**
      add $\{x, v_i\}$ to H
      update $(ID(v_i), d_G(v_i), \mathbf{b}(v_i))$ in W according to the removal of x from G
    end for
    remove $(ID(x), d_G(x), \mathbf{b}(x))$ from W
  end while
  return H

treewidth k are also of degeneracy at most k, and more generally, for each fixed graph H, the class of H-minor free graphs is also of bounded degeneracy [7,12,13]. It is worth to mention that with our tools we can deal with graphs having a node ordering where each node v has degree at most k or at least $n - k - 1$, in the graph induced by nodes appearing later than v in the ordering.

4 Hard problems for SIMASYNC

We give in this section examples of problems that cannot be solved in SIMASYNC$[f(n)]$, for any function $f = o(n)$. The proof technique is based on a reduction from BUILD.

Recall that BUILD is the problem that consists in computing the adjacency matrix of a graph.

Lemma 3 *Let $\mathcal{G}$ be a family of n-node graphs, and $g(n)$ be the number of graphs in $\mathcal{G}$. Let $f(n)$ be any function, and $\mathcal{C} \in \{$SIMASYNC, SIMSYNC, ASYNC, SYNC$\}$. If* BUILD*, when the input graphs are restricted to the class $\mathcal{G}$, can be solved in the model $\mathcal{C}[f(n)]$ then $\log g(n) = O(nf(n))$.*

Proof Consider any protocol 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 the whiteboard W. Hence, at the end, a total of $O(nf(n))$ bits are available on the whiteboard. For the output function to distinguish two different graphs in $\mathcal{G}$, we must have $\log g(n) = O(nf(n))$. □

Consider the problem TRIANGLE, which consists in deciding whether an arbitrary graph G contains a triangle as subgraph (i.e. three pairwise adjacent nodes).

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

Proof For the sake of contradiction, let us assume that there is a SIMASYNC$[f(n)]$ protocol $\mathcal{A}$ for detecting triangles. We shall prove that by using $\mathcal{A}$ we can build a SIMASYNC$[f(n+1) + \log n]$ protocol $\mathcal{A}'$ for reconstructing bipartite graphs with parts $\{v_1, \ldots, v_{n/2}\}$ and $\{v_{n/2+1}, \ldots, v_n\}$. Since there are $\Omega(2^{(n/2)^2})$ such bipartite graphs we would get a contradiction with Lemma 3.

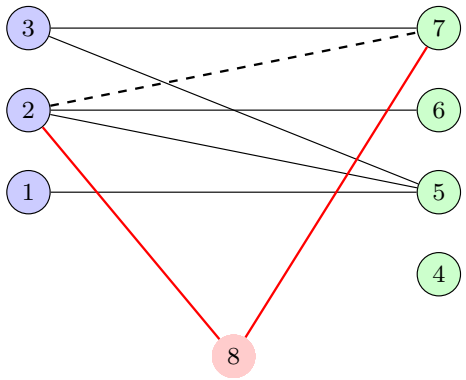

Fig. 1 Reducing reconstruction of arbitrary graph to detection of triangles: given the graph G (*circled nodes*), in order to check whether $(2, 7)$ is an edge of G, we build the auxiliary graph $G'_{2,7}$ by adding node 8 as depicted on the figure. It contains a triangle if and only if $(2, 7)$ is an edge of G

Informally, for each $s, t \in \{1, \ldots, n\}$, we shall simulate the behavior of $\mathcal{A}$ on the graph $G'_{s,t}$ obtained from a graph $G = (V, E)$ by adding an extra node v_{n+1}, and edges $\{v_s, v_{n+1}\}$ and $\{v_t, v_{n+1}\}$. If G is bipartite, then $G'_{s,t}$ has a triangle if and only if $\{v_s, v_t\} \in E$ (see Fig. 1). Therefore, since for each s, t, protocol $\mathcal{A}$ can decide whether $G'_{s,t}$ has a triangle, it will be possible to reconstruct G.

On an input graph $G = (V, E)$, the message function of protocol $\mathcal{A}'$ is as follows. Let m'_i be the message produced by protocol $\mathcal{A}$ on a node with ID i and neighborhood $N(i)$ (where $N(i)$ is the neighborhood of v_i in G) and let m''_i be the message produced by $\mathcal{A}$ on a node with ID i and neighborhood $N(i) \cup \{v_{n+1}\}$. Protocol $\mathcal{A}'$ will produce, for node v_i, the message containing the triple (i, m'_i, m''_i). Thus, it uses $2 \cdot f(n+1) + \log n$ bits.

Let us describe the output function of $\mathcal{A}'$. For each s, t, and for $i \in \{1, \ldots, n+1\}$, let $m_i(s, t)$ be defined as m'_i when $i \notin \{s, t\}$, as m''_i when $i \in \{s, t\}$, and as the message produced by $\mathcal{A}$ on node v_{n+1} in $G'_{s,t}$, when $i = n + 1$. Clearly, this information can be retrieved from the whiteboard of $\mathcal{A}'$. Moreover, it corresponds exactly to the whiteboard produced by protocol $\mathcal{A}$ for $G'_{s,t}$. Therefore, using the output function of $\mathcal{A}$, we detect whether $\{v_s, v_t\}$ is an edge of G.

By doing this on each s and t, the output function of protocol $\mathcal{A}'$ builds the whole graph G. □

It is worth mentioning that in Theorem 3, we have provided a reduction from SIMASYNC$[f(n)]$ protocols for BUILD to a SIMASYNC$[f(n) + \log n]$ protocol for TRIANGLE on bipartite graphs with a given partition. Moreover, the local time

complexities of the new protocol are polynomially bounded in terms of the original protocol.

In the sequel, we shall use this type of reductions to prove that several problems are hard for some of our models. Also, we have proved in [2] that computing the diameter cannot be performed in SIMASYNC$[f(n)]$ for any $f = o(n)$; the construction is quite similar to the one of Theorem 3.

5 A computing power lattice

In this section we intend to show that our four models form a lattice in which the computational power grows strictly whenever either the synchronization 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 4 *For all* $\Omega(\log n) = f(n) = o(n)$, PSIMASYNC$[f(n)] \subsetneq$ PSIMSYNC$[f(n)] \subsetneq$ PASYNC$[f(n)]$ *and* PASYNC$[f(n)] \subseteq$ PSYNC$[f(n)]$.

We start with the following weaker result:

Lemma 4 *For all* $f(n) = o(n)$, PSIMASYNC$[f(n)] \subseteq$ PSIMSYNC$[f(n)] \subseteq$ PASYNC$[f(n)] \subseteq$ PSYNC$[f(n)]$.

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

Proof **PSIMASYNC**$[f(n)] \subseteq$ **PSIMSYNC**$[f(n)]$. 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.

PSIMSYNC $[f(n)] \subseteq$ **PASYNC**$[f(n)]$. 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 ASYNC one if we fix an order (for instance $v_1, \ldots, v_n$) and use this order for a sequential activation of the nodes.

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

5.1 SIMASYNC vs. SIMSYNC

We consider here a "rooted" version of the INCLUSION MAXIMAL INDEPENDENT SET problem. This problem, denoted by MIS, 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 5 MIS *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, a protocol 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 nodes with their IDs on the whiteboard consists of an inclusion maximal independent set containing x. □

Theorem 6 *For any* $f(n) = o(n)$, MIS *cannot be solved in the* SIMASYNC$[f(n)]$ *model.*

Proof Let $f(n) = o(n)$. We proceed by contradiction, following the same ideas as in the proof of Theorem 3. Let us assume that there exists a protocol $\mathcal{A}$ for solving MIS in the SIMASYNC$[f(n)]$ model. Then we show how to design a protocol $\mathcal{A}'$ to solve BUILD for any graph in this model, contradicting Lemma 3.

Let $G = (V, E)$ be any 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 node x adjacent to every node 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 $\{v_i, v_j\} \notin E$. Moreover, 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. We denote by $\mathcal{A}(v_k, G^{(x)}_{i,j})$ the message created by node v_k following protocol $\mathcal{A}$ when the input graph is $G^{(x)}_{i,j}$.

Notice that, for a given k, the node v_k can generate only two possible messages depending on whether $k \in \{i, j\}$ or $k \notin \{i, j\}$. That is, only two messages are generated by each node for all 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 BUILD 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 the decision function of $\mathcal{A}$ in $G^{(x)}_{s,t}$ by using messages m_s, m_t and $\{m'_k : k \in \{1, \cdots, n\}\backslash\{s, t\}\}$. Since the output of $\mathcal{A}$ is $\{x, v_s, v_t\}$ if and only if $\{v_s, v_t\} \notin E$, the result follows. This would mean that from $O(nf(n))$ bits we can solve BUILD in the class of all graphs, a contradiction. □

Corollary 2 *For all* $\Omega(\log n) = f(n) = o(n)$, PSIMASYNC$[f(n)] \subsetneq$ PSIMSYNC$[f(n)]$.

Actually, the same result holds for TRIANGLE mentioned above.

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 nodes .

It is easy to show that 2- CLIQUES can be solved in the SIMSYNC$[\log n]$ model. Indeed, a trivial protocol can partition the nodes into two cliques numbered 0 and 1 if the input consists of two cliques, or otherwise indicates that it is not the case. The first node u to be chosen by the adversary writes $(ID(u), 0)$ on the whiteboard W. Then, each time a node 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 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 2- CLIQUES cannot be solved in any 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?*

5.2 SIMSYNC vs. ASYNC

We say that a graph is *even-odd-bipartite* if there are no edges between nodes having identifiers with the same parity. For the purpose of separating classes PSIMSYNC and PASYNC, 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 7 EOB-BFS *can be solved in the* ASYNC$[\log n]$ *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 protocol 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 on 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 $(ID(v), l(v), p(v), d_{-1}(v), d_{+1}(v))$ where:

$ID(v)$ 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
$d_{-1}(v) = |N^*_v|$
$d_{+1}(v) = |N(v)| - |N^*_v|$,

where $l(v)$ represents the layer of v, $d_{-1}(v)$ its degree towards the previous layer, $d_{+1}(v)$ its degree towards the next layer and $p(v)$ its parent in the BFS-forest. The message created by v_1 at the beginning is $(1, 0, \text{ROOT}, 0, |N(v_1)|$. 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 the nodes of the same layer to become active simultaneously, the protocol later on works as follows: an arbitrary node v becomes active (and computes its message) if the following two conditions are satisfied:

1. A neighbor w of v has already written its message on the whiteboard, and
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 nodes in layer k have written their messages before layer $k+1$ is activated. To ensure this property, the second condition gives a certificate that all edges from layer $k-1$ to layer k have been acknowledged by nodes in layer k.

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 8 *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 BUILD for even-odd-bipartite graphs in SIMSYNC$[f(n)]$, in contradiction with Lemma 3. Note that there are $2^{\Omega(n^2)}$ even-odd-bipartite graphs with n nodes.

Let $G = (V, E)$ be an even-odd-bipartite graph with $V = \{v_2, \ldots, v_n\}$. Assume that n is odd.

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

Suppose now that we run $\mathcal{A}$ on G_i. It follows that a node v_j is at layer 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 all G_i (i.e., for all $3 \le i \le n$ odd), 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 neighborhood 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.

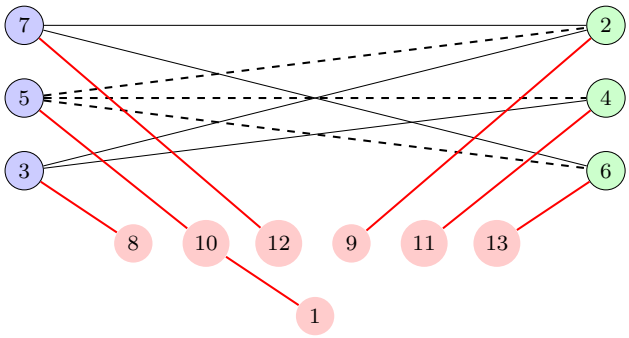

Fig. 2 Reducing reconstruction of even-odd-bipartite graph to construction of BFS tree in even-odd-bipartite graphs: given the even-odd-bipartite graph G (with node set $\{2, \ldots, 7\}$), in order to check the edges of G incident to 5, we build the auxiliary graph G_5 by adding nodes $\{1, 8, \ldots, 13\}$ as depicted on the figure. Node $j \in \{2, 4, 6\}$ is in the third layer of a BFS-tree rooted in 1 if and only if $(5, j)$ is an edge of G

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 neighborhood of v_i. Thus, EOB-BFS is not in PSIMSYNC$[f(n)]$. □

Corollary 3 *For all $\Omega(\log n) = f(n) = o(n)$,* PSIMSYNC $[f(n)] \subsetneq$ PASYNC$[f(n)]$.

5.3 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 $\in$ PSIMSYNC$[\log n]$ but MIS $\notin$ PSIMASYNC$[o(n)]$. In other words, there are problems that can not be solved if we go down in the synchronization hierarchy *no matter the extra length given to the size of the messages*. Now we are going to prove a more general result.

Theorem 9 *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 keeping only edges between nodes in $\{v_1, \ldots, v_{f(n)}\} \subseteq V$. Let $g(n) = o(f(n))$. We have that* SUBGRAPH$_f \in$ PSIMASYNC$[f(n)]$ *but* SUBGRAPH$_f \notin$ PSYNC$[g(n)]$.

Proof It is obvious that SUBGRAPH$_f$ is in PSIMASYNC$[f(n)]$: 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 PSYNC$[g(n)]$, 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 3 because these graphs need $O(n \log n + (f(n))^2)$ bits to be defined. □

6 Connectivity and related problems

One of the main questions 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 Sect. 5 we saw that it is possible in the ASYNC[$\log n$] model 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 synchronization, 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 10 BFS *can be solved in the* SYNC[$\log n$] *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 layer (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 $(ID(v), l(v), p(v), d_{-1}(v), d_0(v), d_{+1}(v))$ where:

$ID(v)$ 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

$d_{-1}(v) = |\{w \in N_v^* : l(w) = l(v) - 1\}|$

$d_0(v) = |\{w \in N_v^* : l(w) = l(v)\}|$

$d_{+1}(v) = |N(v)| - d_{-1}(v)$.

Consider nodes v at distance at least 2 from the ROOT. These nodes v become active if either the conditions (a) and (b) are satisfied, or the condition (c) is satisfied, where

(a) A neighbor w of v has already written its message on the whiteboard.

(b) $$\sum_{u \in L_{l(w)}} d_{-1}(u) = \sum_{u \in L_{l(w)-1}} d_{+1}(u) - 2 \sum_{u \in L_{l(w)-1}} d_0(u),$$

where L_k is the set of nodes in layer k that have already written a message on the whiteboard.

Table 2 Classification of communication models

	SIMASYNC	SIMSYNC	ASYNC	SYNC
BUILD k-degenerate	Yes	Yes	Yes	Yes
Rooted MIS	No	Yes	Yes	Yes
TRIANGLE	No	Yes	Yes	Yes
EOB- BFS	No	No	Yes	Yes
BFS	?	?	?	Yes

(c) 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 (b), 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 (c) ensures that, when the active connected component changes (because there are no edges *going outside* the last layer), exactly one node is activated. □

Corollary 4 *There exists a protocol in* ASYNC[$\log n$] *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. □

7 Conclusion

We have introduced four communication models, combining locality and congestion, in which communication is provided through a whiteboard shared by all nodes. Table 2 resumes the status of several problems like the reconstruction of bounded degeneracy graphs, "rooted" maximal independent set, triangle detection, even-odd-bipartite BFS and BFS in these models. Each cell marked *yes* indicates that the problem can be solved in that model, using messages of size $O(\log n)$. Each cell marked *no* establishes that there is no protocol for that problem, using $o(n)$ bits. In particular, for any $f(n) = o(n)$ we have that PSIMASYNC[$f(n)$] $\subsetneq$ PSIMSYNC[$f(n)$] $\subsetneq$ PASYNC[$f(n)$] $\subseteq$ PSYNC[$f(n)$].

Let us emphasize several questions that remain open:

Open problem 2 *Is it possible to solve* SPANNING- TREE *or even* CONNECTIVITY *in the* ASYNC$[f(n)]$ *model? For which* $f(n)$*?*

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

Another natural direction for further research would be to investigate *randomized* protocol for these models. Recall that 2- CLIQUES problem consists in deciding whether G is the disjoint union of two complete graphs with n nodes. It can be shown that 2- CLIQUES admits a randomized protocol for these models. Hence, a natural question is

Open problem 4 *Which problems are solvable by a randomized protocol in* SIMASYNC$[\log n]$*?*

To conclude, we discuss a strong hypothesis of our model, namely, nodes have distinct IDs in $\{1, \cdots, n\}$. It would be interesting to know whether our positive results still hold if nodes are only constraint to have distinct IDs from some linearly or polynomially bounded domain. Our algorithm to recognize and build graphs with bounded degeneracy (Theorem 2) does not rely on the IDs' domain. On the other hand, our algorithms to compute the MIS (Theorem 5) and a BFS-tree in Even-odd-bipartite graphs (Theorem 7) rely only on the knowledge of the smallest ID. Finally, our algorithm to compute BFS-trees in general graphs in the SYNC model deeply relies on the knowledge of the IDs' domain (in the activation condition (c) that allows to avoid deadlocks).

References

1. Babai, L., Gál, A., Kimmel, P.G., Lokam, S.V.: Communication complexity of simultaneous messages. SIAM J. Comput. **33**, 137–166 (2004)
2. Becker, F., Matamala, M., Nisse, N., Rapaport, I., Suchan, K., Todinca, I.: Adding a referee to an interconnection network: What can(not) be computed in one round. In: Parallel and Distributed Processing Symposium, International, pp. 508–514. IEEE Computer Society (2011)
3. Chandra, A.K., Furst, M.L., Lipton, R.J.: Multi-party protocols. In: Proceedings of the 15th Annual ACM Symposium on Theory of Computing, STOC '83, pp. 94–99. ACM (1983)
4. Feldman, J., Muthukrishnan, S.M., Sidiropoulos, A., Stein, C., Svitkina, Z.: On distributing symmetric streaming computations. ACM Trans. Algorithms **6**, 66:1–66:19 (2010)
5. Fraigniaud, P., Korman, A., Peleg, D.: Towards a complexity theory for local distributed computing. J. ACM **60**(5), 35 (2013)
6. Grumbach, S., Wu, Z.: Logical locality entails frugal distributed computation over graphs. In: Proceedings of 35th International Workshop on Graph-Theoretic Concepts in Computer Science (WG), Lecture Notes in Computer Science, 5911, pp. 154–165 (2009)
7. Kostochka, A.V.: Lower bound of the Hadwiger number of graphs by their average degree. Combinatorica **4**(4), 307–316 (1984)
8. Kuhn, F., Moscibroda, T., Wattenhofer, R.: What cannot be computed locally! In: Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 300–309. ACM (2004)
9. Linial, N.: Locality in distributed graph algorithms. SIAM J. Comput. **21**(1), 193–201 (1992)
10. Naor, M., Stockmeyer, L.: What can be computed locally? SIAM J. Comput. **24**(6), 1259–1277 (1995)
11. Peleg, D.: Distributed computing: a locality-sensitive approach. SIAM Monographs on Discrete Mathematics and Applications (2000)
12. Thomason, A.: An extremal function for contractions of graphs. Math. Proc. Cambridge Philos. Soc. **95**, 261–265 (1984)
13. Thomason, A.: The extremal function for complete minors. J. Comb. Theory. Ser. B **81**(2), 318–338 (2001)
14. Wright, E.: Equal sums of like powers. Bull. Amer. Math. Soc. **8**, 755–757 (1948)