Empirical Study on an improved distributed algorithm for maximal independent set

BOWEN XUE*, University of Washington

Additional Key Words and Phrases: MIS, distributed computing

ACM Reference Format:
Bowen Xue. 2019. Empirical Study on an improved distributed algorithm for maximal independent set. 1, 1 (December 2019), 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

This project implements and evaluates an improved randomized distributed algorithm, by Mohsen Ghaffari [1], for solving Maximal independent set problem, which is important in symmetry breaking for distributed communication system. The software is available at https://github.com/bx3/distributed_MIS, whose input is a graph stored in the form of adjacency list and whose output is a maximal independent set. The implementation uses multi-thread framework to simulate multiple independent nodes, and communication among nodes are achieved through message passing; a single coordinator collects results reported from each node to generate a maximal independent set. Thousands of randomly generated test graph of node size varying from 100 to 10000, of max degree less than 1000 are feed into the algorithm; all results passes a maximal independent set verifier which shows the correctness of the implementation. The current implementation solves a connected graph, but extension to multiple components graph could be an extension to this work.

2 THEORETICAL ALGORITHM DESCRIPTION

The MIS algorithm relies on a standard distributed computation model called LOCAL [2], which is a communication network of independent nodes, whose structure can be abstracted into a graph $G = (V, E)$. Each node only knows and communicates with its direct neighbors. All Communication are performed under a synchronous environment, such that all nodes need to finish the current step before starting the next step.

The algorithm is a randomized distributed algorithm which runs iteratively to generate a correct result. The correctness is guaranteed, and the execution time, as measured in iteration, is likely to terminate quickly. The fast termination is proved by a theorem that the probability for a node, v, not making decision on joining MIS within $\beta(\log deg + \log 1/\epsilon)$ iteration is at most ϵ, for some constant β, and deg represents the degree of v at the initial setup.

The algorithm can be summarized as follows: At round t, every node, v, probabilistically decides for itself to be marked according to its desire level, $p_t(v)$, marking is the first step for joining the MIS. After all nodes has made the marking decisions, they communicates their marks to neighbors. A node v joins MIS if none of its neighbors is marked, and this node is removed from the graph along with its neighbors. If at least one of neighbors of v get marked, v adjusts its desire level by using the following rule, where $d_t(v) = \sum_{n \in N(v)} p_t(u)$ as effective-degree of the node v

$$p_{t+1}(v) = \begin{cases} p_t(v)/2 & \text{if } d_t(v) \geq 2 \\ \min\{2p_t(v), 1/2\} & \text{if } d_t(v) < 2 \end{cases},$$

The intuition behind the algorithm is that, when the effective degree of a node is small, it is more likely to join MIS, and therefore its desire should be increased. On the other hand, if its effective degree is high, all nodes should decrease its desire, such that it becomes more likely to have a single one to mark itself in the neighborhood, not multiple. The correctness of the algorithm is natural, since the MIS node along with its neighbors are removed altogether.

3 ARCHITECTURE OVERVIEW AND LOCAL NETWORK

The implementation consists of four components: Coordinator, Node, Creator and Verifier. Creator is used for creating random testing graph data, and Verifier is for verifying the MIS result; both are discussed in later sections. One Coordinator and Multiple Nodes constitutes a LOCAL network. Given an input graph, each node creates channels and shares its sender side of channel to both its neighbors and the coordinator; each nodes keeps the receiver side to read messages. The coordinator also creates a channel, and gives sender to all Nodes. The implementation is programmed in Rust languages, which provides a FIFO buffered message queue. Fig. 1 gives is a visualization of a setup of 5 nodes:

4 NODE AND COORDINATOR

Nodes and Coordinators maintains the data structure for its connections, and they implements a LOCAL network. Every Node runs in an independent threads, holding context of its neighbors. Coordinator runs a separate threads for orchestrating actions of nodes. Interactions within an iteration is described in Table.1, which summarize and structures the procedures.

4.1 Procedures Description

In Round 1, each nodes indicates its desire to join MIS by setting itself to be marked or non-marked. They exchange information to let neighbors know. Since all channel are asynchronous in nature, the Coordinator needs to make sure all nodes are complete before proceeding to next actions; the Coordinator knows by checking if

*Student project of group size 1

Author's address: Bowen Xue, bx3@uw.edu, University of Washington, Seattle, Washington, 98105.

XXXX-XXXX/2019/12-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

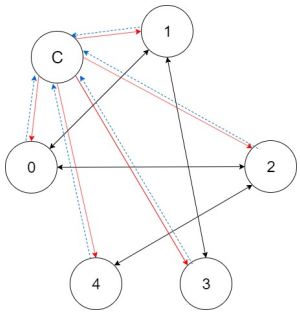

Fig. 1. LOCAL network of 5 nodes; 'C' stands for Coordinator; 'number' indicates node id; Arrow represents message direction; Solid line represents Network Message; Black lines are network message among nodes; Red lines are network messages from the coordinator to nodes; dashed blue messages are synchronization messages from nodes to the coordinator

every node has sent the message to indicate a node's completion. After a node sends completion to coordinator, the node can respond to query from its neighbors asking if the node is marked.

In Round 2 a, every node decide if itself should join the MIS set, by checking if any of its neighbors are marked. Then every node sends its MIS decision to all its neighbor. Every node needs to wait all its neighbors' MIS decision messages to proceed; otherwise a node keeps waiting. In Round 2 b, after receiving all messages, every node decides if itself should leave the network. If a node is a MIS node, it send message to Coordinator indicating it wants to leave, and it belongs to MIS. If a node is a neighbor to a MIS node, it also needs to leave. All leaving nodes wait for confirmation from the coordinator before terminating their threads. Coordinator needs to receive decision from all nodes before proceed.

Round 3 is an step in this implementation for re-configuring the network. In Round 3, Coordinator collects all leaving nodes, and sends a summary to all nodes. MIS Nodes and MIS neighbor Node terminates thread after receiving the summary. Other nodes clean their neighbor. If a node finds itself having no neighbor, it indicates that all its neighbors are not in MIS, therefore the node automatically join MIS by notifying the Coordinator, then terminates.

4.2 Round analysis

The implemented algorithm needs 4 round trip time to complete, whereas the theoretical paper [1] claims only 2 round trip time. The extra 2 round trip consists of 3 Sync and round 3, which are used for network synchronization and network reconfiguration.

5 CREATOR AND VERIFIER

Creator creates a random single component graph for the distributed randomized algorithm to solve. It takes two parameters: number of node and max degree for every node. The graph is stored in a adjacency list, every node has a ID starting from 0 and increments by 1. To create random edges, Creator generates a random integer from 1 to max degree, then continue to find new neighbor by drawing random integer from 0 to number of node.

Table 1. Procedures

step	comments
round 1.a	Decide if the node v itself is marked based on $p_t(v)$, and Request all neighbors to send $p_t(u)u \in N(v)$
round 1.b	Respond neighbors and send synchronization to Coordinator
Sync	Coordinator collects all messages from all nodes, and start round 2
round 2.a	Each node v decides if join MIS or updates its desire level, according to marks from neighbors; each node notifies its neighbor about its MIS decision
round 2.b	each nodes must receive all MIS messages from all its neighbors; if any decides to join MIS, the node needs to leave, too; each node send leaving/staying message to Coordinator
Sync	Coordination collects all leaving/staying messages and updates its current MIS; Coordinator also records nodes to remove, and sends the removing list to remaining nodes to update neighbors. removing list contains not only MIS node, also neighbors of MIS nodes
round 3	Each node checks if itself is contained in the removing list; if yes, the node terminates its thread; if no, remove all neighbors in the list; check the number of neighbor > 0; if yes, sends stay message to Coordinator; but if no, sends leaving message to Coordinator, and indicates that it joins MIS, then terminate its thread
Sync	Coordinator must collects all messages from all remaining nodes, update its current MIS. Start another iteration from round 1.a

Verifier takes a MIS output generated from the algorithm and verify according to Algorithm 1 to check correctness, n represents number of node, i=1:

Algorithm 1 Verifier Algorithm

```
for i ≤ n do
    if i not in MIS then
        if any neighbor in MIS then return FALSE
    i ← i + 1
return TRUE
```

6 EXPERIMENT

The software takes 3 input parameters to run the algorithm: number node(#n), max degree(# d) and number run(#r). The software, in response, creates (#r) number of graph, with all of them havig the same configuration, i.e. (#n) nodes with max degree (#d). Then the

Table 2. Experiment setup and result, comma separated values are mean and standard deviation; the unit of time is second

node(n)	deg(d)	run(r)	iter	mis size	time
10	5	1000	1.8, 1.1	6.7, 1.55	3, 3
100	10	1000	1.1, 0.27	77, 16	23, 5
100	50	1000	3.4, 1.2	66, 16	49, 12
100	80	1000	4.6, 1.7	53, 16	81, 24
1000	10	1000	1, 0	480, 47	219, 17
1000	100	1000	1.05, 0.39	561, 125	337, 164
1000	500	343	4.5, 3.3	605, 172	2391, 1656
1000	800	144	6.6, 3.6	484, 163	5058, 2759
10000	100	1000	1, 0	5052, 707	7080, 468
10000	1000	145	1.49, 1.73	6200, 1369	27185, 23658

software runs the randomized distributed algorithm and generates an output file with name "nXXdXXrXX". The output file contains all statistics including: number of round to complete, time to compute, output mis, input graphs.

To run the software, clone git hub repo at https://github.com/bx3/distributed_MIS and install. Then type command ./run.sh 10 5 3 for generating 3 random graphs each with 10 nodes and 5 max degree, and run the randomized distributed MIS algorithm.

6.1 Result

Since Creator can generates random graph, and verifier can test for correctness, there is unlimited number of possible run. The experiment result is summarized in Table 2. All solution from the algorithm passes the Verifier.

The results on the table confirms that each node can quickly decide when to leave the network. It takes more iteration to complete as max degree of the graph becomes large. Some setup is not repeated 1000 times in the table is due to the time limitation to run the experiment, not for its correctness.

7 CONCLUSION AND FURTHER WORK

The project implements a randomized distributed algorithm for solving maximal independent set, which is important for applications which need to breaking network symmetry. The project not only implements the MIS algorithm, but also designs a synchronous LOCAL network using multi-thread and asynchronous message passing channel. The project also designs a random graph generator and a verifier to test for output correctness; after heavy testing, it demonstrates that both LOCAL and the MIS algorithm are correct.

There are multiple tasks not done yet due to the time limitation and complexity to analyze and debug a distributed algorithm within a brief period. First, the algorithm currently works for connected graph, an extension to multiple component graphs would be the next step. Second, although the implementation can generate the correct MIS output, it does not release threads properly, as the result, the implication can only handle a single shot run, and cannot write experiment loop as part of the software; currently I use a shell script to start a MIS process.

The system can be easily modified to run Luby's Algorithm [3] by modifying round 2 of this implementation to find out the strict local minimal. Due to the time constraint, it is not experimented, and it will be interesting to try other novel idea.

8 REFERENCE

[1] http://arxiv.org/abs/1205.2525

[2] David Peleg. Distributed Computing: A Locality-sensitive Approach. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[3] Michael Luby. A simple parallel algorithm for the maximal independent set problem. In Proc. of the Symp. on Theory of Comp. (STOC), pages 1–10. ACM, 1985