Opiii

October 30, 2024

Contents

1 Dynamic Networks

Dynamic graph networks are graph networks that change over time. Communication is in synchronous, asynchronous or semi-synchronous rounds. Additionally shared memory is possible. Network elements may be failure-free or failure-prone. A classical example are mobile ad-hoc networks. Those are temporary interconnection networks of mobile wireless nodes without a fixed infrastructure. Communication happens whenever mobile nodes come within the wireless range of each other.

Example 1.1. *In mobile ad hoc networks, one may want to colour the graph or maintain a routing mechanism for communication to any particular destination in the network.*

1.1 Almost constant message-passing vertex colouring in a tree

Let T be a tree network with n labelled vertices in $[n]$. Colouring the graph can be done in almost constant, i.e. in $\log^*$ time.

Definition 1.2. $\log^*(x)$ is defined as the number of log functions that need to be applied to x such that the result is at most 1. E.g. $\log^*(16) = 3$ and $\log^* 2^{65536} = 5$.

1. begin by rooting the tree at vertex 0. This defines an order on the tree
2. each parent sends its number to all of its children
3. each child computes the smallest index i where its number differs from the parent's number. It is important to note that this can be done in constant time with suitable hardware
4. It computes a new ID for itself consisting of a trailing bit corresponding to the bit where IDs disagreed. The new ID begins with the binary representation of the digit where the Ids differed.
5. the new ID is now only $\log \log n$ bits long. This is repeated until there are only six distinct numbers left. This takes $\log^*$ rounds each taking only constant time.
6. each parent sends its number to its children which relabel themselves accordingly
7. This is repeated another time and the IDs are taken $\mod 3$ resulting in a three colourin

Definition 1.3. The collection of the initial states of all nodes in the r-neighbourhood of a node v is the r-hop view of v.

Definition 1.4. Let $\mathcal{G}$ be a family of network graphs. The r-neighbourhood graph $N_r(\mathcal{G})$ is defined as follows:
The node set is the set of all possible labelled r-neighbourhoods (i.e. all possible r-hop views). There is an edge between tow labelled r-neighbourhoods V_r and V'_r if V_r and V'_r can be the r-hop views of adjacent nodes.

Lemma 1.5. *For a given family of network graphs $\mathcal{G}$ there is an r-round algorithm that colours graphs of $\mathcal{G}$ with c colours of the chromatic number of the neighbourhood graph is $\chi(N_r(\mathcal{G})) \leq c$.*

Definition 1.6. We define a directed graph B_k which is closely related to the neighbourhood graph. The vertex set is made up of all k-tuples consisting increasing node labels. For two nodes $\alpha = (\alpha_1, ..., \alpha_k)$ and $\beta = (\beta_1, ..., \beta_k)$ there is an edge from α to β if $\forall i$ it holds that $\beta_i = \alpha_{i+1}$.

Lemma 1.7. *Viewed as an undirected graph, B_{2r+1} is a subgraph of the r-neighbourhood graph of directed rings with n nodes.*

Lemma 1.8. *If $n > k$ the graph B_{k+1} can be defined as the line graph $\mathcal{L}(B_k)$ of B_k.*

Lemma 1.9. *It holds that*

$$\chi(\mathcal{L}(G)) \geq \log_2(\chi(G))$$

Lemma 1.10. *For all $n \geq 1$ it holds that $\chi(B_1) = n$. Further for $n \geq k \geq 2$ it holds that $\chi(B_k) \geq \log^{(k-1)} n$.*

Theorem 1.11. *Every deterministic distributed algorithm to colour a directed ring with at most 3 colours needs at least $\log^*(\frac{n}{2}) - 1$ rounds.*

Corollary 1.12. *Every deterministic distributed algorithm to compute a maximal independent set on a directed ring needs at least $\log^*(\frac{n}{2}) - \mathcal{O}(1)$ rounds.*

1.2 <u>MIS</u>

The following randomized algorithm gives a good solution to the maximum independent set.

1. the algorithm operates in synchronous rounds grouped into phases
2. each node marks itself with probability $\frac{1}{2d(v)}$
3. if no higher degree neighbour of v is marked, node v unmarks itself again

4. delete all nodes that joined the MIS and their neighbours as the cannot join the MIS any more

Lemma 1.13. *A node v joins the MIS in step 3 with probability $p \geq \frac{1}{4d(v)}$*

Lemma 1.14. *A node is called good if*

$$\sum_{w \in N(v)} \frac{1}{2d(v)} \geq \frac{1}{6}$$

A good node will be removed in Step 4 with probability $p \geq \frac{1}{36}$.

Lemma 1.15. *An edge is called bad if both its endvertices are bad. Otherwise it's called good. At any time at least half of the edges are good.*

Lemma 1.16. *A bad node has out-degree at least twice its in-degree.*

Lemma 1.17. *The algorithm terminates in expectation in $\mathcal{O}(\log n)$ rounds.*

2 Consensus

In a distributed system with each node starting with input x_i, we speak of consensus if an algorithm can achieve the following properties

1. Agreement: all alive nodes decide on a single value x
2. Validity: the decided value x is one of the initial inputs
3. Termination: each vertex terminates at some point (either voting for one value or crashing)

The following randomized consensus algorithm works in an asynchronous setting with less than half the nodes crashing

1. input bit $v_i \in \{0, 1\}$, $round = 1$, decided = false
2. broadcast $(v_i, round)$
3. while true
4. wait until majority of messages of current round arrived
5. if all messages contain the same value v:
6. propose $(v, round)$, decided = true

7. else:
8. propose $(\perp, round)$ //$\perp$ is a signal of disagreement
9. end if
10. wait until a majority of proposals of current round arrived
11. if all messages propose the same value v:
12. $v_i = v$, decide = true
13. else if there is at least one proposal for v:
14. $v_i = v$
15. else:
16. choose v_i uniformly at random
17. end if
18. $round = round + 1$
19. broadcast $(v_i, round)$
20. end while

Theorem 2.1. *The above algorithm satisfies validity, termination and comes to an agreement. In expectation it takes exponential time.*

2.1 shared coin

The following algorithm allows a dynamic network to use the same coin for all vertices at the same time. Here f is the number of nodes that can turn byzantine. It should hold that $f \leq \frac{n}{3}$.

1. choose local coin $c_u = 0$ with probability $\frac{1}{n}$
2. broadcast c_u
3. wait for $n - f$ coins and store them in the local coin set C_u
4. broadcast C_u
5. wait for $n - f$ coin sets

6. if at least one coin is 0 among all coins in C_u:
7. return 0
8. return 1
9. end if

2.2 byzantine consensus

Definition 2.2. A node which can have arbitrary or malicious behaviour is called byzantine. This includes not sending messages, sending wrong messages, sending different messages to different neighbours and many more. A node that is not byzantine is called correct or truthful.

The following probabilistic algorithm achieves consensus in an asynchronous setting with $< \frac{n}{9}$ byzantine nodes.

1. $x_i \in \{0, 1\}$, $r = 1$, decided $=$ false
2. propose(x_i, r)
3. while not decided
4. wait until $n - f$ proposals of current round r arrived
5. if at least $n - 2f$ proposals contain the same value x: $x_i = x$ decided $=$ true
6. elseif at least $n - 4f$ proposals contain the same value x: $x_i = x$
7. else: choose x_i randomly with $\mathbb{P}[x_i = 0] = \mathbb{P}[x_i = 1] = \frac{1}{2}$
8. endif
9. $r = r + 1$, propose(x_i, r)
10. endwhile
11. decision $= x_i$