Opiii

November 8, 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 <u>mobile ad-hoc networks</u>. 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 <u>Almost constant message-passing vertex colouring in a tree</u>

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 MIS

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 $= x_i$

Lemma 2.3. *Let $f < \frac{n}{9}$. If a correct node chooses value x in line 6, then no other correct node chooses value $y \neq x$ in line 6.*

Theorem 2.4. *The algorithm solves binary agreement for up to $f < \frac{n}{9}$ byzantine nodes.*

Definition 2.5. $N[u] = N(u) \cup \{u\}$

3 Dominating Set

The following algorithm gives an approximation to a minimal dominating set. To this end, we colour vertices white in the beginning, black if they are added to the dominating set S and grey if they are covered by a neighbour in S. For a vertex u we define $W(u) = \{v \in N[u] \mid v \text{ is white}\}$.

1. while v has white neighbours
2. compute $|W(v)|$ and send it to all neighbours at distance at most 2
3. if $|W(v)|$ is largest among neighbours of distance 2
4. join S
5. endif
6. endwhile

Theorem 3.1. *Let S^* be the optimal dominating set and S the one returned by the algorithm. Then $\frac{|S|}{|S^*|} \leq \ln \Delta + 2$. The algorithm takes $\Theta(n)$ rounds.*

In the following we try to push this runtime to sublinear.

3.1 Fast Dominating Set Algorithm

1. $W(v) = N[v]$, $w(v) = |W(v)|$
2. while $W(v) \neq \varnothing$
3. $w'(v) = w(v)$ rounded down to the nearest power of 2
4. if $w(v) = \max_{u \in N_2(v)} w'(u)$ then $v.active = true$
5. else $v.active = false$
6. endif
7. compute active neighbours $a(v) = \{u \in N(v) \mid u.active\}$
8. $v.candidate = false$
9. if $v.active = true$ then
10. $v.candidate = true$ with probability $\frac{1}{\max_{u \in W(v)} a(u)}$

11. endif
12. compute $c(v) = |\{u \in W(v) \mid u.candidate\}|$
13. if $v.candidate$ and $\sum_{u \in W(v)} c(u) \leq 3w(v)$ then
14. node v joins dominating set
15. endif
16. update W, w
17. endwhile

Theorem 3.2. *The algorithm computes a dominating set of size at most* $(6 \ln \Delta + 12)\, |S^*|$.

Lemma 3.3. *Consider an iteration of the while loop. Suppose that a node u is white and that $2a(u) \geq \max_{v \in C(u)} \max_{y \in W(y)} a(y)$ where*

$$C(u) = \{v \in N(u) \mid v.candidate\}$$

Then the probability that u becomes dominated in this iteration is larger than $\frac{1}{9}$.

4 maximal matching

This section introduces a new technique called rounding. The idea is to solve a given integral problem as a continuous problem and then rounding the results to the nearest integer. This often allows for polylogarithmic time complexity.

Definition 4.1. A maximal matching is a subset S of edges s.t. no vertex has two incident edges in S. Furthermore, there are no edges $e \in E \setminus S$ that can be added to S without breaking the first condition.

This problem is a typical integer linear problem, i.e. one where variables x_e for $e \in E$ are in $\{0, 1\}$. The idea is now to allow continuous variables and fix the result by rounding. A non-integral result is called a fractional matching.

Definition 4.2. In a fractional matching we call a vertex v loose if $c_v = \sum_{e \in E(v)} x_e \leq \frac{1}{2}$. An edge is called loose if both its vertices are loose. We call a fractional matching f-fraction if $c_v \geq f \;\forall v \in V$.

The following algorithm runs in $\mathcal{O}(\log n)$ time and yields a 4-approximation to for a fractional matching.

1. $x_e = 2^{-(\lceil \log \Delta \rceil)}$
2. while both endpoints of e are loose:
3. $x_e = 2x_e$
4. endwhile

Theorem 4.3. *The algorithm computes a 4-approximation $\frac{1}{\Delta}$-fractional matching in $\mathcal{O}(\log \Delta)$ time.*

The idea is now to get rid of the fractional edges. This works by starting to either multiply all edges of value $\frac{1}{\Delta}$ by a factor of 2 or by rounding them down to 0. In the next step this is done for edges of value $\frac{2}{\Delta}$ and so on.

Definition 4.4. We define the subgraph G_f as the graph induced by all edges of value f in G.

Definition 4.5. Rounding the graph G_f means to identify a subset of edges of value f in E_f which will be doubled. All other edges in (which are also of value f!) will be assigned value 0. The resulting graph should still be a valid $2f$-fractional matching.

Definition 4.6. A perfect rounding of a graph G is a rounding of the graph such that for all nodes v half of its edges are assigned twice its value and the other half 0. Notice that c_v and the size of the matching remain unchanged.

We start by introducing the idea for bipartite graphs. For this, we need to construct a 2-decomposition of the graph G_f.

Definition 4.7. For a graph G we define a decomposition G' of G by copying each vertex v $\frac{d(v)}{2}$ times. The edges incident to v are distributed among all these copies such that each copy has degree 2 (in the case that $d(v)$ is odd, one copy may have degree 1).

Note that in G' each vertex v has $d(v) \in \{1, 2\}$. I.e. G' is a disjoint union of cycles and paths.

Definition 4.8. A cycle or path is called short if its length is at most $l = 24 \log \Delta$ and long otherwise.

Since we assumed that the starting graph is bipartite, all cycles are even. Therefore we would want the edge values to be raised and dropped alternately along the cycle. This is a perfect rounding. If the cycle is short, the following algorithm achieves this in $\mathcal{O}(\log \Delta)$.

1. Orient the cycle in one direction
2. if e goes from a node with colour 1 to colour 2:
3. $x_e := 2x_e$
4. else: $x_e := 0$

In a long cycle the first line is not easy to solve. Therefore, we contend ourselves with a common direction only on subpaths of length at least l on long cycles.

Definition 4.9. Consider a cycle with an orientation of each edge. A maximal directed path is a directed path that can not be extended since both neighbouring edges have inconsistent orientation.

Starting from a random orientation we can achieve long directed paths by determining the length of a subpath and compute the length of the subpath it points towards. By flipping the edges of the shorter path, we create a longer directed path.

1. orient e arbitrarily
2. for $i = 1, ..., \log(l)$:
3. compute the length of the path pointing in the opposite direction and flip the edges of the shorter path
4. endfor

Since we can now compute long subpaths of long cycles, we discuss rounding of long cycles in the next step.

1. compute long paths
2. if e is a boundary edge or goes from a node of colour 2 to colour 1: $x_e := 0$
3. else: $x_e := 2x_e$

Lemma 4.10. *Rounding long cycles leads to a loss of* $\leq \frac{3}{l} \sum_{e \in E} x_e$.

Obviously the same approach works for long paths.
The algorithm for short paths is a bit more complicated.

1. orient the graph with start node s and end note t
2. if e is first edge:

3. if s is tight (*not loose*): $x_e := 0$
4. else: $x_e := 2x_e$
5. else if e is the last edge:
6. if t is tight: $x_e := 0$
7. else: $x_e := 2x_e$
8. else if e is an even edge: $x_e := 0$
9. else: $x_e := 2x_e$

Lemma 4.11. *Rounding short paths results in a loss* $\leq 4f \sum_{e \in E} x_e$

Lemma 4.12. *In the rounding step going from an* f*-fractional matching to a* $2f$*-fractional matching the matching decreases by a factor of at most* $(1 - \frac{3}{l} - 4f)$ *and the rounding step takes* $\mathcal{O}(\Delta)$ *time.*

Lemma 4.13. *This results in a* $\frac{1}{16}$*-fractional matching which is a constant factor smaller than the initial* $\frac{1}{\Delta}$ *matching.*

Lemma 4.14. *A constant factor approximation 16-fractional matching can be computed in* $\mathcal{O}(\log^2 \Delta)$ *in a 2-coloured bipartite graph.*

Lemma 4.15. *A constant factor approximation matching can be computed in* $\mathcal{O}(\log^2 \Delta)$ *time in a 2-coloured bipartite graph with maximum degree* Δ*.*