Constructing an MIS in Synchronous Graphs With Single Bit Messages

based on

An optimal bit complexity randomized distributed MIS algorithm

Jakub Oskwarek

January 2024

Abstract

This project aims to understand the MIS construction algorithm proposed by Métivier, Robson, Saheb-Djahromi, and Zemmari, to outline the necessary intuitions and proofs for it, and to implement it in Go.

1 Network properties

The algorithm [1] is applicable to any graph, as long as the system adheres to the following assumptions:

- Nodes **do not have identifiers** and are effectively indistinguishable. As an implementation detail, I believe that seeding the local pseudorandom generators is a reasonable exception to this rule.
- The algorithm proceeds in so-called *bit rounds*, i.e. **synchronous rounds** in which the nodes exchange messages consisting of single bits.
- The communication channels are **fully reliable** — bits are neither lost nor flipped in transmission.

We will be referring to the number of nodes as n, but the algorithm does not have to know this value. In fact, the algorithm barely has to know anything. A node only needs to be aware of which communication channel corresponds to which neighbor and to be able to draw random bits.

2 An evolving idea

In the original paper, the authors develop an algorithm that, with high probability, uses $\mathcal{O}(\log n)$ bit rounds. They recall other results concerning the coloring problem and its reduction to the MIS problem in order to draw the conclusion that their algorithm is optimal.

They start with an idealistic and intuitive approach.

2.1 Algorithm $\mathcal{A}$

This will be a greedy algorithm. The natural observation is that if we find a strict local minimum in the graph (for some definition of a "minimum") and put it into the MIS once and for all, then this node's neighbors can never be in the MIS (by the very definition of the problem). So, once we put a node into the MIS, we can ignore it and its neighbors in the operations that will follow.

Thus, the first algorithm proceeds in *phases* as follows: in every *phase*:

1. Sample a random number uniformly from $[0, 1)$ — this will be our criterion for the minimum
2. Send this number to all neighbors
3. Receive such number from all neighbors
4. Now you know whether you are a strict local minimum:
   - if you are, then enter the MIS and announce it to all neighbors
   - else, do not send anything
5. Receive analogous information from all neighbors
6. Now you know whether you have a neighbor in the MIS.

- if you do, then you will never have the chance to be in the MIS, so announce it to all neighbors
- else, do not send anything

7. Receive analogous information from all neighbors
8. Now you know which neighbors can be ignored from now on:
   - those already in the MIS
   - those with no chance to enter

   So, delete them.

The node can finish processing when it knows its status in the MIS (definitely inside or definitely not).

There is a problem: we wanted an algorithm that only sends single bit messages. All real numbers in $[0, 1)$ are far from being representable with one bit. This calls for some improvement. Please give a round of applause for

2.2 Algorithm $\mathcal{B}$

It is a simple modification of algorithm $\mathcal{A}$ — basically, it splits each phase into smaller parts and sends the sampled numbers bit by bit. A node also exchanges some additional information about its status in the MIS. It allows everyone to keep track of which neighbors are in the MIS, which have no chance of entering in the current phase, and which have no chance of entering at all.

Each phase is accomplished by repeating these operations. When u knows its status in the MIS or has no more active neighbors, it can break this loop and start another phase, if necessary, or terminate.

However, the next phase cannot always be started immediately because all the other vertices must be ready. A question arises: is it really necessary? The answer is provided by

2.3 Algorithm $\mathcal{C}$

Such delays are, in fact, not necessary. Instead of patiently waiting for the start of another phase, we can do some honest work during that time. In particular, if we, as a node u, have a neighbor v with whom we have already broken the symmetry for this phase, what is stopping us from attempting to break it in advance for the upcoming phases as well? Therefore, on a given node, we will be running two alternating processes:

1. perform a round of symmetry breaking with the neighbors and remember the results — they may (and probably will) refer to many different phases
2. update the MIS status and announce the changes, if necessary

At a node u, we will record three kinds of information, all represented by single bits:

- `wins`$[v][p]$ — whether we won against neighbor v during the symmetry breaking for phase p (or undefined, if symmetry is yet to be broken with that neighbor for that phase)
- `ins`$[v][p]$ — whether v entered the MIS in phase p (or undefined, if not yet decided)
- `outs`$[v][p]$ — whether v was removed from the graph in phase p (or undefined, if not yet decided)

We will also maintain `win-phase`$[v]$, `in-phase`$[v]$, `out-phase`$[v]$, and `index`$[v]$, which will enable us to know at which phase we are with neighbor v for each kind of information, and at which bit in that phase. The procedure $b(p, i)$ returns the i-th bit of the p-th phase by simply reading its value, if already sampled, or sampling it for the first time and recording it for future reference.

In the following code, N will denote the set of *all* neighbors, X will denote the set of *active* neighbors (i.e. those who have not yet decided to exclude themselves from the graph at any phase), and P will denote the current phase considered by the second process.

Algorithm 1 A logical round of algorithm $\mathcal{C}$ (composed of three physical send-receive rounds) at a node u

```
▷ Perform a round of symmetry breaking ◁
for v ∈ X do
    send b(win-phase[v], index[v]) to v                ▷ send the next bit for the appropriate phase
for v ∈ X do
    receive b_v from v
    if b_v ≠ b(win-phase[v], index[v]) then
        wins[v][phase[v]] ← [b(win-phase[v], index[v]) = 0]   ▷ the symmetry is broken and we win if we sent a zero
        win-phase[v] ← win-phase[v] + 1                ▷ we can start the symmetry breaking for another phase
        index[v] ← 0                                   ▷ we will, naturally, start that next phase from the beginning
    else
        index[v] ← index[v] + 1                        ▷ the symmetry is not broken and we will need to exchange another bit
▷ Check whether we have enough information to decide if we enter the MIS ◁
if ∀_{v∈N} wins[v][P] ≠ ⊥ ∧ not already decided for phase P then
    for v ∈ X do
        i ← [∀_{v∈N} wins[v][P] = 1]                   ▷ we enter the MIS if we won against all our neighbors
        send i to v
        if i = 1 then
            enter the MIS
for v ∈ X do
    receive i_v from v
    if i_v ≠ ⊥ then
        ins[v][in-phase[v]] ← i_v
        in-phase[v] ← in-phase[v] + 1
▷ Check whether we have enough information to decide if we should be deleted ◁
if ∀_{v∈N} ins[v][P] ≠ ⊥ ∧ not already decided for phase P then
    o ← ∃_{v∈N} ins[v][P] = 1 ∨ we are in the MIS
    for v ∈ X do
        send o to v
    if o = 1 then
        terminate when the round is completed
for v ∈ X do
    receive o_v from v
    if o_v ≠ ⊥ then
        outs[v][out-phase[v]] ← o_v
        out-phase[v] ← out-phase[v] + 1
        if o_v = 1 then
            X ← X \ {v}                                ▷ neighbor v has decided to be removed from the graph
▷ Try to advance to the next phase ◁
if ∀_{v∈N} outs[v][P] ≠ ⊥ then
    N ← {v ∈ N : outs[v][P] = 0}                       ▷ keep only the neighbors that were not disconnected in phase P
    P ← P + 1
```

This is the final version of the algorithm.

3 Intuitions about correctness

Let us, for a moment, "tune out" the interspersed rounds of symmetry breaking and only look at these decisions of algorithm $\mathcal{C}$ that update the contents of the independent set or the "shape" of the graph, in a per-phase manner.

The probability that algorithm $\mathcal{C}$ will go on forever is zero. To see that, let us look at a phase that terminates. Each of the bit strings generated by the nodes for this phase represents the binary expansion of some number $r \in [0, 1)$. Let us pick a node u that generated the bit string corresponding to the smallest of such numbers. This node cannot have a neighbor with the same bit string because in such scenario the phase would not have terminated. Thus, u is a local minimum, so it will enter the set and exclude itself from the graph. Therefore, with each phase that terminates, the number of nodes in the graph decreases. Since n is finite (which is not explicitly stated in the original paper, but it seems like a fair assumption), in order to go on forever, algorithm $\mathcal{C}$ would need some phase to never terminate. However, the probability of such event is zero because it would require a pair of nodes to generate identical infinite bit strings.

Now, let us investigate, how algorithm $\mathcal{C}$ makes its decisions. In every phase, it waits for the symmetry to be broken and only then does it decide whether a node should enter the MIS. Then it waits for all the messages carrying such decisions in order to decide whether it should be excluded, and then it waits for all such messages in order to update its local "vision" of the graph. Thus, we can think of algorithm $\mathcal{C}$ as making decisions consistent with those that $\mathcal{A}$ would make if, for each phase, it sampled the real numbers corresponding to the bit strings that $\mathcal{C}$ saw.

On the other hand, it is easy to see that algorithm $\mathcal{A}$ constructs a valid independent set — there can never be more than one strict local minimum, so no two adjacent nodes will ever end up together in the resulting collection in the same phase. They cannot enter the set in different phases either — once the first one becomes a member of the collection, the other one is immediately excluded from the graph.

The set constructed by $\mathcal{A}$ is also maximal because the only way to terminate without being chosen as a member of the constructed collection is to receive a signal that a neighbor has entered this collection.

Thus, we know that $\mathcal{C}$ constructs an MIS.

4 Proof of complexity (outline)

Lemma 1. *With probability $1 - o\left(\frac{1}{n}\right)$, algorithm $\mathcal{A}$ needs $\mathcal{O}(\log n)$ phases.*

Proof. Firstly, let us prove that, in expectation, the number of edges in the graph is halved in every phase. To do that, let us consider situations in which a node u samples the smallest number among the nodes $N(u) \cup N(v)$, for some $v \in N(u)$. The probability of such event is at least

$$\frac{1}{\deg(u) + \deg(v)}$$

as it roughly corresponds to taking the first spot in a sorted sequence of $\deg(u) + \deg(v)$ numbers. Then, u will become the local minimum and remove v together with its $\deg(v)$ edges. By linearity of expectation, the expected number of edges that disappear in such situations is

$$\begin{aligned}
&\frac{1}{2} \sum_{\{x,y\} \in E} \left(\deg(x) \cdot \Pr(y \text{ removes } x \text{ and all its edges}) + \deg(y) \cdot \Pr(x \text{ removes } y \text{ and all its edges})\right) \\
&\qquad = \frac{1}{2} \sum_{\{x,y\} \in E} \left(\deg(x) \cdot \frac{1}{\deg(x) + \deg(y)} + \deg(y) \cdot \frac{1}{\deg(x) + \deg(y)}\right) = \frac{1}{2} \sum_{\{x,y\} \in E} 1 = \frac{|E|}{2}
\end{aligned}$$

The lemma follows from choosing a constant a such that after $a \log n$ phases the expected number of edges is $o\left(\frac{1}{n}\right)$. In the paper, $a = 4$ is deemed sufficient. □

Lemma 2. *In algorithm $\mathcal{C}$, over a single edge, symmetry is broken for $a \log n$ phases in $\mathcal{O}(\log n)$ rounds with probability $1 - o\left(\frac{1}{n^3}\right)$.*

Proof. The rounds are independent Bernoulli trials — each one breaks the symmetry with probability $\frac{1}{2}$. Assume that we have r rounds. Let W be the number of `wins` resolved during those r rounds. We have

$$\mu = \mathbb{E}[W] = \frac{r}{2}$$

The lemma is then proven by plugging its statement, together with the above observation, into Chernoff's inequality and solving for all the necessary constants. □

Theorem 1. *Algorithm $\mathcal{C}$ terminates in $\mathcal{O}(\log n)$ rounds with probability $1 - o\left(\frac{1}{n}\right)$*

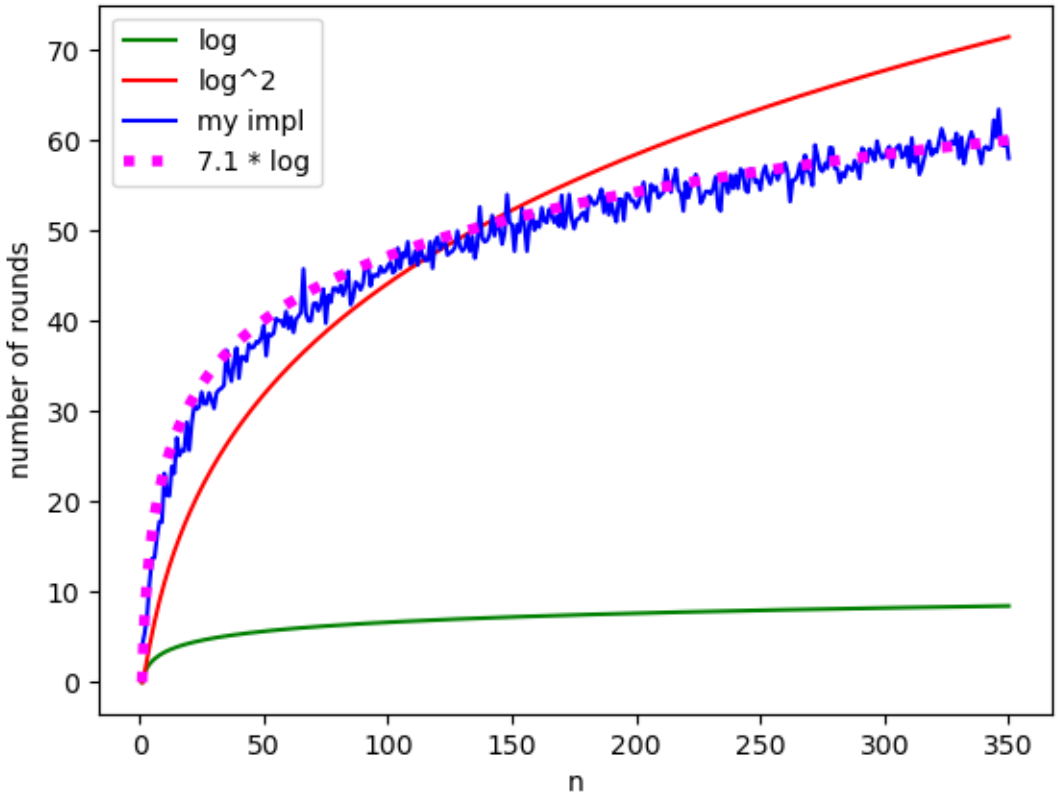
Proof. Since there are $\mathcal{O}\left(n^2\right)$ edges, then by lemma 2 and the union bound, the probability that there exists an edge over which the symmetry was broken fewer than $a \log n$ times is $o\left(\frac{1}{n}\right)$. The number of rounds in the second process of algorithm $\mathcal{C}$ is bounded by a constant factor times the number of rounds in the first process. By lemma 1, only $a \log n$ phases are needed with probability $1 - o\left(\frac{1}{n}\right)$. Combining all that proves the desired property. □

5 Remarks

The exact way in which the two processes of algorithm $\mathcal{C}$ alternate was not entirely clear to me.

1. In my implementation, many bit exchange phases are processed concurrently. However, the second process is only concerned with one phase at a time. I do not know, how it could be made concurrent. I believe that it would bring catastrophic consequences, e.g. a node entering the set in anticipation, only to learn that it should have been deleted some phases earlier. However, the paper does not clearly reject that option.
2. Currently, the logical rounds are organized in triplets: win — in — out. Probably, they could also be organized in quadruplets: win — in — win — out. This might be in closer correspondence with the paper's original phrasing: one round of each process. However, I believe that this only has the potential to shave off a constant.

Despite all those doubts, I believe that my implementation is correct, i.e. $\mathcal{O}(\log n)$ and not $\mathcal{O}\big(\log^2 n\big)$. For n up to 350, the average number of rounds seems to match the logarithmic curve with an empirically adjusted constant equal to 7.1.



It is worth noting that the execution of the algorithm for large n and dense graphs tends to take a lot of real time (for $n = 1000$, around 10 minutes) and quite a lot of messages are sent. In theory, the per-channel bit complexity should be $\mathcal{O}(\log n)$, both on average and with high probability. This might need further investigation.

References

[1] Métivier Yves, John Michael Robson, Saheb-Djahromi Nasser, and Akka Zemmari. An optimal bit complexity randomized distributed mis algorithm (extended abstract). In Shay Kutten and Janez Žerovnik, editors, *Structural Information and Communication Complexity*, pages 323–337, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.