Allowing Multiple Rounds in the Shared Whiteboard Model: Some More (Im)possibility Results

Dharmeet Singh Hora, Piyush Bansal, Kishore Kothapalli and Kannan Srinathan
International Institute of Information Technology, Hyderabad, India
Email: {dharmeet.singh, piyush_bansal}@research.iiit.ac.in, {kkishore, srinathan}@iiit.ac.in

***Abstract*—The shared whiteboard model for distributed computing is one of the recent interesting models to be proposed (See Becker et al. (SPAA 2012)). In its basic form, this model allows all nodes to write a message of no more than $O(\log n)$ bits on a whiteboard that every node can read. However, each node can write at most once. In this model, a variety of problems from graphs are shown to be either possible or impossible.**

In this paper we extend the work of Becker et al. to allow for nodes to write on the shared whiteboard more than once. However, each node can write at most $O(\log n)$ bits at any one instant. Interestingly, in this model, we show that allowing just two rounds of writing on the whiteboard, one can color the vertices of a d-degenerate graph using $d+1$ colors. Similarly, we show that two rounds suffice to find maximal independent set (MIS), whereas 2-ruling sets can be computed in one round in *simultaneous synchronous* models. Finally, we show that for finding connected components in a graph, even $O(1)$ rounds is not enough in general. We show that any deterministic algorithm that follows certain rules requires at least $\Omega(\log n/\log\log n)$ rounds to find the connected components of an n-vertex graph. At the same time, we show the existence of a $O(\log n)$ round algorithm for the same. Thus, our results indicate that the multi-round shared whiteboard model has interesting consequences.

I. Introduction

An interconnected distributed system is typically represented by a graph, where nodes represents the processors and edges corresponds to the communication links between processors. In general the nodes lack the global knowledge about the network, and the communication takes place only through inter-processor message passing, hence new models with new algorithmic techniques and complexity notions arise. In the classical distributed models, the communication is much slower and costly than local computation, hence the complexity analysis of distributed algorithms mainly focuses on message passing where the size of the messages sent by each node or the number of communication rounds needed to perform some computation becomes important performance measure.

One of the recent developments in models of distributed computing is the shared whiteboard model of Becker et al. [3]. Becker et al. [3] model the distributed system is a graph in which each node knows only about its neighborhood, and the communication takes place through a whiteboard. Each node can write a message of fixed length on the whiteboard and read the contents of the whiteboard at any given time. The computation each node needs to perform is related to some property of graph. The shared whiteboard model falls between the classical *CONGEST* [11] and the *LOCAL* model introduced by Linial [8]. In the *CONGEST* model, each node can send a message of $O(\log n)$ bits through each of its outgoing links, whereas in the *LOCAL* model there is no restriction on the size of the message sent by each node.

In the shared whiteboard model, Becker et al. [3], have identified four variants *free-sync*, *SimultaneousSync*, *Free-Async* annd the *SimAsync*. By now, several possibilities and impossibilities are known in these models when each node is allowed to write on the whiteboard at most once. For instance, finding an inclusive maximal independent set MIS(v), that is a maximal independent set with node v in the MIS, is possible in the SimSync model and is impossible in the SimAsync model. Similarly finding a breadth-first ordering of the nodes of an even-odd bipartite graph is possible in FreeAsync model but is impossible in SimSync model. However, such a breadth-first ordering of the nodes of a graphs is possible in the FreeSync model.

However, it is not clear if the shared whiteboard model should naturally be limited to one round. Hence, a natural extension is to study the power of the above models when each node is allowed to write on the whiteboard more than once. The parameters of the whiteboard model therefore are the number of rounds that nodes can write to the whiteboard, the message size during each round and the maximum message size written by any node over all the rounds. Once each node can write more than once on the whiteboard, which all nodes can see at all times, it is conceivable that several fundamental problems on graphs that are impossible to solve with one round may now be solvable. For instance, we show that the MIS problem can be solved in two rounds in the SimSync model.

Extending the model of Becker et al. [3], our work therefore tries to answer the following questions. Is the one round shared whiteboard model strictly less powerful than a 2-round shared whiteboard model? What problems are possible and impossible in the multi-round shared whiteboard model. Is the relative hierarchy of the four different shared whiteboard models as identified in [3] maintained over multiple rounds? Answers to these questions can help one understand the true potential of the shared whiteboard model. We extend the shared whiteboard model to a multi-round model. In this model, we show the following results.

- We show that in a 2 round shared whiteboard model one can color planar graphs using 6 colors and graphs with bounded degeneracy d using $d+1$ colors. These algorithms operate in the FreeAsync model, and each node is required to write $O(\log n)$ bits on the shared whiteboard in each round. This result shows that the 2 round model is strictly more powerful than the one round model of [3].

- We show that a 2-ruling set of a graph can be computed in one round in the SimSync model, and computing an MIS in the same model can be done in two rounds.
- We show a nontrivial lower bound of $O(\log n/\log\log n)$ rounds for finding the connected components of a graph in the SimSync model. This lower bound applies to several standard deterministic algorithms. We further show that the popular parallel algorithm of Shiloach and Vishkin [12] can be adapted to run in the SimSync and the SimAsync models, indicating an $O(\log n)$ upper bound on the number of rounds for finding the connected components of a graph. Surprisingly, we show that in the FreeAsync model, one can solve this problem in one round.

A. Related Work

The shared whiteboard model of Becker et al. [3] follows another similar model by Becket et al. [4]. The model of [4] is called as a *referee* model where there is a referee who can see all the messages written on the whiteboard, perform a computation, and can deduce certain properties of the graph. Nodes however are restricted to write at most $O(\log n)$ bits on the whiteboard.

For coloring planar graphs, there exist parallel and distributed algorithms that use $O(1)$ colors and run in $O(\log n)$ rounds [2]. It was also shown that one cannot color a planar graph in less than $O(2^{\sqrt{\log n}})$ rounds unless $O(\sqrt{\log n})$ colors or more are used [1]. However, it seems that the ability to see the decisions of all the nodes on a whiteboard can help in reducing the number of rounds to two.

Becker [3] show that MIS(v) can be solved in one round in SimSync model, where MIS(v) stands for an inclusive maximal independent set, that is an MIS which contains a given node v. It is not known whether the MIS problem can be solved in round in the above model. We show that two rounds suffice by first computing a 2-ruling set in the first round, and extending the 2-ruling set to an MIS in the second round.

Finding the connected components of a graph has been studied in various settings. Shiloach and Vishkin [12] and Chong et al. [5] show that $O(\log n)$ parallel time suffices to solve the problem in the CRCW and the EREW PRAM model respectively. Becker et al. [3] show that the problem can be solved in one round in FreeSync i.e. free synchronous model in shared whiteboard setting.

II. A Summary of the Shared Whiteboard Model

In this model a whiteboard is shared with all the nodes in the interconnection network, where each node can write a message m after performing its local computation based on its ID, its local knowledge and the contents of the whiteboard. The interconnected network is modeled as a simple undirected connected n-node graph $G = (V, E)$. Each node $v \in V$ has a unique identifier $ID(v)$ between 1 and n. Typically, $V = \{v_1, v_2, ..., v_n\}$, where v_i is such that $ID(v_i) = i$.

At any point of time a node v can be in any one of three states $\{$*awake, active, terminate*$\}$, where *awake* stands for the nodes which have not written their message on whiteboard and are not ready to write, *active* means that the nodes are ready to write on whiteboard and if there are more than one active node, then the adversary chooses the order of nodes to write on the whiteboard. When the nodes have written their message m on the whiteboard they go to the *terminate* state.

This model can be further classified into four ways, depending on whether all the nodes are active, called the *simultaneous* models, or the nodes become active according to some activation function, called the *free* models. Another classification is based on whether the nodes generate their message as soon as they become active, called *asynchronous* model, or whether they generate their message when they are chosen to write on the whiteboard, called the *synchronous* models. We now formalize the synchronous and the asynchronous models.

A. Synchronous Protocol

In a synchronous protocol, each node generates its message depending on its ID, the local knowledge and the contents on the whiteboard, and the node becomes active according to a activation function (for Free model) which decides based on the ID of the node, its local knowledge and the contents of the whiteboard. After end of each round, all the nodes will go to *awake* state at the start of next round, and the contents on the whiteboard of the prior rounds will be considered in activation and message functions. So formally:

- $act_n : [1,n] \times 2^{[1,n]} \times W_{n,f(n),r(n)} \rightarrow \{awake, active\}$ where W represents the whiteboard, $f(n)$ stands for the message size and $r(n)$ represents the number of previous rounds.
- msg_n: $[1,n] \times 2^{[1,n]} \times W_{n,f(n),r(n)} \rightarrow \{0,1\}^{f(n)}$

B. Asynchronous Protocol

Nodes generates their messages as soon as they become active in the asynchronous protocol, so if two nodes becomes active simultaneously, then the message written by first node does not affect the message of second node. Hence we can combine the activation function and message generation function, and write them formally as:

- act/msg_n: $[1,n] \times 2^{[1,n]} \times W_{n,f(n),r(n)} \rightarrow \{awake, active\} \times \{0,1\}^{f(n)}$. The message is created only when the node becomes active.

Depending on whether all nodes are initially active or awake, and the message creation protocol used we have 4 models: *free asynchronous* FreeAsync, *simultaneous asynchronous* SimAsync, *free synchronous* FreeSync and *simultaneous synchronous* SimSync. Table **??** summarizes the above discussion.

As shown in [3], the above models follow the following hierarchy:

$$\text{SimAsync} < \text{SimSync} < \text{FreeAsync}.$$

The FreeSync is the most powerful of all models, but whether FreeAsync is weaker or equivalent to FreeSync is left as an open problem.

III. Graph Coloring

In this section we will solve the problem of graph coloring for planar graphs and then extend our arguments to graphs with bounded degeneracy. We will show that in the FreeAsync model, we can color the vertices of a planar graph using 6 colors, and color the vertices of a graph with degeneracy d using $d+1$ colors. Both our algorithms need nodes to write $O(\log n)$ bits on the whiteboard over two rounds.

A. Vertex Coloring a Planar Graph with 6 Colors

The basic idea of our algorithm is to mimic the recursive algorithm for six coloring the vertices of a planar graph. This is possible due to the fact that every planar graph has a vertex of degree at most 5. Hence, once such a vertex v is identified, we can color the rest of the graph recursively using 6 colors. In the end, we can then extend the coloring to vertex v easily as the neighborhood of v can have at most 5 distinct colors.

We will use the above idea to color the vertices of a planar graph in two rounds. In the first round, a node becomes *active* if its remaining degree is at most 5. Each such node will write its ID on the whiteboard over the first round. In the second round each node will become active in the reverse order of the order in which they write their ID's on the whiteboard. Each node will now compute its color, using the information available on the whiteboard and then writes its choice along with its ID on the whiteboard. Algorithm 1 describes our algorithm whose correctness is shown by the following theorem.

Algorithm 1: 6-Coloring Planar Graphs

```
   Round 1:
1  while deg(v) > 5 OR state(v) = awake do
2      if deg(v) ≤ 5 then
3          state(v) = active;
4          if node(v) chosen by adversary to write then
5              write ID(v) on whiteboard;
6              Nbrs(v) update their remaining degree;
           end
       end
   end
   Round 2:
7  while state(v) = awake do
8      Of the remaining nodes, node v which wrote last in
       the first round becomes active;
9      Node v writes its ID, and picks the recently used
       color c if no neighbors used it, else pick a new
       color;
   end
```

Theorem 1. *In the FreeAsync shared whiteboard model, in two rounds, the vertices of a planar graph can be colored with 6 colors. Further, each node writes $O(\log n)$ bits in each of the two rounds.*

Proof: In Algorithm 1 each node with at most degree 5 will become active and write its ID in the order chosen by the adversary. Notice that by the time a node v gets its turn to write on the whiteboard in the second round, it holds that at most five colors would be used up in its colored neighborhood. So, node v can color itself in second round and write its choice on the whiteboard. In essence, the protocol is playing the recursive algorithm by cleverly using the whiteboard.

Notice further that in the second round, only one node is active at any moment and the message this active node writes in the second round will be dependent on contents on whiteboard and the local knowledge of v.

Since the ID of a node and its color choice are both in $O(\log n)$ bits, the claim on the message complexity holds. ■

B. Coloring graphs with bounded degeneracy

Algorithm 1 which is coloring the planar graph using 6 colors, can be extended to Algorithm 2 which is algorithm to color a graph G with bounded degeneracy d using $d+1$ colors in 2 rounds. The basic idea of the algorithm is to adapt the recursive algorithm for $d+1$ coloring a d-degeneracy graph. The recursive algorithm identifies a vertex v of degree at most d, removes it from the graph, and colors the rest of the graph with $d+1$ colors. This coloring is then extended to vertex v noting that the neighborhood of v has at most d colors used up. We now present Algorithm 2 that achieves the required coloring in the FreeAsync model.

Algorithm 2: d+1 Coloring

```
   Round 1:
1  while deg(v) > d OR state(v) = awake do
2      if deg(v) ≤ d then
3          state(v) = active;
4          if node(v) chosen by adversary to write then
5              write ID(v) on whiteboard;
6              Nbrs(v) update their remaining degree;
           end
       end
   end
   Round 2:
7  while state(v) = awake do
8      Of the remaining nodes, node v which wrote last in
       the first round becomes active;
9      Node v writes its ID, and picks the recently used
       color c if no neighbors used it, else pick a new
       color;
   end
```

Theorem 2. *In the FreeAsync shared whiteboard model, in two rounds, the vertices of a graph of degeneracy d can be colored with $d+1$ colors. Further, each node writes $O(\log n)$ bits in each of the two rounds.*

C. MIS and 2-Ruling Sets

In this section, we focus on the problems of computing a maximal independent set in a graph and computing a 2-ruling set in a graph. It is shown in [3] that a relaxed notion of an MIS, called the inclusive-MIS, where one given node v is known to be the part of the MIS, can be computed in one round in the SimSync model. We show that while finding an MIS in the SimSync model is not possible in one round, a 2-ruling set can be computed in one round in the same model. Further, a 2-ruling set can be extended to an MIS with one more additional round.

D. One round 2-ruling set and 2 round MIS

MIS is impossible in one round in the SimSync model, but 2-ruling set is possible in the same setting in one round. The basic idea is that each node will write a bit 0 or 1 chosen uniformly at random, if none of its neighbor is in independent set S, and will write 0 if any one neighbor is in S.

If a node v writes 0, and all its neighbors have not yet written their message on whiteboard, then v picks one of its neighbor w uniformly at random from those neighbors who have not written their message. Node v uses its writing opportunity to advise w with an advise bit 1, so that w can decide to write 1 or 0 depending on whether any of its neighbor is in S. This helps to ensure that there is at least one node at a distance at most 2 from v in set S. We now formally describe our algorithm which each node will follow.

Algorithm 3: 2-Ruling Set

```
1  Each node v_i will check if there is any advise for it;
2  if yes then
3      if no neighbor of v_i has written 1 then
4          write 1;
       else
5          write 0;
       end
   else
6      check the bits written by neighbors of v_i;
7      if none wrote 1 then
8          choose a bit from {0,1} u.a.r.;
9          if 0 then
10             if some neighbors are pending then
11                 write 0 and give a advise bit 1 to any
                   one neighbor v_j chosen u.a.r. among the
                   pending ones;
               else
12                 write 1;
               end
           else
13             write 1;
           end
       else
14         write 0;
       end
   end
```

Each node in the graph will follow the above algorithm, when it is chosen by the adversary to write on the whiteboard and will write its message. To compute set S we will just include the nodes who have written 1 and exclude those who wrote 0.

Lemma 3. *When all the nodes have written their message according to Algorithm 3 The set we compute from the whiteboard is a 2-ruling set.*

Proof: When a node gets it turn to write on the whiteboard, it will write 0 if any of its neighbor is in set S which is done in line 5 and line 14 of the algorithm. So no two neighbors can be in set S.

Now we need to check that at least one neighbor which is at most at distance 2 from a node v is in set S. For this if all the neighbors of v have written their messages and none is in S then it will write 1 (line 12), thus maintaining the conditions of independent set.

If few neighbors are pending and v picks 0, so it will give a advise bit to any of its neighbor v_j in lines 9-11 which will make sure either v_j or any of its neighbor is in set S (lines 2-4). So for any node v_i if it is not in S then its neighbor which is at distance ≤ 2 from v_i will be in S. Hence set S is a 2-ruling set. ■

Algorithm 3 and Lemma 3, leads us to the following theorem.

Theorem 4. *The problem of finding a 2-ruling set can be solved in SimSync(f(n)) model for f(n)=$O(\log n)$ in one round.*

Now using Algorithm 3, after the first round to compute S, we can compute MIS in second round using set S. In the second round any node v that is not in S and non of its neighbors are in S, will write 1 and will be added to S. Nodes which are in S or whose neighbor(s) is in S will write 0. We will consider the updated S at any given time as any node can compute current S from the contents of the whiteboard. The above discussion leads to following theorem.

Theorem 5. *The problem of finding maximal independent set (MIS), can be solved in 2 rounds in SimSync(f(n)) model for $f(n) = (O(\log n)$ bits.*

IV. Finding the Connected Components of a Graph

In this section, we will focus on the problem of finding the connected components of a given graph. Our techniques cover the four shared whiteboard models, starting with the Free models first. We show that in Free models, we can solve the problem in one round, whereas in Simultaneous models we get lower bound of $(O(\log n/\log\log n))$ rounds and a upper bound of $O(\log n)$ rounds. So the Free models are more powerful than Simultaneous models even in the case of multiple rounds.

A. One Round Protocol for Free Models

In this section we work with the FreeSync and FreeAsync models. Recall from Section II that in these models, nodes are free to become active as per their activation function, and they will generate their messages as soon as they become active in case of asynchronous model, whereas they will generate their message only when they are chosen to write in case of synchronous model.

In [3] Becker et al. have shown that BFS can be solved in one round in FreeSync model and hence connected component can be found in one round. But for the FreeAsync model he has conjectured that BFS cannot be solved for general graphs in one round, here we will show that still connected components can be found in one round in FreeAsync model. Following is the algorithm to find connected components in FreeAsync model in one round.

Theorem 6. *Finding the connected components of a graph can be solved in one round in FreeSync and FreeAsync model. Further, each node has to write message of size $O(\log n)$ bits on the whiteboard.*

Algorithm 4: Connected Components in Free Model

```
while state(v) = awake do
    Node v_i with highest ID changes its state from
    awake to active;
    v_i writes its ID on whiteboard, neighbors of v_i
    become active;
    while state(v) = active do
        node v_j which the adversary chooses to write,
        writes its ID;
        Neighbors of v_j become active;
    end
end
```

Proof: We will prove that Algorithm 4 runs correctly and we will get connected components in the graph. In fact Algorithm 4, will give us connected components in the form of spanning trees. At the beginning, node v_n will activate, and then its neighbors will activate, and as soon as any neighbor writes its message on whiteboard then its neighbor will activate. The inner $while$ loop will make sure that all the nodes belonging to the same connected component activate and write their message.

If there are no active nodes and there is still some node left in *awake* state, then the outer $while$ loop will make sure that of the remaining nodes the one with highest ID activates. So the algorithm will execute, till all the nodes don't go to *terminate* state. In the end we will get connected components with representative as highest ID nodes. ■

B. Lower Bounds for the SimSync Model

In the simultaneous synchronous model, all the nodes are active and the order in which nodes write is chosen by the adversary, and the nodes will generate their message when the adversary chooses them to write on the whiteboard depending on the contents of the whiteboard and their local knowledge. We construct a graph, for which we will show that any general algorithm like choosing the lowest ID neighbor or highest ID neighbor or average of the IDs of neighbors etc. in SimSync model will take $O(\log n/\log\log n)$ rounds to find the connected components which will give us the lower bound.

1) Structure of the Graph: Now we will construct the graph for which any general deterministic algorithm will take $O(\log n/\log\log n)$ rounds. We first give the structure of the graph and then assign the IDs to the nodes. We start with two nodes a and b and an edge between them. Let d be the degree of the nodes a and b that we wish to create. We will add trees of size, $k!/1! + k!/2! + k!/3! + ... + k!/k!$, for $k = 1, 2, \cdots, 2(d-1)$, alternatively to the nodes a and b.

For a given k between 1 and $2(d-1)$, each such tree added to the nodes a and b would be such that it has $k!$ nodes as leaves, $k!/2!$ nodes at the level prior to leaves, and so on. Further, the nodes will be uniformly distributed in the subtrees, such that root of the subtree will have k children, each child of root will have $k-1$ children, then their child will have $k-2$ children, all the way up to nodes which have 2 children. Figure 1 shows the construction of graph step by step, where we have added trees of size $\{1(1!/1!), 3(2!/1!+2!/2!), 10(3!/1!+3!/2!+3!/3!)\}$ nodes, alternatively to nodes a and b, starting with node a. In the graph in Figure 1, we have chosen $d = 2$.

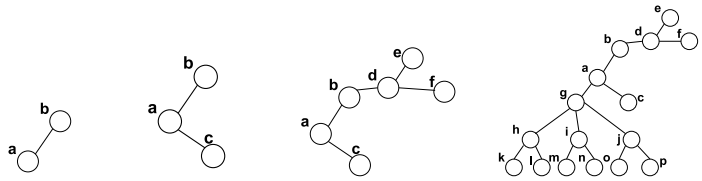

Fig. 1: Step by step construction of our graph

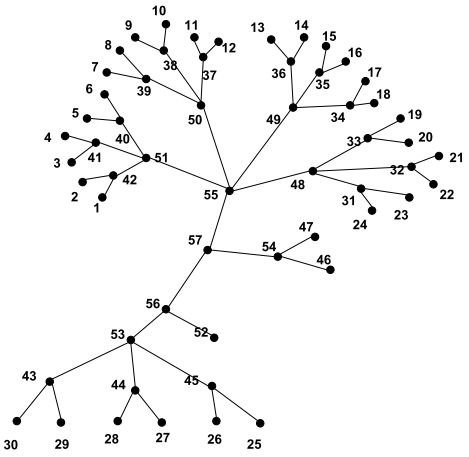

Fig. 2: Numbering of Graph

2) Numbering of nodes for the example graph: We consider node a and node b and number them v_{n-1} and v_n respectively. Then we consider node b as the root of our graph, and then number the nodes level by level starting with bottom most level starting from v_1.

We consider the subtrees connected to node a and b, if two nodes are at the same level, but in different subtrees then the nodes of subtree connected to node b will be numbered first. If the nodes are in same level, but in different subtrees both connected to same center i.e. a or b, then the nodes of smaller subtree would be numbered first.

When we number the nodes at any level whose one level below is already numbered, then the node with highest ID children, should be numbered with lowest number among the remaining numbers.

Considering our graph in Figure 2, we consider v_{57} as root and at level 0, so we have have levels from $\{0, 1, .., 4\}$. We number nodes at level 4 from $\{1, 2, ..., 24\}$ and then from $\{25, ..., 30\}$, then we move to level 3, and number parents of v_{24} and v_{23} as they are in subtree connected to node b or v_n. Then we number nodes of subtree connected to node a at level 3, then we move to nodes at level 2 and so on.

3) Value of k for such a graph: In the graph constructed above we have not mentioned the value of k. In this section we will try to calculate the value of k in terms of the number of nodes, n, in the graph. Notice that for a given k, the number of nodes in that subtree is $r_k = k!/1! + k!/2! + k!/3! + ...k!/k!$. We can simplify this is $r_k = (e-1)\cdot k!$. Hence the total

number of nodes in the graph is $2 + \sum_{k=1}^{2(d-1)} r_k$. If n is the number of nodes in the graph, then from the above, we can see that $d = \Theta(\log n / \log\log n)$. Further, the largest subtree we add to the nodes a (or b), has $\Theta(d)$ levels.

To arrive at a lower bound for deterministic algorithms, we proceed in two steps. We first consider an algorithm in which each node joins the component that its lower numbered neighbor belongs to. Such a rule is used in most parallel/distributed algorithms for connected components [12], [6]. For such an algorithm, we show the following lemma and then generalize the result for general deterministic algorithms.

Lemma 7. *Consider a deterministic algorithm in the SimSync shared whiteboard model with each node allowed to write $O(\log n)$ bits in each round. Suppose that according to the algorithm, each node will join the component to which its lower numbered neighbor belongs to. Then, the algorithm requires $\Omega(\log n / \log\log n)$ rounds.*

Proof: We prove the above lemma with the help of our example graph in Figure 2. Recall that in the SimSync model, the adversary can choose the order in which the nodes are writing on the whiteboard. The basic idea therefore is to show an adversary strategy through which nodes v_n and v_{n-1} will require several rounds before they join a common connected component. In that case, we can say that the algorithm requires a large number of rounds.

This can be achieved by adversary choosing the nodes in decreasing order of their IDs i.e., v_n then v_{n-1} and so on. Each node can write only $\log n$ bits, apart from its ID, and choose its lowest ID neighbor. For our example graph, with the above ordering, nodes v_n and v_{n-1} will join a common component only in round ℓ where ℓ is the number of levels the largest subtree added. As $\ell = \Omega(\log n / \log\log n)$, it will take $\Omega(\log n / \log\log n)$ rounds to find the connected components of a graph in the SimSync model. ■

Using Lemma 7, we give the following theorem which is generalization of Lemma 7, as we can change the numbering for choosing highest ID algorithm or average of IDs algorithm.

Theorem 8. *There exists no deterministic algorithm to find the connected components in a graph in SimSync model in less than $\Omega(\log n / \log\log n)$ rounds provided each node can write only $\log n$ bits in each round apart from its ID.*

The above theorem, gives us the lower bound on the number of rounds required to find connected components in SimSync model. We have proved it for any general and simple deterministic algorithm like choosing lowest ID, highest ID, maximizing the size of the component, average ID etc. that we can generate the graph in such a way that it will take $O(\log n / \log\log n)$ rounds to find the connected components, but we leave it as an open problem to extend it for any deterministic algorithm.

We can show that the parallel algorithm by Shiloach and Vishkin [12], can be simulated in the SimSync model which will give us the upper bound of $O(\log n)$ rounds to find connected components for any general graph. We show the following theorem.

Theorem 9. *The problem to find connected components can be solved in $O(\log n)$ rounds in the SimSync model, when each node is allowed to write a message of size $O(\log n)$ bits on the whiteboard.*

Proof: The proof for the above theorem is a result of adapting the Shiloach-Vishkin algorithm [12] to the shared whiteboard model. ■

C. Bound for SimAsync Model

Recall from Section II that the SimAsync model is the weakest of all the models. Hence, the lower bound result from Section IV-B on the SimSync model applies to the SimAsync model too. However, we show that the algorithm of Shiloach and Vishkin [12] can be adapted to run in the SimAsync model too. Hence, we have the following theorem.

Theorem 10. *The problem to find connected components in a graph can be solved in $O(\log n)$ rounds in SimAsync model, when each node is allowed to write a message of $O(\log n)$ bits on the whiteboard.*

V. Conclusion

Our work indicates that there are interesting observation to be made in the shared whiteboard model as far as graph algorithms are concerned. Our work has the potential to form a basis for gaining a deeper understanding of the shared whiteboard model. Further questions that are worth answering center around the presence of a heirarchy of problems with respect to the round complexity of their solutions in the multiround shared whiteboard model.

References

[1] L. Barenboim and M. Elkin. *Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition*. PODC pages 25-34, 2008.

[2] L. Barenboim and M. Elkin. *Deterministic distributed vertex coloring in polylogarithmic time*, PODC, pages 410-419, 2010.

[3] F. Becker, A. Kosowski, N. Nisse, I. Rapaport and K. Suchan. *Allowing Each Node to Communicate Only Once in a Distributed System: Shared Whiteboard Models*, SPAA pages 11-17 2012

[4] 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*, IPDPS pages 508-514, 2011

[5] K. W. Chong, Y. Han and T. W. Lam. *On the parallel time complexity of undirected connectivity and minimum spanning trees* SODA, pages 225-234, 1999.

[6] D. S. Hirschberg, A. K. Chandra, and D. V. Sarwate. *Computing connected components on parallel computers*, Commun. ACM, 22(8), 461–464, 1979.

[7] K. Kothapalli and S. V. Premmaraju. *Super-fast 3-ruling sets*. FSTTCS, pages 136-147, 2012.

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

[9] M. Luby. *A simple parallel algorithm for the maximal independent set problem*. *SIAM J. Comput.*,15(4):1036-1053, 1986.

[10] Y. Métivier, J. M. Robson, N. Saheb-Djahromi and A. Zameri. *An optimal bit complexity randomized distributed MIS algorithm*, *J. Distributed Computing*,23(5):331-340, 2011

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

[12] Y. Shiloach and U. Vishkin, *An $O(\log n)$ Parallel Connectivity Algorithm*, J. Algorithms,3(1):57-67, 1982.