Theory of Distributed Systems
Notes WS 18/19

Martin Hoefer

Organizational

Lectures:

- Email: mhoefer@cs.uni-frankfurt.de
- Office: 115, R.M.S. 11-15, just drop by whenever I'm there.
- Lectures Tue/Wed every week, mostly writing on the board
- Copying down my board writing is hard and not necessary, all text is in this document
- Only figures with examples are missing (I make them up on the fly in the lecture)
- Part I: Based to large extent on the book by Peleg, copies in CS library
- Part II: Will provide material on the wepage

Exercises/Exams:

- Weekly exercise sheets, published on Wednesday of week i, due Tuesday of week $i+1$,
- Due **14:15h** before the lecture or in letterbox on 1.OG. of R.M.S. 11-15
- Returned and discussed Friday of week $i+1$, 10:15h in the exercise session
- If you score x% of total number of exercise points, then
  If $50 \le x < 75$, one grading step bonus for exam (e.g., 2.0 to 1.7, or 3.7 to 3.3)
  If $75 \le x$, two grading steps bonus for exam (e.g., 2.0 to 1.3, or 3.3 to 2.7)
- Oral exams: March 7/8, 2019

Contents

1 Introductory Remarks **7**
1.1 Basics of Communication . . . 7

2 Modeling Assumptions **9**
2.1 Network Model . . . 9
2.2 Some Graph Terminology . . . 9
2.3 Computational Model . . . 10
2.4 Complexity Measures . . . 11
2.5 Three Representative Models . . . 12

3 Broadcasting Algorithms **13**
3.1 Flooding Algorithm . . . 13
3.2 Convergecast . . . 14
3.3 Bottom-Up Computation on Trees . . . 14
3.4 Pipelined Convergecast . . . 15
3.4.1 Upcast . . . 16
3.4.2 Applications of Upcast . . . 17

4 Dealing with Asynchrony **19**
4.1 BFS Trees and Asynchrony . . . 19
4.2 Synchronization . . . 20
4.2.1 Synchronizer α . . . 22
4.2.2 Synchronizer β . . . 22
4.2.3 A Hybrid: Synchronizer γ . . . 23

5 Symmetry Breaking **25**
5.1 Coloring . . . 25
5.1.1 Coloring Trees and Bounded-Degree Graphs . . . 26
5.1.2 Linial's Lower Bound . . . 28
5.2 Maximal Independent Set (MIS) . . . 30
5.2.1 Relations to Coloring . . . 31
5.2.2 A Fast Randomized Algorithm for MIS . . . 32
5.2.3 Applications . . . 38

6 Minimum Spanning Trees **41**
6.1 GHS Algorithm 42
6.2 Distributed Dual Greedy 43
6.3 GKP Algorithm 45
6.4 Lower Bound 48

7 Distance and Route Approximation **51**
7.1 Exact APSP in Unweighted Graphs 52
7.2 APSP with Relabeling in Unweighted Graphs 54
7.3 Weighted Graphs 56

8 Packet Routing **59**
8.1 Deterministic Oblivious Routing 60
8.2 Randomized Oblivious Routing 63
8.2.1 Path Selection for the Hypercube 63
8.2.2 Packet Scheduling for the Hypercube 65
8.2.3 Packet Routing in General Networks 69

9 Rumor Spreading **73**
9.1 Stars and Cliques 74
9.2 General Graph Topologies 76
9.2.1 Random Geometric Graphs $G(n, r)$ 78
9.2.2 Preferential Attachment Graphs 79

10 Wireless Networks **81**
10.1 Leaders, Initialization and the ALOHA Protocol 81
10.1.1 Initialization 82
10.1.2 Leader Election 84
10.2 Modeling Interference 88
10.3 Coloring 91
10.3.1 Acknowledgements 95
10.4 Maximum Independent Set 97
10.4.1 Online Learning 97
10.4.2 Learning in Bounded-Independence Graphs 99
10.4.3 Jamming-Resistant Learning 100

11 Blockchain and Consenus **105**
11.1 Cryptocurrencies, Trust, and Consensus 105
11.2 Fault Tolerance and Byzantine Generals 107
11.3 Proof-of-Work Consensus in Bitcoin 110

Chapter 1

Introductory Remarks

Distributed system?

- Multiple processing units working together, exchanging messages etc.
- Not here: Parallel systems, Collaborative/cooperative, joint infrastructure, joint memory access
- Instead: Coordination needed, communication networks, ad-hoc networks, sensor networks, wireless networks
- Issues: Locality, asynchrony, communication, information flow, etc.

Focus on fundamental system protocols for basic services:

- Communication: routing, broadcast, end-to-end communication
- Maintanance of control structures: spanning trees, topology update, leader election
- Resource control: load balancing, queueing, etc.

1.1 Basics of Communication

Shared memory vs. message passing:

- Shared memory mostly used in parallel computing (PRAM etc)
- Here: Message passing models
- Explicitly model communication, allows to study locality issues

Initially: Point-to-point communication (not broadcast networks, where messages can be delivered to many recipients).

[Pic: Graph]

Unique Issues in TDS

- Communication: Explicity modeled, incurs cost, limits to speed and capacity of information transmission
- Coordination: Partial information of input and computational results of other processors, partial information about environment (e.g. topology, IDs)
- Failures: Transient or permanent, links/processors, loss or corruption of messages, robustness issues, fault-tolerant algorithms

- Synchrony vs. asynchrony: We consider two model variants - both extreme, but good for understanding

Synchronous models: Each processor has local clock with pulses, message sent at pulse t from u to v must reach v before pulse t+1, global clock with machine cycle in three steps:

1. Perform local computation
2. Send messages to (some) neighbors
3. Receive messange from (some) neighbors

Asynchronous models: Event driven, no global clock, messages arrive in finite but unpredictable time. On the same link: First-in-first-out, but exact time points unknown. On different links: Arrival order is possibly different from sending order.

Gives rise to non-determinism, for same input, many different possibilities how and when messages arrive and what happens next (different *scenarios*)

Distributed Algorithm Distributed algorithm Π consists of one "protocol" Π_i for each processor i:

- Synchronous model: In round t, processor i does Y_t
- Asynchronous model: On event X, processor i does Y_X

Chapter 2

Modeling Assumptions

2.1 Network Model

- Undirected, connected graph $G = (V, E)$
- Sometimes edge weights $\omega(e) \geq 0$ for $e \in E$ (capacity, interference, length, etc)
- Each node has and knows its unique ID.
- IDs are numbers in $\{1, \ldots, n^c\}$ for some constant c, represented in $O(\log n)$ bits
- Vertex v has $\deg_G(v)$ ports, numbered $1, \ldots, \deg_G(v)$
- For each edge $e = \{u, v\}$, we assume there is a port for u, a port for v and a channel connecting them
- Each port has input buffers on both sides for incoming messages
- Only one message on the channel each point in time
- No collisions, one message in each of the two directions is allowed
- Message size is $O(\log n)$ bits, one message can send an ID.

[Pic: Node pair and edge with ports]

2.2 Some Graph Terminology

Recall graph-theoretic distances $dist_G(u, v)$ as studied in undergrad classes.

Diameter, Radius and Center

- Diameter of a graph $Diam(G) = \max_{u,v \in V} dist_G(u, v)$
- Radius of a node v is $Rad(v, G) = max_u \; dist_G(u, v)$
- Radius of a graph G is $Rad(G) = min_v \; Rad(v, G)$
- Center is a vertex v such that $Rad(G) = Rad(v, G)$

Note that $Rad(G) \leq Diam(G) \leq 2 \cdot Rad(G)$

[Pic: Diameter, Radius, Center]

Depth

- $Depth(v, T)$ of node v in tree T is the distance from root.

- $Depth(T) = \max_{v \in T}\{Depth(v, T)\}$

Neighborhoods

- $\Gamma(v)$ is set of all neighbors including v itself
- ρ-neighborhood $\Gamma_\rho(v)$ given by nodes of distance ρ to v.

  0 : v
  1 : v and all neighbors
  2 : v, all neighbors, all neighbors of neighbors
  3 : etc.

[Pic: Neighborhoods]

Tree Levels

- Level 0: root
- Level 1: direct children of root
- Level 2: direct children of direct children of root
- etc
- $Depth(v, T)$ of a node v in T is the tree level $L(v)$

[Pic: Example Tree Levels]

Tree Max-Levels $\hat{L}(v) = Depth(T_v)$, the depth of subtree T_v rooted at v,

$$\hat{L}(v) = \begin{cases} 0 & \text{if } v \text{ is leaf, and} \\ 1 + \max_{u \text{ child of } v} \hat{L}(u) & \text{otherwise.} \end{cases}$$

Tree Min-Levels

$$\bar{L}(v) = \begin{cases} 0 & \text{if } v \text{ is leaf, and} \\ 1 + \min_{u \text{ child of } v} \bar{L}(u) & \text{otherwise.} \end{cases}$$

[Pic: Example Max- and Min-Levels]

2.3 Computational Model

- Algorithm Π composed of n protocols $\Pi_1, \ldots, \Pi_n$ for processors/nodes $v_1, \ldots, v_n$
- Each node i has state set Q_i and is in some state q_i at any point in time
- Each link $e_i = \{u, v\}$ has state set $\bar{Q}_i$ and is in some state $\bar{q}_i$ at any point in time
- Link state $\bar{q}_i$ has a components $(q_{u \to v}, q_{v \to u})$ for each direction.
- Link state $q_{u \to v} \in \mathcal{M} \cup \{\lambda\}$, where $\mathcal{M}$ is set of possible messages, and λ means channel is empty.

- Execution of distributed algorithm governed by three types of events:

1. Compute(i)
2. Send(i, j, MSG) for some $MSG \in \mathcal{M}$
3. Deliver(i, j, MSG) for some $MSG \in \mathcal{M}$

- Events change configuration of states of nodes and links as defined by the algorithm

- Execution of algorithm Π on input I with network G is denoted $\eta^{\Pi}(G, I)$. It is a (possibly infinite) sequence of configurations C_i alternating with events ϕ_i:

$$\eta^{\Pi}(G, I) = (C_0, \phi_1, C_1, \phi_2, C_2, \phi_3, \ldots)$$

- By definition, a finite execution always ends with a configuration.
- Asynchronous model: Messages get delivered in finite time (i.e. Send(i, j, MSG) event triggers a corresponding Deliver(i, j, MSG) event after finite time)
- Synchronous model: Each round r for node i proceeds as follows:
  1. Compute(i)
  2. Messages of i for neighbors are sent out
  3. Messages for i sent out by neighbors in round r get delivered

2.4 Complexity Measures

Time complexity $\mathsf{Time}(\Pi, G)$ of algo Π and network G:
- Worst-case number of rounds/time units from start of execution of first processor to end of execution of last processor
- Asynchronous: assuming messages get delivered in at most one time unit, worst-case is worst-case input and scenarios

Message complexity $\mathsf{Message}(\Pi, G)$ of algo Π and network G:
- Overcoming locality might require a lot of communication
- Algorithm that needs many messages is bad: Overhead in the network, vulnerability to failures, etc.
- Message complexity: Worst-case number of basic messages sent by all nodes throughout the execution
- Basic message: $O(\log n)$ bits

Example Time vs. Message Complexity:
- Complete graph
- Π_A sends 1 message from node 0 to node 1
- Π_B sends simultaneously separate messages from node i to $i + 1$
- Π_C sends sequentially a message from node i to $i + 1$
- Π_D sends simultaneously separate messages from 0 to everyone else

Time: A,B,D are fast
Message: only A is fast.

Sometimes we incorporate a communcation cost e.g., for links with different lenghts.
Communication cost $\mathsf{Comm}(\Pi, G)$: cost of message weighted by link length $w(e)$

2.5 Three Representative Models

Extremely large variety of model variants. We make some assumptions:

1. No faults, no outage of nodes, no dynamic changes of network
2. Nodes have access to unique IDs, ID size is $O(\log n)$ bits
3. Computation is free, nodes are allowed to solve, e.g., NP-hard problems

Three traditional models to capture main limitations in distributed systems:

	CONGEST	LOCAL	ASYNC
Main Limitation	Communication Volume	Locality	Asynchrony
Message Size	$O(\log n)$ bits	unlimited	limited/unlimited
Local Computation	unlimited	unlimited	unlimited
Communication	synch. or asynch.	synchronous	asynchronous
Wakeup	simul. or asynch.	simultaneous	asynchronous

Message size in ASYNC often does not matter much.

Chapter 3

Broadcasting Algorithms

Broadcast: Source node r_0 has message, distribute the message to all n nodes

A network is **clean** if the nodes know nothing about the topology.

Lemma 1. *For any broadcast algorithm B and any graph $G = (V, E)$ in both synchronous and asynchronous models:*

- $\mathsf{Message}(B, G) \geq n - 1$
- $\mathsf{Message}(B, G) \geq |E|$ *if the network is clean,*
- $\mathsf{Time}(B, G) \geq Rad(r_0, G) = \Omega(Diam(G))$

Proof. Messages: Each node except r_0 needs to get the message.
Messages+Clean: If you don't try every edge, in worst-case you miss parts of the network.
Time: Message needs to reach farthest destination, which is $Rad(r_0, G)$ away. In worst-case, each message needs 1 time step to travel. □

Message complexity: Broadcast ⇔ Spanning Tree Construction (up to $O(n)$ messages)

- Any broadcast algorithm B can be used to build spanning tree T_B:
- Parent is the node you received the message from first.
- Hence, message complexity of tree construction is at most that of broadcast. Also, vice versa (plus $O(n)$ for actual broadcast)

3.1 Flooding Algorithm

Algorithm **Flood**: If node v has not seen the message and receives it in some time step from neighbors, it forwards it over all other neighbors in the next time step. If v has seen the message, it does nothing.

Lemma 2. $\mathsf{Message}(Flood, G) = \Theta(|E|)$ *and* $\mathsf{Time}(Flood, G) = \Theta(Rad(r_0, G)) = \Theta(Diam(G))$ *in both synchronous and asynchronous models*

Proof. Message: Every edge delivers message at least once and at most twice.
Time: By induction, at time step t, message has reached every node with distance at most t from r_0 (i.e., all of $\Gamma_t(r_0)$). In the asynchronous model, message might have reached nodes beyond $\Gamma_t(r_0)$. □

Lemma 3. *Let T be spanning tree constructed by Flood.*
Synchronous: T is a BFS tree with root r_0, so $Depth(T) = Rad(r_0, G)$.
Asynchronous: T can have $Depth(T) = n - 1$.

Proof. Synchronous: By induction, message reaches vertices at distance t from r_0 precisely in round t, hence depth in T is t.
Asynchronous: Message travels faster on some paths than others, so no guarantee on depth (except trivial ones) □

[Pic: Complete graph, star vs. path-tree depending on speed of messages]

3.2 Convergecast

How to realize broadcast has terminated? Reverse direction: Accumulate in r_0 a message (e.g., acknowledgement or echo) from all other nodes

Convergecast: Collect information bottom-up in a tree

Algorithm **Converge(Ack)**: If i is leaf, sends directly acknowledgement (ack) to its parent. In i is non-leaf, after all children sent ack to i, then i sends ack to its parent.

Ack inductively certifies: All of subtree T_i received the message.

[Pic: Schema Acks]

Lemma 4. *On a tree T we have that $\mathsf{Time}(Converge(Ack), T) = Depth(T)$ and $\mathsf{Message}(Converge(Ack), T) = n - 1$.*

We augment the Flood algorithm with Converge(Ack) (termed Flood&Echo)

Flood builds tree that is used for convergecast. Synchronous: Time complexity at most doubles, acks not much more demanding. Asynchronous: Can be $\Theta(n)$, even though broadcast finishes very quickly, when broadcast builds a very skewed tree.

Lemma 5.

1. *$\mathsf{Message}(Flood\&Echo, G) = O(|E|)$*
2. *$\mathsf{Time}(Flood\&Echo, G) = \begin{cases} O(Diam(G)) & \text{in the synchronous model} \\ O(n) & \text{in the asynchronous model} \end{cases}$*
3. *In both models, broadcast ensures message MSG reaches all vertices by time $Diam(G)$.*

3.3 Bottom-Up Computation on Trees

Each vertex v has a value x_v, compute a global function $f(x_1, \dots, x_n)$.

Definition 1. *f is a **semigroup function** if it has three properties*

1. *f well-defined for any subset of inputs, i.e., $f(\mathcal{Y})$ defined for any $\mathcal{Y} \subseteq \mathcal{X} = \{x_1, \dots, x_n\}$*
2. *f associative and commutative*
3. *representation of $f(\mathcal{X})$ "relatively short" compared to that of the inputs $x_1, \dots, x_n$.*

Convergecast can be used to compute $f(\mathcal{X})$.
Procedure $Converge(f, X)$:

- Node v waits to receive $f(T_{v_i})$ from children v_i.
- If v leaf or all children $v_1, \ldots, v_k$ of v delivered, v computes $f(x_v, f(T_{v_1}), \ldots, f(T_{v_k}))$
- Then v sends result to parent.

Result is correct due to associativity and commutativity.

$f(\mathcal{X})$ can be much larger than n. For every $\mathcal{Y} \subseteq \mathcal{X}$ we assume $f(\mathcal{Y})$ needs $O(p)$ bits for representation. Then we need $O(p/\log n)$ messages to send the result to the parent.

Lemma 6. *If representing $f(\mathcal{Y})$ needs at most $O(p)$ bits for any $\mathcal{Y} \subseteq \mathcal{X}$, then*

- *Message*$(Converge(f, X), T) = O(np/\log n)$
- *Time*$(Converge(f, X), T) = O(Depth(T) \cdot p/\log n)$

Globally-sensetive f: Result relies on every input value.

Lemma 7. *For any every tree T, computing any globally-sensitive f on T has message complexity $\Omega(n)$ and time complexity $\Omega(Depth(T))$.*

Examples [add Pics]:

- Addition of m-bit integers
  Any sum has at most $O(m + \log n)$ bits.
  Message: $O(nm)$, Time: $O(Depth(T) \cdot m)$.

- Maximum of m-bit integers:
  Any maximum has at most m bits.
  Message: $O(nm/\log n)$, Time: $O(Depth(T) \cdot m/\log n)$

- Logical conditions
  Each node v has a predicate $Pred(v)$ that is either true or false
  Logical combinations with either $\wedge$ or $\vee$ are associative and commutative.
  In this way, the source can be informed if $\bigvee_i Pred(v_i)$ (at least one) or $\bigwedge_i Pred(v_i)$ (all) of the predicates $Pred(v_i)$ in the network hold true. Converge(Ack) above is equivalent to checking if
  $$\bigwedge_i Pred(v_i)$$
  is true, where $Pred(v_i) =$ "v_i received the original message".

3.4 Pipelined Convergecast

Suppose each node v of tree T has a k-dimensional vector $(x_{1,v}, \ldots, x_{k,v})$, where every $x_{i,v}$ is a $\log(n)$-bit value. The goal is to compute k semigroup functions for each position, e.g., compute the vector $(\max_u x_{1,u}, \max_u x_{2,u}, \ldots, \max_u x_{k,u})$. Trivial solution: Perform k convergecast operations. Needs time $O(k \cdot Depth(T))$

Better solution in synchronous model: Each leaf starts convergecast for first position in round 1, second convergecast for position 2 in round 2, third one in round 3, etc.

Information rises up in the tree like a pipeline. Interior nodes v receive partial results for subtrees for the second, third, etc. position over time. Inductively, all maxima of subtrees will first be collected for position 1 in round $\hat{L}_v$, and then for position 2 in round $\hat{L}_v + 1$, etc. The maximum in subtree T_v for position i will correctly be reported to the parent of v in round $\hat{L}_v + i$.

Lemma 8. *Synchronous: Computing k global semigroup functions on a tree T can be done in time $O(Depth(T) + k)$.*

Asynchronous: Use the fact that messages over the same link are processed in FIFO order.

Lemma 9. *Asynchronous: Computing k global semigroup functions on a tree T can also be done in time $O(Depth(T) + k)$.*

3.4.1 Upcast

We concentrate on message complexity in the synchronous CONGEST model.
Upcast: Vertices have a total of m messages $A = \{\mu_1, \ldots, \mu_m\}$, a message may be present at mulitple vertices. Collect one copy of every message at the root.

[Pic: Example Upcast]

Obvious lower bounds for time complexity of $\Omega(Depth(T))$ on a given tree T, and $\Omega(m)$ on any tree.

Algorithm **Upcast**: In each round, forward to parent an arbitrary message that has not been upcast so far.

Lemma 10. *Consider vertex v and integer t. Suppose that for every $1 \leq i \leq k$, at the end of round $t + i$, v stored at least i messages. Then at the end of round $t + k + 1$, v's parent w has received from v at least k messages.*

Proof. By induction on t.
Init: Round $t + 1$, v has at least one message. Either transmitted to parent earlier, or v will transmit in round $t + 1$. By round $t + 2$ at the latest, parent has received at least one message.
Step: Suppose true until round $t + i$. By inductive hypothesis, parent received at least $i - 1$ messages in round $t + i$. If it received more than i messages, done. Otherwise, parent received exactly $i - 1$ messages. v has at least i messages in round $t + i$, so one is transmitted in round $t + i$, reaches parent in round $t + i + 1$, done. □

M_v – set of messages initially stored anywhere in the subtree T_v.

Lemma 11. *For every $1 \leq i \leq |M_v|$, at the end of round $\hat{L}(v) + i - 1$, at least i messages are stored at v.*

Proof. We fix i and prove it by induction on $\hat{L}(v)$.

Init: For leaf with $\hat{L}(v) = 0$ the lemma is true – all of M_v stored at v from the start.

Step: Suppose claim holds for all nodes w with $\hat{L}(w) = \ell - 1$. Consider v with $\hat{L}(v) = \ell$.

- Consider child w_j of v, let $\ell_j = \hat{L}(w_j)$, $m_j = |M_{w_j}|$ and $\gamma_j = \min(i, m_j)$
- Note $\ell_j \le \ell - 1$. Inductive hypothesis: For all $1 \le i' \le m_j$, at the end of round $\ell_j + i' - 1$, w_j has at least i' messages. Apply previous lemma ($t = (\ell_j - 1)$ and $k = \gamma_j$): At the end of round $(\ell_j - 1) + \gamma_j + 1$, v already received from w_j at least γ_j messages.
- Hence, if v has child with $m_j \ge i$, then v received $m_j \ge i$ messages. Lemma is shown.
- Otherwise, $m_j < i$ (so $\gamma_j = m_j$) for all children. Then, by above arguments, at round $\ell + i - 1$, v received all m_j messages from every child w_j. Thus v stores all of the $|M_v| \ge i$ items.

□

Previous lemma: Root has m messages at the end of round $Depth(T) + m - 1$.

Corollary 1. *Upcast of m messages on a tree T can be done in time at most $m + Depth(T)$.*

3.4.2 Applications of Upcast

Information gathering and dissemination Collect all messages and broadcast them to all nodes in the network

- Upcast of items to the root takes time $Depth(T) + m$
- Downcast all messages from the root in pipelined fashion
- Takes time $Depth(T) + m$.

Route-Disjoint Matching Given rooted tree T and a set W of $2k$ marked vertices in T ($k \le \lfloor n/2 \rfloor$). Wanted: Perfect matching for W such that all matched pairs have pairwise edge-disjoint routes connecting them in T. Each node $w \in W$ should know the ID of its matched partner. Can be found by suitable algorithm (Exercise).

[Pic: Example Route-Disjoint]

Lemma 12. *For every tree T and every set W, there exists a route-disjoint matching. The matching can be found by a distributed algorithm on T in time $O(Depth(T))$.*

Token Distribution n tokens distributed on the nodes ($O(\log n)$ bits each). Each node at most K tokens. Redistribute tokens such that every node has exactly one token.

[Pic: Example Tokens]

- Each token can be sent in 1 message
- Total cost of redistribution = Sum of distances traversed by tokens
- Use convergecast to determine:
    1. s_u: number of tokens in subtree T_u.
    2. n_u: number of vertices in T_u

3. $p_u = s_u - n_u$: number of tokens that must leave T_u.

- Total number of messages necessary is $P = \sum_{u \neq r_0} |p_u|$.
- Can be achieved by suitable distributed algorithm. (Exercise)

Lemma 13. *There is a distributed algorithm for token distribution using an optimal number of P messages and $O(n)$ time, after preprocessing with $O(Depth(T))$ time and $O(n)$ messages.*

Chapter 4

Dealing with Asynchrony

4.1 BFS Trees and Asynchrony

Synchronous model: Flood builds a BFS tree. Here: BFS trees in asynchronous CONGEST model using repeated acks with Dijkstra's algorithm.

Algorithm 1: D-BFS

Start phase $p = 0$ with T composed of root r_0
repeat
  r_0 broadcasts "start p" in T
  if leaf of T gets "start p" **then** sends "join $p+1$" to all quiet neighbors (that u has received no msg from before)
  if *$v \notin T$ gets "join $p+1$"* **then**
    Picks one parent w from the senders, replies "ACK parent" to w
    Replies "ACK no parent" to all other senders, becomes leaf of T at level $p+1$
  if $v \in T$ gets "join $p+1$" **then** replies "NACK" to all such messages.
  Leaves of T at level p collect answers from neighbors
  Then every leaf v starts convergecast, indicating if new child of v was found.
  When convergecast ends at r_0, r_0 increments phase.
until *no new node discovered*

[Pic: Schema]

Correctness: Simple induction.

Theorem 1. *For D-BFS in the asynchronous* CONGEST *model*
- $\mathsf{Time}(D\text{-}BFS, G) = O(Diam(G)^2)$
- $\mathsf{Message}(D\text{-}BFS, G) = O(|E| + n \cdot Diam(G))$

Proof. In phase p:
- Broadcast and convergecast in T: Total time at most $2p$
- Exploration of new neighbors: Time at most 2
- Broadcast and convergecast need $O(n)$ messages

Every edge: Exactly one "join x" message (for some number x), and exactly one ACK/NACK message in the whole algorithm. This gives

- $\mathsf{Time}(\text{D-BFS}, G) = \sum_p 2p + 2 = O(Diam(G)^2)$
- $\mathsf{Message}(\text{D-BFS}, G) = 2|E| + \sum_p O(n) = O(|E| + n \cdot Diam(G))$

□

Better idea: Bellman-Ford, very important in the Internet, basic version of border gateway protocol (BGP)

Algorithm 2: BF-BFS

```
Root sets L(r_0) ← 0, all other nodes L(v) ← ∞
r_0 sends "1" message to all neighbors
on node v gets message "d" with d < L(v) from neighbor w do
    parent(v) ← w, L(v) ← d
    Send "d + 1" to all neighbors except w
```

Theorem 2. *For BF-BFS in the asynchronous* CONGEST *model*

- $\mathit{Time}(\text{BF-BFS}, G) = O(Diam(G))$
- $\mathit{Message}(\text{BF-BFS}, G) = O(n \cdot |E|)$

Proof. Time complexity by induction: Node at distance d received message "d" by time d.

Init: $\Gamma_1(r_0)$ receives "1" by time 1.

Step: v at distance d has neighbor w at distance $d-1$. Induction hypothesis: w gets "$d-1$" by time $d-1$. Then v gets "d" by time d.

Message complexity: Node can reduce distance at most $n-1$ times, every time sends a message to all neighbors. □

There is an algorithm B that yields an optimal trade-off:

- $\mathsf{Time}(B, G) = O(Diam(G) \cdot \log^3 n)$
- $\mathsf{Message}(B, G) = O(|E| + n \log^3 n)$

4.2 Synchronization

Given: Algorithm Π_S for some synchronous model
Goal: General "synchronizer" ν, such that $\Pi_A = \nu(\Pi_S)$ is algorithm for corresponding asynchronous model

Both components Π_S and ν have their own local variables, environment, etc.

Approach: **Pulse Generator**

- Pulse - essentially a coordinating tick of a clock
- Each processor maintains internal variable of current pulse
- In pulse p, processor performs exactly the actions in round p specified in the synchronous algorithm Π_S (i.e., (i) compute, (ii) send messages of round p, (iii) receive messages of round p).

- Maintain coordination of **neighboring** nodes. Globally, nodes might be in very different pulses at the same time

Definitions:

- **Original message**: Message sent due to Π_S.
- v **at pulse** p: Internal pulse variable of v is set to p
- **Pulse compatibility**: v at pulse p sends original MSG to neighbor w. Then MSG must be received by w at pulse p.
- **Similar execution**: Π_S and $\Pi_A = \nu(\Pi_S)$ have similar executions if
  1. start of pulse p in Π_A same values are stored at every processor as in the start of round p in Π_S,
  2. original messages sent/received during pulse p are exactly the ones sent/received in round p
  3. at the end of execution same final output at every processor
- **Correct Simulation**: Π_A simulates Π_S if for every input, executions are similar. ν is correct if for all synchronous protocols Π_S the algorithm $\nu(\Pi_S)$ simulates Π_S.

Lemma 14. *ν satisfies pulse compatibility $\Rightarrow$ ν is correct.*

v must wait for all original messages sent by neighbors during pulse $p-1$ before generating pulse p. Messages sent from neighbors in later pulses $p' > p$ must be used by v only by the time v itself advances to pulse $p'+1$.

- **Readiness Rule**: v is **ready for pulse** p if it received all messages sent by neighbors during pulse $p-1$. v is allowed to generate pulse p once finished with required original computation for pulse $p-1$ and ready for pulse p.
- **Delay Rule**: v receives in pulse p a message MSG from a neighbor in a later pulse $p' > p \Rightarrow v$ stores MSG in a buffer, consumes it only when it advances to pulse $p'+1$.

To satisfy delay rule, attach local pulse number to original message.

Lemma 15. *ν satisfies readiness and delay rules $\Rightarrow$ ν satisfies pulse compatibility and is correct.*

Readiness easy if ν makes every processor send a message to every neighbor at every pulse. What if this is not the case? We might wait forever for a message that was never sent. Also, do we really need the buffer for "future" messages?
Problem 1: Wait forever for a message that might never been sent.
Problem 2: Limited buffer to store messages for future pulses?

Two or Three Phase Implementation:

- **Phase A**: Send original messages. Every receiving neighbor is required to return ack.
- v is **safe w.r.t. pulse** p if all messages sent during pulse p arrived.
- Obviously, if all neighbors w of v are safe, then v is ready for pulse $p+1$.
- **Phase B**: Apply procedure to let each processor know that all neighbors are safe w.r.t. pulse p.

Phases A+B take care of Problem 1. For Problem 2:

- v is **enabled for pulse** p once all neighbors w are ready for pulse p.
- **Enabling Rule**: v starts Phase A of pulse p only when it is enabled for pulse p.
- **Phase C**: Apply procedure to let each processor know that all neighbors are ready for pulse p.

Lemma 16. *ν satisfies readiness and enabling rules $\Rightarrow$ ν satisfies pulse compatibility and is correct.*

Notation and Complexity Measures

- Initialization: $\mathsf{Time}_{init}(\nu)$, $\mathsf{Message}_{init}(\nu)$
- v generates pulse p at some global time $t(v,p)$, $t_{\max}(p) = \max_v t(v,p)$
- $\mathsf{Time}_{pulse}(\nu) = \max_{p\geq 0} t_{max}(p+1) - t_{\max}(p)$
- $\mathsf{Message}_{pulse}(\nu)$: Number of messages for coordination during a single pulse

Lemma 17.

1. $\mathsf{Message}(\Pi_A) \leq \mathsf{Message}_{init}(\nu) + \mathsf{Message}(\Pi_S) + \mathsf{Time}(\Pi_S) \cdot \mathsf{Message}_{pulse}(\nu)$
2. $\mathsf{Time}(\Pi_A) \leq \mathsf{Time}_{init}(\nu) + \mathsf{Time}(\Pi_S) \cdot \mathsf{Time}_{pulse}(\nu)$

Phase A does not contribute to the overhead.
Remains to show: Efficiently implement Phases B and C, nodes must be informed when all neighbors satisfy a binary property (safe, ready).

4.2.1 Synchronizer α

When node v is safe (or ready), it sends this fact to every neighbor.
Straightforward implementation of readiness and enabling rules.

Initialization:

- Broadcast init message from source r_0 using Flood (no echo).
- $\mathsf{Message}_{init}(\alpha) = O(|E|)$
- $\mathsf{Time}_{init}(\alpha) = O(Diam(G))$

Each pulse:

- $\mathsf{Message}_{pulse}(\alpha) = O(|E|)$
- $\mathsf{Time}_{pulse}(\alpha) = O(1)$
- The node that is latest to complete the pulse just suffers a constant-factor overhead in time due to the sending of coordination messages in Phases B and C.

Easy, very good time complexity, rather large message complexity.

4.2.2 Synchronizer β

Assume nodes know spanning tree T rooted in r_0. We collect all safety information in Phase B using a convergecast in T:

- If v learns that itself and all descendants are safe, it sends this info to $parent(v)$.
- Once root learns all nodes are safe, it broadcasts this info in the tree. Then all nodes start a new pulse (or proceed to Phase C).

Another straightforward implementation of readiness and enabling rules.

Initialization:
- Build BFS tree from source r_0, using, e.g., Bellman-Ford.
- $\mathsf{Message}_{init}(\beta) = O(n|E|)$
- $\mathsf{Time}_{init}(\beta) = O(Diam(G))$

Each pulse:
- $\mathsf{Message}_{pulse}(\beta) = O(n)$
- $\mathsf{Time}_{pulse}(\beta) = O(Diam(G))$
- The node that is latest to complete the pulse just suffers a constant-factor overhead in time due to the sending of coordination messages in Phases B and C.

Very good message complexity, rather bad time complexity. Good for low-diameter networks.

4.2.3 A Hybrid: Synchronizer γ

In the initialization, we assume some more structure is established
- Node set is partitioned into clusters.
- Each cluster C is connected and organized into a rooted BFS tree.
- Root node is called the **leader** of the cluster.
- Clusters C_1, C_2 are **neighboring** if there is an edge between them.
- For every pair of neighboring clusters, we pick a single one of the edges connecting them as their **intercluster edge**.

[Pic: Example]

Phase B works as follows:
- Safety for nodes within each cluster is communicated to the leader via convergecast.
- When all nodes are safe, this info is broadcast from the leader to all nodes in the cluster and via intercluster edges to all neighboring clusters.
- Safety information from neighboring clusters is communicated to the cluster leader via convergecast.
- Then the leader sends a broadcast to start the next pulse (or start the same procedure for Phase C and readiness).

Synchronizer γ applies synchronizer β inside each cluster, and synchronizer α between clusters. Optimal trade-off between time and message complexities of α and β.

Notation:
- E_C is the set of all intercluster edges
- T_C is the tree used in cluster C
- $k = \max_C Depth(T_C)$

Inititalization is more complicated. Here only complexity for each pulse:
- $\mathsf{Message}_{pulse}(\gamma) = O(|E_C| + n)$
- $\mathsf{Time}_{pulse}(\gamma) = O(k)$

It is possible to achieve $|E_C| \in O(n^{1+1/k})$, which is an optimal trade-off between cluster radius and number of intercluster edges. For $k = \lceil \log n \rceil$ message complexity becomes $O(n)$ (same as synchronizer β) but time complexity $O(\log n)$ (instead of $O(Diam(G))$ for β).

Chapter 5

Symmetry Breaking

Fundamental problem: Leader election. Initially, all nodes might be symmetric and in the same state, every node thinks it is the leader (or not the leader). This and similar problems in distributed environments require techniques for **symmetry breaking**.

Leader election is global problem. We consider local analogs:

Vertex Coloring Assign a rank (or color) to each node s.t. in each neighborhood every rank appears at most once

Maximal Independent Set (MIS) Find a set of leaders s.t. in the neighborhood of each node there is exactly one leader

5.1 Coloring

Distributed vertex coloring

- Palette of m possible colors (priorities, channels, access rights, resources, etc)
- Assign each vertex a single color s.t. neighboring vertices have different colors
- How many colors do we need? Nodes have unique IDs, with n colors it's possible
- Use as few colors as possible

Minimum number of colors needed for graph G is **chromatic number** $\chi(G)$. Chromatic number can be NP-hard to compute, even hard to approximate within non-trivial factors

We consider simultaneous wakeup, synchronous LOCAL model. An obvious approach to symmetry breaking is to use the unique IDs. Towards this end, consider the Reduce algorithm.

Lemma 18. *Let $\Delta = \max_{v \in V} \deg_G(v)$ be the maximum degree. Reduce terminates in at most n rounds and uses at most $\Delta + 1$ colors.*

Proof. The proof is simple:

- No two neighbors choose simultaneously. Hence, coloring is feasible.

Algorithm 3: Greedy procedure **Reduce** for each node v

send ID to all neighbors
while *exists uncolored neighbor with higher ID* **do**
  send "undecided" to all neighbors
choose smallest admissible free color
send color choice to all neighbors

- Each round (at least) the uncolored node with highest ID gets colored. We need at most n rounds.
- Neighbors of v can only block $\deg_G(v)$ many colors, so v always finds a permissible color within the first $\deg_G(v) + 1$ colors.
- Algorithm uses at most $\Delta + 1$ colors, where $\Delta = \max_{v \in V} \deg_G(v)$.

□

5.1.1 Coloring Trees and Bounded-Degree Graphs

Each tree T is a bipartite graph, so $\chi(T) \leq 2$.
Obvious algo A: Broadcast with alternating 0/1 messages. Root colors itself $c_{r_0} \leftarrow 0$, sends "1" to all children. Each $v \in V$ upon receiving $x \in \{0, 1\}$ colors itself $c_v \leftarrow x$, sends $1 - x$ to all children. $\mathsf{Time}(A, T) = \mathit{Depth}(T)$, too slow if tree is deep.

Algorithm 4: Amazingly fast **6-Color** algorithm

$c_v \leftarrow ID(v)$ for all v
Send own color c_v to all children
repeat
  Receive color c_p from parent
  Interpret c_p and c_v as bit strings, let ℓ be number of bits of c_v
  Let i be index of smallest bit, where c_v and c_p differ (if $v = r_0$ set i to 0)
  New label: i (as bitstring) followed by i^{th} bit of c_v
  Send c_v to all children
until c_v *still has* ℓ *bits*

Example: (last bit has index 0, second-to-last index 1, etc.)

Grand-Parent:	0010110000	$\rightarrow$	10010	$\rightarrow$	...
Parent:	1010010000	$\rightarrow$	01010	$\rightarrow$	111
Child:	0110010000	$\rightarrow$	10001	$\rightarrow$	001

i-times application of the logarithm: $\log_2(\log_2(\ldots(\log_2(\log_2(n)))\ldots)) = \log_2^{(i)}(n)$.
$\log^* n$ is the smallest integer i such that $\log_2^{(i)}(n) \leq 2$.

Theorem 3. *6-Color legally colors the tree with at most 6 colors in* $\mathsf{Time}(\text{6-Color}, T) = O(\log^* n)$.

Proof. Legal coloring: Consider neighboring v, w, let $w = parent(v)$.

- They pick different indices in step 6: Color labels differ afterwards.
- They pick same index: Rule in step 7 ensures the color number differs in last bit.

Running time: Let n_i be the maximum number of bits needed to represent a color after i-th iteration.

- Initially, colors are IDs, so $n_0 = O(\log n)$.
- Then $n_{i+1} = \lceil \log_2(n_i) \rceil + 1$, due to assignment in step 7.
- Some numeric facts of the series $(n_i)_{i=0,1,2,\ldots}$:
  - $n_{i+1} < n_i$ as long as $n_i \geq 4$.
  - $n_i \leq \lceil \log_2^{(i)} n_0 \rceil + 2$ for every i with $\log^{(i)} n_0 \geq 2$.
- Number of bits for colors shrinks logarithmically to 3 in at most $O(\log^* n)$ rounds.
- Then: 3 choices for a bit index in step 6 and 2 choices for the appended bit in step 7.
- Hence, in the end at most $3 \cdot 2 = 6$ colors □

Algorithm 5: Subsequent Refinement: **Six2Three**

```
1 for x ∈ {3, 4, 5} do
2     Every node v ≠ r_0 simulatenously adopts color of its parent
3     Root uses new color in {0, 1, 2}, different from current one
4     Every node v with c_v = x picks a legal color in {0, 1, 2}
```

[Pic: Example]

Theorem 4. *Algorithms 6-Color and Six2Three legally color any tree with at most 3 colors in time* $O(\log^* n)$.

Proof. Six2Three needs only $O(1)$ rounds. The Shift-Down in step 2 keeps the coloring legal, since each parent and child had different colors before. After Step 2 all siblings use same color. Then for each node v one color for all children, another one for parent, so v has a free legal color in $\{0, 1, 2\}$. Thus, colors $\{3, 4, 5\}$ can be removed. □

What about 2 colors? Intuition: Correct coloring of a path from the root yields information about distance being even or odd. In the worst case, this information needs time $\Omega(n)$ to propagate!

Beyond trees: Colorings with $\Delta + 1$ colors in graphs G with constant max-degree Δ.

Theorem 5. *There exists a deterministic distributed algorithm for coloring arbitrary bounded-degree graphs with* $\Delta + 1$ *colors in time* $O(\log^* n)$.

Proof. For each $w \in \Gamma(v)$, execute Steps 6 and 7 in 6-Color separately using color c_w (instead of c_p). Concatenate all the resulting bitstrings into a new label.[1]

[1] The previous algorithm for trees works also in the CONGEST model, since the initial IDs in the first round constitute the largest messages we send. Here, in the first round, the new label might be longer than a single ID and a single message size in the CONGEST model. Hence, for bounded-degree graphs we explicitly use the assumption of unbounded message sizes in the LOCAL model.

For the formal proof we need to show the following steps:

1. Coloring is legal in every round
2. Set up recursion for n_i (bit-length of color number)
3. n_i stops shrinking when color number is a $O(\Delta \log \Delta)$-bit label
4. Constant Δ: n_i stops shrinking after time $O(\log^* n)$

In the end, $O(\Delta \log \Delta)$-bit labels imply $2^{O(\Delta \log \Delta)}$ colors. Interpret color number as fake-ID and run **Reduce** to (re-)color every node with at most $\Delta + 1$ colors. Reduce produces legal coloring – since fake-IDs in each neighborhood are unique, no two neighboring nodes are (re-)colored simultaneously. In each step, at least the nodes with highest remaining fake-ID get (re-)colored. Thus, **Reduce** needs time $2^{O(\Delta \log \Delta)} = O(1)$ (since $\Delta = O(1)$).

Steps 1.-3. are left as an exercise, step 4. follows from numeric facts, similar to the ones above. □

For general (unbounded-degree) graphs, one can beat the running time of Reduce. We mention the result without proof.

Theorem 6. *There exists a deterministic distributed algorithm for coloring arbitrary graphs with $\Delta + 1$ colors in time $O(\Delta \log n)$.*

5.1.2 Linial's Lower Bound

Goal: Lower bound for 3-coloring rooted trees in the LOCAL model
Consider a rooted tree as a path rooted at one of the endpoints. We assume that root has smallest ID, and the IDs are strictly increasing along the path. We call this instance a *monotone* path.

Algorithms in the LOCAL model:

- Unlimited computation, unlimited communication
- Consider any algorithm in the LOCAL model that terminates in $t + 1$ rounds.
- Nodes do not need to send more than the unknown information about their input, edge structure, and IDs.
- All results of local computations in earlier rounds that are able to reach v by round t can only depend on information that v also receives by round t.
- Hence, all computation can be done by v itself.
- Wlog every algorithm in the LOCAL model that terminates in $t + 1$ rounds:
    1. Learn about edges, IDs and inputs in t-neighborhood $\Gamma_t(v)$ in the first t rounds.
    2. In round $t + 1$ compute some function $f(\Gamma_t(v))$.

LOCAL model captures locality restriction for computation in a mathematically rigorous way.

Every deterministic 3-coloring algorithm that terminates in t rounds on the directed path:

- In rounds $1, \ldots, t-1$: Learn (1) IDs of all $2t - 2$ neighbors of distance at most $t - 1$, and (2) their order along the path
- In round t: Compute $c_v \in \{0, 1, 2\}$ based on the ordered vector of $2t - 1$ IDs in $\Gamma_t(v)$.

B is a k**-ary** q**-coloring function** if for all $1 \leq a_1 < a_2 < \ldots < a_k < a_{k+1} \leq n$ we have

P1: $B(a_1, \ldots, a_k) \in \{0, 1, 2, \ldots, q-1\}$
P2: $B(a_1, \ldots, a_k) \neq B(a_2, \ldots, a_{k+1})$

Lemma 19. *A deterministic distributed 3-coloring algorithm for an n-node monotone path in $t < \frac{\log^* n}{2} - 1$ rounds computes a k-ary 3-coloring function with $k = 2t - 1 < \log^* n - 3$.*

Proof. Given $1 \leq a_1 < \ldots < a_{k+1} \leq n$, construct imaginary monotone path with IDs a_i along the path. Run coloring algorithm on nodes a_t and a_{t+1}.

[Pic: Example]

- In the first $t-1$ rounds, algorithm collects vector of $2t-1$ IDs in each neighborhood.
- Round t: Algorithm computes color $f_A(a_1, \ldots, a_k)$ at node a_t and color $f_A(a_2, \ldots, a_{k+1})$ at node a_{t+1}
- Algorithm correct, so each one a feasible color in $\{0, 1, 2\} \Rightarrow$ P1 holds for f_A
- Algorithm correct, so legal coloring $\Rightarrow$ P2 holds for f_A

Hence, f_A is a k-ary 3-coloring function. □

These functions allow for an interesting tradeoff between size of input and number of colors:

Lemma 20. *If there is a k-ary q-coloring function B, then there is a $(k-1)$-ary 2^q-coloring function B'.*

Proof. Given an input $1 \leq a_1 < \ldots < a_{k-1} \leq n$ for B', we define B' based on k-ary q-coloring function B:

- First, let $B'(a_1, \ldots, a_{k-1})$ be the **subset of colors** that would be used by B when adding to the input a new largest number. More formally,

$$B'(a_1, \ldots, a_{k-1}) = \{i \in \{0, 1, \ldots, q\} \mid \exists a_k \text{ with } n \geq a_k > a_{k-1} \text{ and } B(a_1, \ldots, a_k) = i\}.$$

To show the two properties we observe:

- $B'(a_1, \ldots, a_{k-1})$ is a subset of $\{0, 1, \ldots, q-1\}$. We can **turn the subset into a bit-string**, where bit b_i is 1 iff i is in the subset and 0 otherwise, for every $i = 0, 1, \ldots, q-1$
- Then, $B'(a_1, \ldots, a_{k-1}) \in \{0, 1, \ldots, 2^q - 1\}$, so P1 holds for B'.
- Assume for contradiction that P2 is violated:
  There are $1 \leq a_1 < \ldots < a_{k-1} < a_k \leq n$ with $B'(a_1, \ldots, a_{k-1}) = B'(a_2, \ldots, a_k)$.
- Let $q^* = B(a_1, \ldots, a_k)$. By definition $q^* \in B'(a_1, \ldots, a_{k-1})$.
- By assumption that P2 is violated: $q^* \in B'(a_2, \ldots, a_k)$
- Hence, there must be $n \geq a_{k+1} > a_k$ with $B(a_2, \ldots, a_k, a_{k+1}) = q^*$.
- Then $B(a_1, \ldots, a_k) = q^* = B(a_2, \ldots, a_{k+1})$, so B would violate P2.
- Contradiction, so P2 holds for B'. □

Hence, if there is a k-ary 3-coloring function, then there are also

- $(k-1)$-ary 2^3-coloring function
- $(k-2)$-ary 2^{2^3}-coloring function
- $(k-3)$-ary $2^{2^{2^3}}$-coloring function
- ...
- 1-ary q_1-coloring function, where $q_1 = 2^{2^{\cdot^{\cdot^{\cdot^{2^3}}}}}$

q_1 results from applying 2^x to 3 at most $k < \log^* n - 3$ times. Suppose $q_1 = n$, reverse the process: Apply $\log_2 x$ at most $k < \log^* n - 3$ times to n. By definition of $\log^*$ gives a number > 3. Thus, it must be $q_1 < n$.

Hence:
A deterministic distributed algorithm for 3-coloring an n-node monotone path in less than $\frac{\log^* n}{2} - 1$ rounds
$\Rightarrow$ k-ary 3-coloring function with $k < \log^* n - 3$
$\Rightarrow$ 1-ary q_1-coloring function with $q_1 < n$.
But:

Lemma 21. *There is no 1-ary q_1-coloring function with $q_1 < n$.*

Proof. Needs that $B(a_1) \neq B(a_2)$ for all $1 \leq a_1 < a_2 \leq n$, i.e., B must assign each of the n possible numbers $a_i \in \{1, 2, \ldots, n\}$ a value from $\{0, 1, \ldots, q_1 - 1\}$ such that all are pairwise distinct. Pigeonhole principle implies $q_1 \geq n$. □

This implies:

Theorem 7. *Any deterministic distributed algorithm for 3-coloring an n-node monotone path needs at least $\Omega(\log^* n)$ rounds.*

5.2 Maximal Independent Set (MIS)

Consider synchronous LOCAL model with simultaneous wakeup.

Centralized MIS is trivial. Greedy algo:
Pick arbitrary node v, include in MIS, remove all nodes in $\Gamma_1(v)$, repeat.
Distributed implementation MIS-Rank picks node with largest ID.

Algorithm 6: MIS-Rank

Every node tells her ID to all neighbors, all nodes set $b_v \leftarrow \perp$
repeat
  if *all neighbors w with larger ID decided $b_w = 0$* **then**
    Set $b_v \leftarrow 1$, send "Decide-1" to all neighbors
  on *getting "Decide-1" from neighbor w* **do**
    Set $b_v \leftarrow 0$, send "Decide-0" to all neighbors
until $b_v \in \{0, 1\}$

Time(MIS-Rank) $= O(n)$ and $\mathsf{Message}$(MIS-Rank) $= O(|E|)$.

IDs are essential for deterministic algorithms. Network is **anonymous** if every node sees exactly the same input and the same ID.

Lemma 22. *There is no deterministic algorithm for computing an MIS on an anonymous ring network with simultanous wakeup.*

Proof. Exercise. □

5.2.1 Relations to Coloring

Using a m-coloring we can also solve MIS.

Algorithm 7: Color2MIS

Run coloring algorithm for G, let $0, \ldots, m-1$ be used colors
for *round* $i = 1, \ldots, m$ **do**
  if *original color of* v *is* $(i-1)$ **then**
    if *no node in* $\Gamma_1(v)$ *is in MIS* **then**
      Set $b_v \leftarrow 1$, send "Decide-1" to all neighbors
    else set $b_v \leftarrow 0$

Lemma 23. *Given a coloring algorithm that colors a graph with $f(G)$ colors in time $T(G)$, Color2MIS constructs a feasible MIS in time $T(G) + f(G)$.*

Proof. IS: Each color class is an independent set. Thus, in each round, set of nodes joining IS is independent. Nodes joining have no edges to previously added nodes.

Maximal: Suppose there is neighborhood $\Gamma_1(v)$ with no node from the final IS. v has some color i_v, at round $i_v - 1$ node v would have entered IS $\rightarrow$ contradiction. □

Corollary 2. *There exists a deterministic distributed MIS algorithm for trees and bounded-degree graphs with time complexity $O(\log^* n)$.*

This bound is best-possible for algorithms that compute a coloring first, but also for any deterministic MIS algorithm (based on coloring or not).

Theorem 8. *Any deterministic distributed MIS algorithm for the n-vertex path or the n-vertex ring requires $\Omega(\log^* n)$ rounds.*

Proof. Turn MIS into a 3-coloring in 3 rounds. Then result follows with Theorem 7.

Simplifying assumptions:
Tree: Root is leftmost node, parent is left neighbor, child is right neighbor.
Ring: Edges oriented consistently, every node v knows who is the "left" and "right" neighbor, every node v is the right neighbor of its' left neighbor.

Algorithm 8: MIS2ThreeColor

Every $v \in MIS$ colors itself 0 and sends "1" to left neighbor in round 1
if $w \notin MIS$ gets message "1" in round 2 **then** w colors itself 1 **else** w colors itself 2

Correctness:

- Walk along the ring in the, say, "right" direction
- Consider node v_i in the MIS
- At most the next 2 nodes to the right (v_{i+1} and v_{i+2}) are outside the MIS
- If v_{i+3} also outside MIS, then v_{i+2} could enter MIS $\rightarrow$ contradiction to MIS maximal

- v_{i+1} gets color 1, v_{i+2} color 2 if not in MIS, otherwise color 0

$\Rightarrow$ Legal coloring with 3 colors

[Pic: MIS to 3-coloirng]

$\Rightarrow$ Any MIS on the oriented ring/rooted path can be turned into 3-coloring in 2 rounds. Thus, in the class of trees and bounded-degree graphs, there are instances where MIS computation cannot be (much) faster than 3-coloring. □

5.2.2 A Fast Randomized Algorithm for MIS

Algorithm 9: Random-MIS

Each vertex sets $b_v \leftarrow \perp$
repeat
  pick r_v uniformly at random from $[0, 1]$, send r_v to all undecided neighbors
  if r_v *is maximum among undecided neighbors* **then**
    $b_v \leftarrow 1$, send "Decide-1" to all neighbors
  on *getting "Decide-1" from at least one neighbor* w **do**
    $b_v \leftarrow 0$, send "Decide-0" to all neighbors
until $b_v \in \{0, 1\}$

Phase: Iteration of the repeat-loop. Every phase consists of 2 rounds (send IDs, determine and send "Decide-x" messages)

Lemma 24. *Random-MIS computes a feasible MIS and terminates with probability 1.*

Proof. Simple proof uses basic probability facts:

- For a single pair of neighbors v and w, we have $\Pr[r_v = r_w] = 0$.
- Union bound over all pairs of nodes: $\Pr[\exists$ any pair v, w with $r_v = r_w] = 0$
  With probability 1: All nodes v, w have $r_v \neq r_w$.

Thus, in every phase, with probability 1:

$\rightarrow$ Unique node with maximum r_v in every neighborhood
$\rightarrow$ No neighboring nodes join MIS in that round
$\rightarrow$ At least one node (maximum r_v of all undecided nodes) joins MIS in that phase
$\rightarrow$ Algorithm behaves like MIS-Rank, where r_v are IDs.

□

Union bound: Events $X_1, \ldots, X_k$ each have probability $p_1, \ldots, p_k$ to occur, resp. The probability that at least one of them occurs is at most $\sum_i p_i$. We'll use this often implicitly.

r_v are continuous variables (since $[0, 1]$ is continuous and contains uncountable infinitely many possible numbers). In the subsequent analysis, we assume for simplicity that there is only a **countably infinite number of possible values** in $[0, 1]$ that r_v can take. Our random variables will be **discrete and assume rational numbers**. We will discuss the bit complexity in the end and see how we can satisfy that assumption.

Goal: Analyze time complexity of Random-MIS and show it is fast!

An important tool: Linearity of Expectation.

Theorem 9 (Linearity of Expectation)**.** *Let X and Y be two discrete random variables over $\mathbb{R}$. Then*

$$\mathbb{E}[X+Y] = \mathbb{E}[X] + \mathbb{E}[Y]$$

Proof.

$$\begin{aligned}
\mathbb{E}[X] + \mathbb{E}[Y] &= \sum_{X=x} x \cdot \Pr[X=x] + \sum_{Y=y} y \cdot \Pr[Y=y] \\
&= \sum_{X=x}\sum_{Y=y} x \cdot \Pr[X=x] \cdot \Pr[Y=y \mid X=x] \\
&\quad + \sum_{Y=y}\sum_{X=X} y \cdot \Pr[Y=y] \cdot \Pr[X=X \mid Y=y] \\
&= \sum_{X=x}\sum_{Y=y} x \cdot \Pr[(X,Y)=(x,y)] + \sum_{Y=y}\sum_{X=x} y \cdot \Pr[(X,Y)=(x,y)] \\
&= \sum_{X=x,Y=y} (x+y) \cdot \Pr[(X,Y)=(x,y)] \\
&= \mathbb{E}[X+Y]
\end{aligned}$$

□

First attempt: Show that many nodes are decided in each phase of Random-MIS.
v joins MIS only if r_v is maximum in $\Gamma_1(v)$, which happens with probability $1/\deg_G(v)$. If v joins the MIS, then $\deg_G(v)+1$ nodes (i.e., all of $\Gamma_1(v)$) are decided "because of v". Hence, by linearity of expectation:

$$\begin{aligned}
\mathbb{E}[\text{Number of decided nodes}] &= \sum_{v\in V} \Pr[v \text{ joins MIS}] \cdot (\text{Number of nodes decided because of } v) \\
&= \sum_{v\in V} \frac{1}{\deg_G(v)+1} \cdot (\deg_G(v)+1) = |V| \qquad \textbf{WRONG!}
\end{aligned}$$

Complete nonsense! Problem: Nodes are overcounted when neighborhoods of joining nodes overlap! There are graphs, where **in expectation only very few nodes are decided** in some phases.

Second attempt: Show that many **edges** are decided in each phase of Random-MIS, where an edge is decided if *at least one of the endvertices* gets decided.

Lemma 25. *In a single phase, we decide in expectation at least half of the remaining undecided edges.*

Proof. Restrict attention to a single phase and undecided subgraph in the beginning of the phase. W.l.o.g. G contains only undecided nodes and undecided edges.

- Suppose v joins MIS, then $r_v > r_y$ for all neighbors y.
- Consider a fixed neighbor $w \in \Gamma(v)$
- **Event** $(\mathbf{v} \to \mathbf{w})$: v joins MIS and also $r_v > r_x$ for all $x \in \Gamma(w) \setminus \{v\}$
- Let $\Gamma'(v,w) = \Gamma(v) \cup \Gamma(w)$ and note $|\Gamma'(v,w)| < \deg(v) + \deg(w)$
- Ordering nodes of Γ based on r gives uniform random permutation.
- Probability v has highest value in $\Gamma'(u,v)$ is $\Pr[(v \to w)] \geq 1/(\deg(v) + \deg(w))$

Now we estimate the decided edges when event $(v \to w)$ occurs.

- Think of edge $\{u,v\}$ as two directed edges (u,v) and (v,u)
- We say: (u,v) gets decided when $u \notin MIS$, and (v,u) gets decided when $v \notin MIS$.
- If $(v \to w)$ occurs, $w \notin MIS$. Then there are $\deg(w)$ many edges (w,x) that are decided, for $x \in \Gamma(w)$
- For each edge $\{v,w\}$, let $X_{v\to w}$ be random variable counting the decided directed edges due to event $(v \to w)$.
- Clearly, $X_{v\to w} = \deg(w)$ if $(v \to w)$ happens, and $X_{v\to w} = 0$ otherwise.

Let X be total number of directed edges that get decided

- Event $(v \to w)$: No other event $(u \to w)$, since $u, v \in \Gamma(w)$ and $r_v > r_u$
- Directed edge (w,x) decided, then due to *at most one event* $(v \to w)$
- Thus, $X \geq \sum_{\{u,v\}\in E} X_{u\to v} + X_{v\to u}$, no overcounting of directed edges

This implies

$$\begin{aligned}
\mathbb{E}[X] \geq \mathbb{E}\left[\sum_{\{u,v\}\in E} X_{u\to v} + X_{v\to u}\right] &= \sum_{\{u,v\}\in E} \mathbb{E}[X_{u\to v}] + \mathbb{E}[X_{v\to u}] \\
&= \sum_{\{u,v\}\in E} \Pr[(u \to v)] \cdot \deg(v) + \Pr[(v \to u)] \cdot \deg(u) \\
&\geq \sum_{\{u,v\}\in E} \frac{\deg(v)}{\deg(v)+\deg(u)} + \frac{\deg(u)}{\deg(v)+\deg(u)} \\
&\geq \sum_{\{u,v\}\in E} 1 \quad = \quad |E|
\end{aligned}$$

Now an original edge gets decided as soon as at least one of its' directed edges is decided. Since for directed edges $\mathbb{E}[X] = |E|$ and there are 2 directed edges for each original edge, in expectation at least $|E|/2$ original edges are decided. □

Now in each phase we make good progess in deciding edges.
How long until **all** edges are decided?

We use another important tool: Markov Inequality

Theorem 10 (Markov Inequality)**.** *Let X be a non-negative random variable. Then, for any $k > 1$*

$$\Pr[X \geq k \cdot \mathbb{E}[X]] \quad \leq \quad \frac{1}{k} \ .$$

Proof. Here: X discrete variable that takes non-negative integer values in $\{0, 1, 2, 3, \dots\}$, straightforward generalization to more general non-negative variables. If $\Pr[X = 0] = 1$, statement is trivial. Hence, $\Pr[X = 0] < 1$, and, thus $\mathbb{E}[X] > 0$. Then

$$\begin{aligned} \mathbb{E}[X] \quad &= \quad \sum_{i=0}^{\infty} \Pr[X = i] \cdot i \quad \geq \quad \sum_{i=\lceil k\mathbb{E}[X]\rceil}^{\infty} \Pr[X = i] \cdot i \\ &\geq \quad k \cdot \mathbb{E}[X] \cdot \sum_{i=\lceil k\mathbb{E}[X]\rceil}^{\infty} \Pr[X = i] \quad = \quad k \cdot \mathbb{E}[X] \cdot \Pr[X \geq k \cdot \mathbb{E}[X]] \ . \end{aligned}$$

Divide by $k \cdot \mathbb{E}[X] > 0$ and eliminate $\mathbb{E}[X] > 0$, gives the result. □

Corollary 3. *In a single phase, with probability at least 1/4 we decide at least a third of the undecided edges.*

Proof. Again suppose G is the undecided subgraph. Let

- X be random variable counting number of decided edges after the phase
- $\hat{X}$ be random variable counting number of undecided edges after the phase

Now $|E| = X + \hat{X}$, and due to Lemma 25

$$\mathbb{E}[\hat{X}] = |E| - \mathbb{E}[X] \leq \frac{|E|}{2}$$

Apply Markov Inequality with $k = 4/3$:

$$\Pr\left[X \leq \frac{|E|}{3}\right] = \Pr\left[\hat{X} \geq \frac{2 \cdot |E|}{3}\right] \leq \Pr\left[\hat{X} \geq \frac{4 \cdot \mathbb{E}[\hat{X}]}{3}\right] \leq \frac{3}{4}$$

Thus,

$$\Pr\left[X > \frac{|E|}{3}\right] = 1 - \Pr\left[X \leq \frac{|E|}{3}\right] \geq \frac{1}{4} \ .$$

□

Hence, with constant probability, we remove a constant fraction of the edges. Happens only $O(\log n)$ times before we run out of edges.

Random variables $X_1, \dots, X_k$ are **independent** if for all i and $(x_1, \dots, x_k)$ it holds

$$\Pr[X_i = x_i] = \Pr[X_i = x_i \mid X_j = x_j \text{ for all } j \neq i]$$

i.e., probability that $X_i = x_i$ is independent of what happens with the other X_j.

Theorem 11. *Random-MIS computes an MIS in an expected number of $O(\log n)$ rounds.*

Proof. No matter how the graph, we decide a third of the edges with probability 1/4.

- **Good phase**: Number of undecided edges decreases by at least a third
- At most one edge remaining after

$$\log_{3/2}|E| < \frac{\ln(n^2)}{\ln(3/2)} < 5\ln n$$

good phases, then in the next phase algorithm terminates.
- How many phases until we see $5\ln n$ good phases?

Precise bound is tedious! Here only very brutal estimate :)

- Consider time sliced up into **blocks** of $60\ln n$ phases.
- In any block of $60\ln n$ phases, expected number of good phases is at least $15\ln n$ (Corollary 3)
- Expected number of bad phases in each block at most $45\ln n$.
- Apply Markov Inequality: Pr[More than $55\ln n$ bad phases] $\leq 45/55 = 9/11$
- Thus, Pr[No $5\ln n$ good phases in a given block] $\leq 9/11$

Compose blocks with probabilistic domination:

- Subsequent blocks are not independent
- But: Pr[No $5\ln n$ good phases in a given block] $\leq 9/11$ for *any set of* $60\ln n$ *phases.*
- Using probabilistic domination, we can upper bound the probability that all the first i blocks fail to have $5\ln n$ good rounds with i independent Bernoulli draws:

$$\Pr[\text{At the end of } i\text{-th block still no } 5\ln n \text{ good phases}] \;\leq\; \left(\frac{9}{11}\right)^i.$$

- Thus, expected number of blocks at most

$$\sum_{i=0}^{\infty}\left(\frac{9}{11}\right)^i\cdot(i+1) = \sum_{i=0}^{\infty} i\left(\frac{9}{11}\right)^i + \sum_{i=0}^{\infty}\left(\frac{9}{11}\right)^i = \frac{9}{11}\cdot(5.5)^2 + 5.5 = 30.25$$

so expected number of phases at most $1815\ln n$.
- Each phase has 2 rounds, $O(\log n)$ rounds in expectation.

□

Some Remarks:

- Actual constant in running time nowhere near 1815, more like 5, but proving this is difficult!
- Analysis tight up to constants. There are regular graphs with, say, $\Delta = n^{0.01}$ and very few short cycles, where in almost every round, the number of undecided edges falls only by a constant factor each round.
- Algorithms with expected running time $O(\log n)$ known since the 1980s – no algorithm with expected time $o(\log n)$ on all graphs to this date!
- Best lower bound $\Omega(\sqrt{\log n/\log\log n})$ or $\Omega(\log\Delta/\log\log\Delta)$, also for randomized algorithms. Closing the gap is a major open problem!
- Best deterministic algorithm in time $2^{O(\sqrt{\log n})}$, may end up collecting all information about the graph in a single node using huge messages.

Concentration Results

Expected running time sometimes not so useful. Often we rather need to be sure that by a certain point in time the task is done (with high probablity).

With high probability (whp):
- Instance with input parameter n (e.g., number of nodes)
- An event occurs *with high probability* if it does so with probability $1 - 1/n^c$ for any n and a fixed constant $c \geq 1$.
- Usual statement in this course: "Event Y occurs after $O(f(n))$ many rounds whp."
- Means: Y occurs with probability $1 - 1/n^c$, where constant $c \geq 1$ can be chosen freely. Larger choice of c increases the constants required in the $O(f(n))$ term.

The $1 - 1/n^c$ makes application of union bound super convenient!
Example: Event Y_v = "node v terminates" happens after $O(\log n)$ rounds whp. Hence, probability that v runs longer is only $1/n^c$, for any constant c (and suitable constant in the O-term). Then pick $c' = c - 1$ and apply union bound over all n nodes. Hence, probability that *at least one* node runs longer is $n \cdot 1/n^c = 1/n^{c'}$. Thus, all nodes terminate in $O(\log n)$ rounds whp.
This trick even extends to $poly(n)$ many events, e.g., for each edge and each node, etc.

Main tool to obtain whp results: Chernoff bound

Theorem 12 (Chernoff Bound)**.** *Let $X = \sum_{i=1}^{k} X_i$ be the sum of k independent Bernoulli (i.e., 0-1) variables. Then, for every $0 < \delta \leq 1$*

$$\begin{aligned} \Pr[X \geq (1+\delta) \cdot \mathbb{E}[X]] &\leq e^{-\delta^2 \cdot \mathbb{E}[X]/3} \\ \Pr[X \leq (1-\delta) \cdot \mathbb{E}[X]] &\leq e^{-\delta^2 \cdot \mathbb{E}[X]/2} . \end{aligned}$$

Corollary 4. *Random-MIS terminates in $O(\log n)$ rounds whp.*

Proof. Using above lemma, with prob. at least $1/4$, a third of the undecided edges is decided in each phase.
- Holds for every phase, no matter what happened in previous phases!
- Recall proof of Theorem 11
- Here: Bound number of rounds until $5 \ln n$ good phases happen using Chernoff bound!
- For $c \geq 1$, consider $k = 40\lceil c \ln n \rceil$ phases.
- Let $Y_i = 1$ when phase i is good. Note that $\Pr[Y_i = 1] \geq 1/4$.
- Thus, number of good phases is at least $Y = \sum_{i=1}^{k} Y_i$, where $\mathbb{E}[Y] \geq 10\lceil c \ln n \rceil$.

Y_i are not independent, but $\Pr[Y_i = 1] \geq 1/4$ holds always, no matter what. Hence, we can use probabilistic domination and assume that Y_i are independent Bernoulli draws with $\Pr[Y = 1] = 1/4$. Then apply Chernoff bound with $\delta = 1/2$:

$$\Pr[Y < 5 \ln n] \leq \Pr\left[Y < \frac{\mathbb{E}[Y]}{2}\right] \leq e^{-\mathbb{E}[Y]/8} < e^{-1.25 c \ln n} < n^{-c}$$

Thus, the probability that Random-MIS *does not terminate* within $k = 40\lceil c \ln n \rceil$ phases is at most $1/n^c$. This implies the algorithm terminates in $2(k+1) \in O(\log n)$ rounds whp. $\square$

Bit Complexity

Problems with real numbers r_v

- Need possibly infinite number of bits to communicate
- Overcountably many possibilities, some formulas above need integrals (nooo! :)

Solution: Consider $r_v \in [0,1]$ in bit representation $r_v = 0.b_1^v b_2^v b_3^v b_4^v \ldots$, draw bits $b_i \in \{0,1\}$ iteratively at random.

For a given edge $\{v,w\}$ decide $r_v > r_w$ or $r_v < r_w$ (note $r_v = r_w$ has probability 0):

- Compare leading bits
- Number of bits X_{vw} that must be drawn and communicated?
- Smallest number i such that $b_i^v \neq b_i^w$. Since v and w draw bits independently

$$\Pr[0.b_1^v b_2^v \ldots b_{i-1}^v = 0.b_1^w b_2^w \ldots b_{i-1}^w] = \left(\frac{1}{2}\right)^{i-1} \quad \text{and} \quad \Pr[b_v^i \neq b_w^i] = \frac{1}{2}$$

- Hence,

$$\mathbb{E}[X_{vw}] = \sum_{i=1}^{\infty} \left(\frac{1}{2}\right)^i \cdot i = 2 \ ,$$

  on every edge in expectation only 2 bits.

Yes, but: Each v draws r_v only **once overall, but not once per edge**, so number of bits on edges $\{u,v\},\{u,v\},\{u,w\}$ not independent!

- Suppose given pair of nodes v,w must exchange $X_{vw} > 1 + (c+3)\log_2 n$ bits.
- Probability of that event is

$$\Pr[X_{vw} > 1 + (c+3)\log_2 n] \ \leq \ \left(\frac{1}{2}\right)^{(c+3)\log_2 n} = \frac{1}{n^{c+3}}$$

- At most n^2 edges, using a union bound:

$$\Pr[X_{vw} > 1 + (c+3)\log_2 n \text{ for at least one } \{u,v\} \in E] \quad < \quad \frac{1}{n^{c+1}}$$

- Thus, whp in a single phase not more than $1 + (c+3)\log_2 n$ bits on any edge.
- Now apply another union bound over the (at most) n phases.
  (actually $O(\log n)$ phases would be enough whp)

Lemma 26. *Random-MIS computes an MIS in $O(\log n)$ rounds and exchanges $O(\log n)$ bits on every edge in every round whp.*

5.2.3 Applications

Fast computation of MIS has lots of interesting applications:

Coloring:
$(\Delta + 1)$-coloring arbitrary graphs can be done very fast.

Theorem 13. *If there is an algorithm in the* LOCAL *model to compute an MIS on* G *in time* $T(G)$*, there is an algorithm to compute a* $(\Delta + 1)$*-coloring on* G *in time* $O(T(G))$*.*

Proof. Exercise. □

Corollary 5. *For any graph* G*, there is a distributed randomized algorithm to compute a* $(\Delta + 1)$*-coloring in time* $O(\log n)$ *whp.*

Matching:
Subset of edges $M \subseteq E$. Constraint: No two edges in M have same endnode
Maximal matching M: No edge from E can be added to M without violating the constraint

Corollary 6. *For any simple graph* G*, there is a distributed randomized algorithm to compute a maximal matching in time* $O(\log n)$ *whp.*

Idea:
- Simple graph $G = (V, E)$ has a *line graph* $G_L = (E, L)$
- G_L has an edge $\ell = (e, e') \in L$ if and only if $e \cap e' \neq \emptyset$ (i.e., edges become nodes and a line-graph-edge exists if only if edges share an endnode).
- Maximal matching in G = MIS in G_L.
- Simulate execution of Random-MIS in G_L.

Vertex Cover:
Subset of nodes $C \subseteq V$. Constraint: For every edge $e \in E$ must be at least one endnode in C (can be both, but at least one)
Minimum vertex cover C^*: Vertex cover with smallest cardinality

Corollary 7. *For any graph* G*, there is a distributed randomized algorithm to compute a vertex cover in time* $O(\log n)$ *whp. The resulting vertex cover* C *is a 2-approximation, i.e.,* $|C| \leq 2|C^*|$*.*

Proof. Distributed version of classic matching heuristic (recall "Theoretische Informatik 1"):
- Compute maximal matching M as above.
- All nodes incident to matching edges join the vertex cover C.
- Minimum vertex cover C^* as least as large as any maximal matching M (no node in C^* can cover two or more matching edges from M).
- Hence $|C^*| \geq |M| = |C|/2$.

□

Chapter 6

Minimum Spanning Trees

General setup:

- Simple graph $G = (V, E, \omega)$ with weights i.e., every edge e has $\omega(e) \geq 0$.
- Synchronous $\mathsf{CONGEST}$ model.
- Every $\omega(e)$ composed of $O(\log n)$ bits, w.l.o.g. we assume integers $\omega(e) \in \{0, 1, 2, \ldots, n^c\}$ for some constant c.
- Every node knows ID and weights of every incident edge.

Goal: Compute a **minimum spanning tree (MST)** of G, i.e., a spanning tree T^* with smallest total weight $\omega(T^*) = \sum_{e \in T^*} \omega(e)$. Every node should know its incident edges of T^*.

Some defintions:

- Edge weights are integers represented in $O(\log n)$ bits, can be sent in one message.
- G has **distinct weights**: There are no two edges with same weight.
- We assume distinct weights w.l.o.g.: Tie-breaking using IDs of involved nodes.
- If T^* is MST of G, then every $T' \subseteq T^*$ is called **fragment** of T^*.
- Edge $e = \{u, v\}$ is **outgoing edge** of T' if $u \in T'$ and $v \notin T'$.
- The minimum-weight outgoing edge $b(T')$ is the **blue edge** of T' (unique because of distinct weights)

Lemma 27. *For G with distinct weights, let T^* be an MST and T' a fragment of T^*. Then $b(T')$ is also part of T^*, i.e., $b(T') \cup T' \subseteq T^*$.*

Proof. Suppose not, then there is $e' \neq b(T')$ connecting T' with rest of T^*. Then, adding $b(T')$ to T^* gives a cycle with both e' and $b(T')$. Now add $b(T')$ to T^* and remove e'. Due to distinct weights, this gives a strictly cheaper spanning tree than T^*. Contradiction. □

[Pic: Example Blue Edge]

Iteratively adding blue edges is key idea of both algorithms of Jarnik-Prim (grows T' starting from a source r_0) and Kruskal (grows T' by the globally best blue edge). The following lemma is thus obvious.

Lemma 28. *G distinct weights $\Rightarrow$ Unique $b(T')$ for every fragment T' $\Rightarrow$ Unique MST T^*.*

6.1 GHS Algorithm

Distributed approach by parallel addition of blue edges. Resembles **Boruvka's algorithm** for MST.

Algorithm **GHS** (Gallagher-Humblet-Spira) starts with singleton fragments. In each phase, simultaneously all maximal fragments add their blue edge to T and get merged. Repeat phase until single tree is formed.

Below is a sketch in pseudocode with some missing details, e.g., fragments can have different size, so in Line 8 node u in fragment T' might need to wait for v to decide if blue edge $b(T')$ is also blue edge of v's fragment and hence v will (eventually) be sending a merge request. Good news: All missing details can be handled!

Algorithm 10: GHS (Gallager-Humblet-Spira)

```
1  Initially each node is root of own fragment. We proceed in phases:
2  repeat
3      All nodes learn fragment IDs of neighbors
4      Root r_{T'} of each fragment T' initiates upcast to find blue edge b(T') = {u, v}
5      r_{T'} sends message to u; while forwarding message, parent-child relations on
         r_{T'}-u-path are inverted. u becomes temporary root of T'
6      u sends merge request on b(T')
7      Fragments of u, v and blue edge b(T') get merged
8      if v also sent merge request on b(T') then
9          Either u or v (the one with smaller ID) is new fragment root
10     else
11         v becomes parent of u
12     Edge {u, v} directed accordingly
13     Newly elected roots inform their fragments (using flood/echo) about their ID
14 until single tree is formed
```

[Pic: Example merge of several fragments in a phase]

GHS joins fragments via blue edges. By Lemma 27 resulting tree must be the MST T^*.

Corollary 8. *GHS computes the MST T^*.*

Lemma 29. *GHS needs $O(\log n)$ phases. Each phase can be implemented in $O(n)$ rounds.*

Proof. Number of phases:
By induction: In the end of phase i, every fragment contains at least 2^i nodes.

- True in the beginning (phase 0), singleton fragments, $2^0 = 1$ node each.
- In phase i, by hypothesis, each fragment has at least 2^{i-1} nodes.
- Every fragment is attached to some other fragment, resulting fragments have at least $2 \cdot 2^{i-1} = 2^i$ nodes.

- Fragment with n nodes after $\log_2 n$ many phases

Number of rounds per phase:
Each phase one upcast to find blue edge in old fragments, one message from root to u, and one flood/echo in new fragments. All can be implemented in $O(n)$ rounds. Remaining steps per phase can be implemented in $O(1)$ rounds. □

Note: Diameter of MST fragments (wrt. number of edges) unrelated to diameter of graph (wrt. number of edges), so number of rounds not necessarily related to $Diam(G)$.

Theorem 14. *GHS computes the MST in* $\mathsf{Time}(GHS, G) = O(n \log n)$.

Original GHS algorithm is asynchronous with message complexity $O(|E| \log n)$, which can be improved to $O(|E| + n \log n)$. This is a lot better than applying the synchronous algorithm with a α-synchronizer. The fragments are used like an β-synchronizer, only exchange of fragment IDs is costly.

6.2 Distributed Dual Greedy

Now we use red edges:
In a graph G with distinct weights, edge $e \in E$ is a **red edge** if there is a cycle C such that $e \in C$ and e is the highest-weight edge of C.

Lemma 30. *$e \in E$ is a blue edge $\Leftrightarrow e \in T^* \Leftrightarrow e \in E$ is not a red edge.*

Proof. Exercise. □

Compute BFS, upcast all edges to root for MST computation. Delete red edges on the fly.

Algorithm 11: Dual Greedy

Compute (unweighted) BFS-tree T^B, denote root by r_0
For each v let $E_v \leftarrow$ Set of edges incident to v, and $S_v \leftarrow \emptyset$
on *v is leaf or received at least one message from every child* **do**
    repeat
        Add all edges received from children to E_v
        foreach *cycle C in E_v* **do**
            Pick heaviest edge $e' = \arg\max_{e \in C}\{\omega(e)\}$
            Remove e' from E_v
        Pick cheapest known and unsent edge $e' = \arg\min_{e \in E_v \setminus S_v}\{\omega(e)\}$
        Send e' to parent, add e' to S_v
    until $E_v \subseteq S_v$ *(no more unsent non-red edges)*
r_0 broadcasts MST E_{r_0} over BFS tree

Regular upcast of all edges: $O(|E| + Depth(T_B)) = O(|E|)$ time. Dual Greedy is faster! Intuitively, blue edges of MST are not delayed long, so red edges are quickly found and removed. Algorithm takes only $O(\mathsf{Time}(BFS, G) + |T^*| + Depth(T_B)) = O(n)$ time.

For simplicity **re-number rounds:** Round 1 is **first round after end of BFS computation**. Easy observation:

Lemma 31. *Every vertex v starts sending messages upwards at round $\hat{L}(v)$.*

We call a node **active** if it still runs the repeat-loop.

Lemma 32.
(a) For each child u of v, u active at round t, E_v at the start of round t contains at least one edge sent by a child of v.
(b) v sends edge of weight ω_0 at round $t \Rightarrow$ All edges v received by active children in round $t-1$ have higher weight.
(c) If v sends edge of weight ω_0 at round $t \Rightarrow$ Any edge v will learn in later rounds has higher weight.
(d) v sends edges in increasing order of weight.
(e) v sends a cycle-free subset of edges.

Proof. Induction over tree level and time. All claims (a) – (e) hold for leaf nodes.
Consider intermediate vertex v, assume all claims (a) – (e) hold for all children of v, and for v until round $t-1$.

(a) Let A_v be set of m edges sent by v to parent during first $m = t - \hat{L}(v)$ rounds it was active (i.e., rounds $\hat{L}(v), \ldots, t-1$). Consider active child u, let A_u be edges sent to v until round $t-1$. u is active on round $t-1$, so transmitted without pause since round $\hat{L}(u) \leq \hat{L}(v) - 1$. Hence, $|A_u| \geq m+1$.
Hypothesis: (e) holds for u and for v until round $t-1$. Hence, both A_u and A_v are cycle-free. Since $|A_u| > |A_v|$, there is at least one edge sent by u that is not in a cycle with A_v and has not been sent by v. Thus, there is at least one edge (by u or another child) in E_v that has not been sent.

(b) Consider active child u. u sent e_t in round $t-1$. There is earlier round $t' \leq t-1$, where u sent $e_{t'}$, which is still in E_v at time t and has not been sent (or an even cheaper one, due to (a)). Hypothesis (d) for u: $\omega(e_{t'}) \leq \omega(e_t)$. v transmits cheapest edge, so $\omega_0 \leq \omega(e_{t'})$.

(c) Follows directly from (b).

(d) Follows directly from (c) and transmitting cheapest edges.

(e) Hypothesis: (d) + (e) holds for v for all rounds until $t-1$. Suppose e sent by v in round t closes a cycle with the edges sent previously by v. Then e closes a cycle in E_v upon arrival at v. Due to (d) and (e), e must be the red edge in that cycle, so e is not sent in round t, contradiction.

□

The following lemma follows directly from (a) above:

Lemma 33. *After v becomes non-active, it does not learn new edges from its children.*

Overall, this implies:

Theorem 15. *Dual Greedy computes the MST in* $\mathsf{Time}(DualGreedy, G) = O(n)$.

Proof. Removal of red edges never hurts the MST $\Rightarrow$ Root receives all MST edges. Lemma 31: Root starts getting messages at time $\mathsf{Time}(BFS, G) + Depth(T^B)$. Lemma 32 (e): Root receives at most $|T^*| = n - 1$ edges from each child in a pipelined consecutive fashion. $\mathsf{Time}(DualGreedy, G) = O(\mathsf{Time}(BFS, G) + Depth(T^B) + |T^*|) = O(n)$. □

This algorithm is asymptotically optimal for some graphs with large diameter.

Lemma 34. *Every distributed algorithm to compute an MST on the ring requires* $\Omega(n)$ *many rounds.*

Proof. Exercise. □

6.3 GKP Algorithm

Advantages and disadvantages of previous algos:

GHS: Reduces number of maximal fragments by a factor of 2, grows the MST quickly (fast!)
Possibly large diameter and coordination overhead inside fragments (slow!)
Fast initially, then gets slower due to larger components

Dual Greedy: If BFS-tree is path, grows the MST one blue edge at a time (slow!)
Processing of edges in pipelined fashion (fast!)
Slow initially, then gets faster due to removal of red edges

Algorithm **GKP** (Garay-Kutten-Peleg) combines advantages. Initially, grows fragments quickly, but more carefully than GHS. When all maximal fragments have size of $\sqrt{n}$, uses Dual Greedy on the inter-fragment edges to quickly finish MST construction.

Algorithm 12: GKP (sketch)

Construct unweighted BFS-tree, determine and inform all nodes about n
$T \leftarrow$ Set of all singleton fragments `// stays forest throughout`
for $i = 0, \dots, \lceil \log_2 \sqrt{n} \rceil$ **do**
  $\mathcal{C} \leftarrow$ Set of maximal fragments (i.e., components) of forest T, and $E_C \leftarrow \emptyset$
  Each $T' \in \mathcal{C}$ of diameter at most 2^i finds blue edge $b(T')$, adds it to E_C
  Find maximal matching M in graph $(\mathcal{C}, E_C)$, add edges of M to T
  if $T' \in \mathcal{C}$ of diameter at most 2^i has no incident edge in M **then** add $b(T')$ to T
Let $G' = (V, E', \omega')$ be weighted multigraph obtained when contracting edges in T (delete loops, keep multi-edges)
Run Dual Greedy on G', add chosen edges to T

Phase: Iteration of the Repeat-Loop.
Contraction of edge $e = \{u, v\}$: Merge u and v into a single vertex, e becomes a loop.

During **for**-loop, GKP builds fragments only via blue edges. Call of Dual Greedy in line 9 adds exactly the MST among remaining maximal fragments. By Lemma 27, this implies:

Corollary 9. *GKP computes the MST T^*.*

Time complexity for graph G' and Dual Greedy (lines 8-9):

Lemma 35. *Suppose in line 8, T contains at most k components, each component has diameter at most $Diam(T)$. Then lines 8-9 take at most $O(Diam(G) + Diam(T) + k)$ rounds.*

Proof. Line 7: For each of the k remaining components, find and broadcast smallest ID of any node in the component as the comonent ID. Each node knows which edges go within the component or to another one. Effectively "contracts" all inner-component edges and takes at most $O(Diam(T))$ time.

Line 8: Dual Greedy applied on BFS tree of the entire (uncontracted) graph G. In pipelined upcast, each node v only places incident inter-component edges into E_v (and avoids upcasting "contracted" inner-component edges). Up to k^2 inter-component edges, but only $k-1$ remain for MST among components. Analysis for Theorem 15 shows that the algorithm terminates in time $O(Diam(G) + k)$. □

[Pic: Example Fragments, Contraction, Final BFS and Upcast]

Time complexity of the **for**-loop (lines 3-7): Show that
- components do not get large diameter (Lemma 36)
- fast implementation of each phase in the loop (Lemma 37)
- only few components at the end of the loop (Lemma 38)

Lemma 36. *At the end of phase i, each component of T has diameter at most $O(2^i)$.*

Proof. Induction hypothesis: Suppose at the end of phase $j = 1, \ldots, i-1$, all components have diameter at most $12 \cdot 2^j$ (trivially true in phase 1). Now consider phase i.
- Graph $(\mathcal{C}, E_C)$ with all components and blue edges for $(\leq 2^i)$-diameter components
- Pick maximal matching M in $(\mathcal{C}, E_C)$
- Unmatched components add their blue edge, denote these edges by M_s
- Since M is maximal, unmatched components attached to matched components
- For $e = \{C_1, C_2\} \in M$, edges of M_s attached directly to C_1 and C_2
- New components have diameter at most 3 w.r.t. $M \cup M_s$
- For each new component, at most one component from $\mathcal{C}$ with diameter more than 2^i: Components with larger diameter do not add their blue edges to E_C, so no edges in $M \cup M_s$ between two such components.
- Total diameter of new component:

$$12 \cdot 2^{i-1} + 3 + 3 \cdot 2^i \quad = \quad 6 \cdot 2^i + 3 \cdot 2^i + 3 \quad \leq \quad 12 \cdot 2^i \ . \qquad \square$$

[Pic: Example, Matching, Attach unmatched nodes, resulting diameter]

Lemma 37. *Each phase can be implemented in time $O(2^i \log^* n)$*

Proof. Previous lemma: Components in phase i have size $O(2^i)$. For each component, each of the following can be done using flood/echo and convergecast in $O(2^i)$ time:

- Find root node (smallest ID)
- Build inner-component BFS-tree from root inside the component
- Determine if size is $\leq 2^i$, if yes determine blue edge of component.

This way we determine E_C. Now build a maximal matching in $(\mathcal{C}, E_C)$ as follows:

- Each edge in E_C directed away from the component that chose it.
- $(\mathcal{C}, E_C)$ becomes a directed graph, each node outdegree 1
- If we follow any directed path of blue edges in $(\mathcal{C}, E_C)$, the weights of blue edges strictly decrease. Hence, components of $(\mathcal{C}, E_C)$ are **rooted trees** with "root" possibly being a single blue edge chosen by both incident components
- Determine a root node for every tree in $(\mathcal{C}, E_C)$, simulate the 3-coloring algorithm for trees, takes $O(\log^* n)$ steps
- Each component acts as a single node in coloring algorithm, coordinated by its' root with broad-/convergecast over inner-component BFS tree
- Hence, each step of the coloring algorithm needs $O(2^i)$ rounds
- Determine maximal matching M of $(\mathcal{C}, E_C)$ from 3-coloring (c.f. Exercises). in $O(1)$ steps, i.e., $O(2^i)$ rounds

[Pic: Forest structure of $(\mathcal{C}, E_C)$]

Adding edges of unmatched components and component merge (i.e., informing all nodes about new component structure) in time $O(2^i)$.
Overall: $O(2^i \log^* n)$ rounds per phase. □

Lemma 38. *At the end of the last phase $\lceil \log_2 \sqrt{n}\ \rceil$, there are at most $\sqrt{n}$ components in T.*

Proof. By induction: At the end of phase i, every component contains at least 2^i nodes. Every component of size at most 2^i is joined in phase i with another component. Same induction as for GHS in Lemma 29. Hence, after $\lceil \log_2 \sqrt{n} \rceil$ phases, every component has at least $2^{\lceil \log_2 \sqrt{n} \rceil} \geq 2^{\log_2 \sqrt{n}} = \sqrt{n}$ nodes $\Rightarrow$ at most $n/\sqrt{n} = \sqrt{n}$ components. □

Theorem 16. *GKP computes the MST in $\mathsf{Time}(GKP, G) = O(Diam(G) + \sqrt{n} \cdot \log^* n)$.*

Proof. Build BFS-tree, determine n, inform all nodes: Time $O(Diam(G))$. **for**-loop phase i: Time $O(2^i \log^* n)$. **for**-loop ends with at most $\sqrt{n}$ components, so finish with Dual Greedy: Time $O(Diam(G) + \sqrt{n})$. In total:

$$\mathsf{Time}(GKP, G) = O(Diam(G)) + O(Diam(G) + \sqrt{n}) + \sum_{i=0}^{\lceil \log_2 \sqrt{n} \rceil} O(2^i \log^* n)$$
$$= O(Diam(G) + \sqrt{n}) + O(\log^* n) \cdot \sum_{i=0}^{\lceil \log_2 \sqrt{n} \rceil} 2^i$$

$$< O(Diam(G) + \sqrt{n}) + O(\log^* n) \cdot 2^{2+\log_2 \sqrt{n}}$$
$$= O(Diam(G) + \sqrt{n}\log^* n)$$

□

6.4 Lower Bound

GKP running time $O(Diam(G) + \sqrt{n}\log^* n)$. Can we improve upon these terms?

Diam(G): Not really :) If using unlimited messages, then in the LOCAL model with simultaneous wakeup and topology knowledge (only weights unknown), running time can be refined to a notion called **cycle-radius** that determines the running time (for details, see Section 24.1 in the Peleg book).

$\sqrt{\mathbf{n}}\log^* \mathbf{n}$**:** G must have low diameter to beat this. There is a lower bound of $\Omega(\sqrt{n}/\log^2 n)$ in graphs G with $Diam(G) = O(\log n)$. For diameter 1 (aka complete graphs), however, we can even solve the problem in time $O(\log n)$. (see Exercises)

Let's discuss the second term. The following is a slightly better lower bound, but allows somewhat larger diameter $Diam(G) = O(n^{1/4})$.

Theorem 17. *There is a class of n-node graphs G with $Diam(G) = O(n^{1/4})$ such that every distributed algorithm for MST in the synchronous* CONGEST *model needs time $\Omega(\sqrt{n}/\log n)$.*

Note: For these graphs GKP has running time $O(\sqrt{n}\log^* n)$.

Running time depends on auxiliary **Mailing Problem** in the CONGEST model: Given graph G with source s, receiver r, and k-bit message at s, inform r about the k-bit message of s.

Given integer $m \geq 1$, we build a hard graph HG_m for the m^2-bit mailing problem:

- Single **highway** of length m, i.e., a path $H = (h_0, h_m, h_{2m}, \ldots, h_{m^2})$, where $h_0 = s$ and $h_{m^2} = r$.
- m^2 many **simple paths** of length m^2 edges each. Path $P^i = (v_0^i, v_1^i, v_2^i, \ldots, v_{m^2}^i)$, for $i = 1, \ldots, m^2$.
- Highway node h_j is **center of star** S_j, i.e., h_j connected to one node in each of the m^2 paths. Star S_j composed of edges $\{h_j, v_j^i\}$ for each $j = 0, m, 2m, 3m, \ldots, m^2$ and all $i = 1, \ldots, m^2$.

Graph has $n = \Theta(m^4)$ nodes and diameter $Diam(HG_m) = O(m) = O(n^{1/4})$.

Intuition: Cannot route all m^2 bits quickly over the highway edges. Alternative routes along star edges and (sub-)paths of some P^i are too long to be helpful.

Prove the intuition formally for explicit delivery algorithms (only transmit bits as they are).

Lemma 39. *For every $m \geq 1$, no explicit delivery algorithm can solve the m^2-bit mailing problem on the hard graph HG_m in time $o(m^2/\log m)$.*

Proof. Consider bit x_i going from s to r, let Q_i be the path taken.

- Q_i goes through every star S_j.
- From S_j to S_{j+1} either Q_i uses highway edge or some path P^i (or even longer path, but this only makes things worse)
- ℓ_i number of highway edges in $Q_i \Rightarrow |Q_i| \geq (m - \ell_i) \cdot m + \ell_i$ time needed for path Q_i
- If there is bit x_i with $\ell_i \leq \lfloor m/2 \rfloor$, then $|Q_i| \geq m^2/2$ and we are done.
- Otherwise, all $\ell_i \geq \lfloor m/2 \rfloor$, i.e., every bit uses at least half of the highway edges
- Summing over all paths Q_i: $\sum_{i=1}^{m^2} \ell_i \geq m^3/2$.
- Highway H has m edges, at least one edge traversed by at least $m^2/2$ bits.
- Since message size is $O(\log n)$, this requires $\Omega(m^2/\log n) = \Omega(m^2/\log m)$ rounds.

$\square$

More generally: Arbitrary algorithms may combine bits, manipulate them, do arbitrary computation, etc. Still, lower bound for the mailing problem applies. Rough proof idea:

- Consider possible states of vertex v (state contains all local data, i.e., input, history, messages it received, etc.)
- Initially, every vertex at unique initial state. Only sender s has 2^{m^2} possible states based on its m^2-bit vector
- Over time more possible executions, hence more possible states at each vertex
- At the end, r must know the bit vector, so be in one of 2^{m^2} possible states
- Growth process of states is slow, forcing algorithm to spend at least $\Omega(m^2/\log m)$ time until set of possible states of r grows to 2^{m^2}.
- Proof uses fundamental insights from communication complexity

Lemma 40. *For every $m \geq 1$, no distributed algorithm can solve the m^2-bit mailing problem on the hard graph HG_m in time $o(m^2/\log m)$.*

Relation to MST:

Proof of Theorem 17. Idea: m^2-bit vector corresponds to incident weights at s. Must become known to r to decide which of his incident edges are in MST of HG_m (global cycle property of MST). Formally, edge weights are as follows:

- Highway H: All edges $\omega(e) = 0$
- Path P^i for $i = 1, \ldots, m^2$: All edges $\omega(e) = 0$
- Star S_j for $j = 1, m, 2m, \ldots, (m-1)m$: All edges $\omega(e) = \infty$
- Star S_{m^2} with root r: All edges $\omega(e) = 2$
- Star S_0 with root s: For $i = 1, \ldots, m^2$ we have

$$\omega(\{s, v_0^i\}) = \begin{cases} 1 & \text{if bit } x_i = 0 \\ 3 & \text{otherwise.} \end{cases}$$

MST T^* contains:

- all edges from highway H and paths P^i, but no star edge from $S_1, \ldots, S_{(m-1)m}$
- For star S_0 at s: Edge $\{s, v_0^i\} \in T^*$ if and only if $x_i = 0$ in the bit vector.
- For star S_{m^2} at r: Edge $\{r, v_{m^2}^i\} \in T^*$ if and only if $x_i = 1$.

To determine incident MST edges, r must learn the bit vector from s. Mailing problem implies the lower bound of $\Omega(m^2/\log m) = \Omega(\sqrt{n}/\log n)$. $\square$

Chapter 7

Distance and Route Approximation

General setup again:

- Simple graph $G = (V, E, \omega)$ with edge weights $\omega(e) > 0$.
- Synchronous $\mathsf{CONGEST}$ model, every node knows ID and weights of incident edges.
- $\omega(e)$ composed of $O(\log n)$ bits. We let $\omega(e) \in \{1, 2, \ldots, n^c\}$ for some constant c.
- For most of the chapter, actually discuss **unweighted graphs** with $\omega(e) = 1$.

Goal: Compute **all-pairs-shortest-paths (APSP)**. At the end of the algorithm every node u needs to have a **routing table**, which contains for every other node $v \neq u$ (1) the shortest distance $\mathsf{dist}(u, v)$ from u to v, and (2) the neighbor of u on a shortest path from u to v.

For $\alpha \geq 1$, an **α-approximation to APSP** produces a routing table for every u such that the entries $d(u, v)$ satisfy $\mathsf{dist}(u, v) \leq d(u, v) \leq \alpha \cdot \mathsf{dist}(u, v)$ for every node $v \neq u$, and the table gives the neighbor of u on a path to v of length at most $d(u, v)$.

For APSP there are near-linear lower bounds for time complexity, even for unweighted graphs.

Theorem 18. *Any deterministic α-approximation to APSP in the synchronous* $\mathsf{CONGEST}$ *model requires $\Omega(n/\log n)$ rounds, even in trees of depth 2.*

Proof Idea: A tree with root node, two children, and $n - 3$ grandchildren. All edge weights 1. Communicate the arbitrary distribution of the $n - 3$ grandchildren to the root $\Rightarrow$ Many bits must be transfered $\Rightarrow$ Many rounds necessary. (Exercise) □

Corollary 10. *Any randomized α-approximation to APSP in the synchronous* $\mathsf{CONGEST}$ *model requires $\Omega(n/\log n)$ rounds, even in trees of depth 2.*

Proof Idea: Consider any deterministic algorithm on a tree as in the previous proof, where grandchildren are attached uniformly at random to one of the two children. After $o(n/\log n)$ rounds, probability for correct output is very small. Apply Yao's principle. (Exercise) □

Similar examples give a lower bound of $\Omega(n)$ for trees with arbitrary weights. Even if we only want distances (and not the paths), a linear lower bound applies.

7.1 Exact APSP in Unweighted Graphs

Trivial solution: Sequentially apply BFS tree algorithms from Chapter 4.1, overall time complexity $O(n \cdot Diam(G))$.

Pipelined Bellman-Ford Algorithm

- solves APSP in unweighted graphs in $\mathsf{Time}(PipeBF, G) = n + O(Diam(G))$
- even if only a subset $S \subseteq V$ of nodes are sources
- solves single-source shortest path in parallel for every source $s \in S$
- based on a comparison of distance/node pairs $(d, u) \in \mathbb{N}_0 \times I$, where I is the numerical ID space of the nodes. Then

$$(d, u) > (d', v) \quad \Leftrightarrow \quad (d > d') \vee ((d' = d) \wedge (u > v))$$

- every vertex v maintains list L_v of distance/source pairs sorted in **ascending order**
- also v maintains $F(s)$ = neighbor on shortest path to source $s \in S$

Algorithm 13: PipeBF

Build BFS tree T_B, inform all nodes about n and $d = Depth(T_B)$
$H \leftarrow 2d$, $K \leftarrow n$
if $v \in S$ **then** $L_v \leftarrow \{(0, v)\}$ **else** $L_v \leftarrow \emptyset$
for $i = 1, \ldots, H + K - 1$ **do**
  $(d_u, u) \leftarrow$ smallest element of L_v not sent before ($\perp$ if there is none)
  if $(d_u, u) \neq \perp$ **then** send $(d_u + 1, u)$ to all neighbors
  foreach (d_u, u) *received from neighbor* w **do**
    if *there is no* $(d'_u, u) \in L_v$ *with* $d'_u \leq d_u$ **then**
      attach $L_v \leftarrow L_v \cup \{(d_u, u)\}$ and set $F(v, u) \leftarrow w$
    if there is $(d'_u, u) \in L_v$ with $d'_u > d_u$ **then** remove $L_v \leftarrow L_v \setminus \{(d'_u, u)\}$

Some notation/definitions:

- L_v^r is list at the end of round r, final list $L_v = \{(\mathsf{dist}(v, s), s) \mid s \in S\}$
- $L_v(h)$ contains only entries of final list L_v with $\mathsf{dist}(v, s) \leq h$.
- $L_v(h, k)$ contains only smallest k entries of $L_v(h)$

We show correctness by induction: After r rounds, consider any $h, k \geq 1$ with $h + k \leq r + 1$. Then the first $|L_v(h, k)|$ entries of L_v are correct.

Lemma 41. *If $(d_w, w) \in L_v^r$ for any $r \geq 0$, then $w \in S$ and $d_w \geq \mathsf{dist}(v, w)$. If $L_v(h, k) \subseteq L_v^r$, it is the head of list L_v^r.*

Proof.

- Never add entries for nodes $v \notin S$.
- Initially, every $s \in S$ has only $(0, s) \in L_s^0$.
- Increase d-values by 1 for each hop and each round. $\mathsf{dist}(v, s) \leq d_s$ for all $(d_s, s) \in L_v^r$.

- For each s, the entries $(d_s, s) \in L_v$ are monotonically decreasing in d_s over time.

$\square$

Lemma 42.

$$L_v(h,k) \subseteq \{(\mathit{dist}(w,s)+1, s) \mid (\mathit{dist}(w,s), s) \in L_w(h-1,k) \wedge \{v,w\} \in E\} \cup \{(0,v)\}$$

Proof.

- $(\mathsf{dist}(v,v), v) = (0,v)$, so $v \in S$ is covered.
- Suppose $(\mathsf{dist}(v,s), s) \in L_v(h,k)$ for some $s \neq v$
- w is neighbor of v on shortest path from v to s, then $\mathsf{dist}(w,s) = \mathsf{dist}(v,s) - 1 \leq h-1$.
- Suffices to show: $(\mathsf{dist}(w,s), s) \in L_w(h-1,k)$. Assume otherwise for contradiction.
- Then there are k pairs $(\mathsf{dist}(w,s'), s') \in L_w(h-1,k)$ with $(\mathsf{dist}(w,s'), s') \leq (\mathsf{dist}(w,s), s)$
- Hence, $(\mathsf{dist}(v,s'), s') \leq (\mathsf{dist}(w,s')+1, s') \leq (h, s')$. If $\mathsf{dist}(v,s) = \mathsf{dist}(v,s')$ then it must be that $\mathsf{dist}(w,s) = \mathsf{dist}(w,s')$ and $s' < s$.
- Therefore, also for v we have $(\mathsf{dist}(v,s'), s') < (\mathsf{dist}(v,s), s)$.
- There are k pairs of this kind, $(\mathsf{dist}(v,s), s)$ is not part of $L_v(h,k)$ - contradiction.

$\square$

[Pic: Distance List]

Lemma 43. *For every $v \in V$, $r = 0, \ldots, H+K-1$ and $h+k \leq r+1$*

1. *$L_v(h,k) \subseteq L_v^r$ and*
2. *v has sent $L_v(h,k)$ by the end of round $r+1$.*

Proof. By induction on r. Both statements hold trivially for $k=0$. They also hold for $h=0$ and all k, since $L_v(0,k) = \{(0,v)\}$ if $v \in S$ and $\emptyset$ otherwise.

- Suppose lemma holds for r, consider round $r+1$
- $h = 0$ covered above, so let $h > 0$.
- By hypothesis: For $h+k \leq r+1$, node v received all elements from $L_w(h-1,k+1)$ and $L_w(h,k)$ of all neighbors w.
- Lemma 42: v received all elements from $L_v(h,k+1)$ and $L_v(h+1,k)$
- Lemma 41: $L_v(h,k)$ always head of list $\Rightarrow$ Part 1. for $h+k \leq (r+1)+1$.

- $L_v(h,k) \subseteq L_v^{r+1}$ for all $h+k \leq r+2$, head of the list, algorithm sends next unsent element from $L_v(h,k)$ if there is any
- By hypothesis: v sent $L_v(h,k-1)$ during first $r+1$ rounds
- Only elements $L_v(h,k) \setminus L_v(h,k-1)$ are missing, but $|L_v(h,k) \setminus L_v(h,k-1)| \leq 1$.
- At most one element from $L_v(h,k)$ remains to be sent and will be in round $r+1$.
- Thus part 2. holds for $h+k \leq (r+1)+1$.

$\square$

Theorem 19. *PipeBF solves APSP in unweighted graphs in $\mathsf{Time}(PipeBF, G) = n + O(Diam(G))$.*

Proof. Build BFS tree T_B rooted at the node with smallest ID. Determine n and $d = Depth(T_B) \leq Diam(G) \leq 2d$. Broadcast them to all nodes. Takes $O(Diam(G))$ rounds. Then root broadcasts a start round $r_0 \in O(Diam(G))$, where all nodes simultaneously start the **for** loop. For APSP we have $S = V$. For $L_v = L_v(Diam(G), n) = L_v(2d, n)$, Lemma 43 shows that PipeBF has correct output. $\square$

7.2 APSP with Relabeling in Unweighted Graphs

Running time of PipeBF essentially best possible, even in trees with constant depth.
Problem: Bottleneck links with lots of messages to identify correct IDs in the subtree.

What if we can **relabel the graph** and assign routing labels (i.e., second set of IDs)? This could enable us to compute and represent routing tables more compactly.

APSP with relabeling: Every node u needs to output an **ID** $\lambda(u)$ and a **routing table** that, given any other ID $\lambda(v) \neq \lambda(u)$, delivers a neighbor of u on the shortest u-v-path.

As a warm-up let's consider trees.

Lemma 44. *Given a tree T, we can compute in $O(Depth(T))$ rounds for every node u and every node v the neighbor of u on a shortest u-v-path, and an upper bound on $\mathsf{dist}(u,v)$.*

Proof. We assign a unique ID label $idLabel(v) \in \{1,\ldots,n\}$ to every node and determine all routing tables. Idea: Enumerate tree nodes in preorder (DFS style). Distributed implementation:

1. For each v determine number of nodes in subtree T_v. Single upcast in time $O(Depth(T))$.
2. Root r_0 gets $idLabel(r_0) = 1$. Assigns each children v mutually disjoint consecutive intervals of $|T_v|$ integers in $\{2,\ldots,n\}$.
3. Recursive application: v takes first number from assigned interval, paritions rest in consecutive intervals for children
4. In the top-down recursion, the distance from root can also be tracked. Final node labels are pairs (ID label, distance to root).

Routing tables: Consecutive intervals for ID labels of children subtrees. Given query label $\lambda(v) = (idLabel(v), dist(v, r_0))$ at node u, route to child whose interval contains $idLabel(v)$. If such a child does not exist, route to parent. Upper bound on $\mathsf{dist}(u,v)$ is given by $\mathsf{dist}(u,r_0) + \mathsf{dist}(v,r_0)$. □

[Pic: Tree with label intervals, distance incorporated into node label]

For general graphs, consider approximate APSP (with relabeling). Idea:

- Determine a small set S of landmark nodes, sampled uniformly at random.
- Each node v learns: (1) closest landmark node $s_v \in S$, (2) next hop on short path for every node closer than some threshold value, (3) next hop on shortest path for every landmark $s \in S$ no matter how far away.
- Landmarks organize associated nodes via tree relabeling as in previous lemma
- Node label of v contains:
  ID of s_v, bit indicating $v \in S$, label for routing tree of s_v.

When $\mathsf{dist}(v,w)$ is small, v knows shortest path to w. Otherwise, w is far away, ok to route via landmark s_w (extracted from known label $\lambda(w)$)

[Pic: Landmarks and relabeling within V_s for every landmark s]

How to route messages from v to a node with label $\lambda(w)$:

Algorithm 14: 5-APSP

Determine n and $d \in [Diam(G), 2 \cdot Diam(G)]$, inform all nodes
Sample each node into landmark set $S \subseteq V$ independently w. prob. $c\sqrt{\log n / n}$
c is a sufficiently large constant
Determine $|S|$, inform all nodes
Add to each ID v a bit b_v indicating if $v \in S$
Use PipeBF with source set S to compute $L_v(d, |S|)$ for all $v \in V$
for each $v \in V$ set $s_v \leftarrow \arg\min_{s \in S} \mathsf{dist}(v, s)$
for each $s \in S$ compute labels $\lambda_s(v)$ for routing from/distances to root s of (partial) BFS tree with nodes $V_s = \{v \mid s_v = s\}$
Relabel each $v \in V$ by $\lambda(v) \leftarrow (v, s_v, \lambda_{s_v}(v))$
Use PipeBF with source set V to compute $L_v(\sqrt{n \log n}, \sqrt{n \log n})$ for all $v \in V$

1. $\lambda(w)$ contains: ID w (appended with a bit if $w \in S$), ID s_w, and tree label $\lambda_{s_w}(w)$. Tree label $\lambda_{s_w}(w)$ contains distance $\mathsf{dist}(s_w, w)$.
2. If there is entry $(\mathsf{dist}(v, w), w) \in L_v(\sqrt{n \log n}, \sqrt{n \log n})$ for sources V, then v knows $\mathsf{dist}(v, w)$ and the next hop on the shortest path.
3. If not, we route from v to s_w using $L_v(d, |S|) = L_v(Diam(G), |S|)$ and then from s_w to w using tree label $\lambda_{s_w}(w)$. Distance is estimated as $\mathsf{dist}(v, s_w) + \mathsf{dist}(s_w, w)$.

Every path is 5-approximate if for every v, the closest source s_v is close enough:

Lemma 45. *Suppose $(\mathsf{dist}(v, s_v), s_v) \in L_v(\sqrt{n \log n}, \sqrt{n \log n})$ for all $v \in V$, then algorithm 5-APSP computes a 5-approximate solution to APSP with relabeling.*

Proof. Consider a node v and a query node w.

- If $(\mathsf{dist}(v, w), w) \in L_v(\sqrt{n \log n}, \sqrt{n \log n})$, then distance and path are optimal.
- Otherwise, since $(\mathsf{dist}(v, s_v), s_v) \in L_v(\sqrt{n \log n}, \sqrt{n \log n})$ we know
  $\Rightarrow (\mathsf{dist}(v, s_v), s_v) \le (\mathsf{dist}(v, w), w)$
  $\Rightarrow \mathsf{dist}(v, s_v) \le \mathsf{dist}(v, w)$.
- Since s_w is source closest to w, we have $\mathsf{dist}(w, s_w) \le \mathsf{dist}(w, s_v)$.
- Use triangle inequality:

$$\begin{aligned}
\mathsf{dist}(v, s_w) + \mathsf{dist}(s_w, w) &\le \mathsf{dist}(v, w) + \mathsf{dist}(w, s_w) + \mathsf{dist}(s_w, w) \\
&\le \mathsf{dist}(v, w) + 2\,\mathsf{dist}(w, s_v) \\
&\le \mathsf{dist}(v, w) + 2(\mathsf{dist}(w, v) + \mathsf{dist}(v, s_v)) \\
&\le 5\,\mathsf{dist}(v, w)
\end{aligned}$$

□

The condition of the lemma and hence the 5-approximation holds w.h.p.

Lemma 46. *W.h.p. it holds $(\mathsf{dist}(v, s_v), s_v) \in L_v(\sqrt{n \log n}, \sqrt{n \log n})$ for all $v \in V$.*

Proof. Rather direct application of Chernoff bounds:

- S sampled uniformly at random with prob. $c\sqrt{\log n / n}$

- Consider any given set I of $\sqrt{n \log n}$ nodes, then $\mathbb{E}[|S \cap I|] \geq c \log n$
- Chernoff bound: $\Pr[|S \cap I| \leq c(\log n)/2] \leq e^{-\Omega(c \log n)} = n^{-\Omega(c)}$
- Nodes indicated by list $L_v(\sqrt{n \log n}, \sqrt{n \log n})$ is such a set I
- Hence, s_v contained in the list with probability $1 - n^{-\Omega(c)}$
- Choose constant c large enough and apply a union bound over all nodes $v \in V$.
- Joint event that all $(\mathsf{dist}(v, s_v), s_v) \in L_v(\sqrt{n \log n}, \sqrt{n \log n})$ occurs whp

$\square$

Remains to analyze time complexity of the algorithm. By previous lemma, the BFS trees rooted at nodes $s \in S$ are not too deep. Then we analyze the time complexity of each step in the algorithm.

Corollary 11. *W.h.p. the partial BFS trees rooted at nodes $s \in S$ containing nodes V_s all have depth $O(\sqrt{n \log n})$.*

Theorem 20. *Algorithm 5-APSP computes a 5-approximation for APSP with relabeling in unweighted graphs in $\mathsf{Time}(5\text{-}APSP, G) = O(Diam(G) + \sqrt{n \log n})$ w.h.p.*

Proof. Concentrate only on the steps with non-local computations:

- Compute global BFS tree T_B, set $d = 2Depth(T_B)$, inform nodes of n and d. Time: $O(Diam(G))$
- Determine $|S|$, inform all nodes. Time: $O(Diam(G))$
- $L_v(d, |S|)$ by PipeBF. $|S| \in O(\sqrt{n \log n})$ whp. Time: $O(Diam(G) + \sqrt{n \log n})$ whp
- Compute partial BFS trees and labels. Time $O(Diam(G) + \sqrt{n \log n})$ whp (by Lemma 44 and Corollary 11)
- List $L_v(\sqrt{n \log n}, \sqrt{n \log n})$ obtained by discarding any entry (d_w, w) with $d_w > \sqrt{n \log n}$ and truncating to (at most) $\sqrt{n \log n}$ elements. Time: $O(Diam(G) + \sqrt{n \log n})$

$\square$

7.3 Weighted Graphs

Reduction via Rounding Trick: Round the weights up to a power of $(1 + \varepsilon)$ for a fixed constant $0 < \varepsilon \leq 1$ to turn the instance into an "unweighted" instance.

Notation/Definition:

- $\omega_{max} = \max_{e \in E} \omega(e)$ maximum weight of any edge
- Fix a constant $0 < \varepsilon \leq 1$. We denote by $i_{max} = \lceil \log_{1+\varepsilon} \omega_{max} \rceil$. For $i = 0, 1, \ldots, i_{max}$,

$$\llbracket x \rrbracket_i = (1 + \varepsilon)^i \left\lceil \frac{\omega(e)}{(1 + \varepsilon)^i} \right\rceil$$

  is x rounded up to multiples of $(1 + \varepsilon)^i$.
- Rounded graph $G_i = (V, E, \omega_i)$: All weights rounded up to multiples of $(1 + \varepsilon)^i$, i.e., $\omega_i(e) = \llbracket \omega(e) \rrbracket_i$. Distance dist_i is distance in G_i wrt. rounded weights

Compare original distances in G and rounded distances in G_i.

- Let $\mathsf{hop}(v, w)$ be the number of edges on the shortest path between v and w, i.e., the "unweighted length" of the (weighted) shortest path. $\mathsf{dist}(v, w)$ and $\mathsf{hop}(v, w)$ can be very different.
- Obviously: $\mathsf{dist}_i(v, w) \geq \mathsf{dist}(v, w)$.
- As i increases, $(1+\varepsilon)^i$ gets bigger. Weights in G_i get more coarse-grained, more and more edge weights turn into the first multiple of $(1+\varepsilon)^i$
- As i increases $\mathsf{dist}_i(v, w)$ turns into $(1+\varepsilon)^i \cdot \mathsf{hop}(v, w)$
- There is a "sweet spot" such that (1) $\mathsf{dist}_i(v, w) \leq (1+\varepsilon)\mathsf{dist}(v, w)$, but also (2) $\mathsf{dist}_i(v, w)$ is roughly $(1+\varepsilon)^i \cdot \mathsf{hop}(v, w)$

Lemma 47. *For*

$$i(v, w) = \max\left\{0, \left\lfloor \log_{1+\varepsilon}\left(\frac{\varepsilon \cdot \mathsf{dist}(v, w)}{\mathsf{hop}(v, w)}\right)\right\rfloor\right\} \quad ,$$

we have

1. $\mathsf{dist}_{i(v,w)} \leq (1+\varepsilon) \cdot \mathsf{dist}(v, w)$
2. $(1+\varepsilon) \cdot \mathsf{dist}(v, w) \in O((1+\varepsilon)^{i(v,w)} \cdot \mathsf{hop}(v, w))$

Proof. If $i(v, w) = 0$, then $\mathsf{dist}_{i(v,w)}(v, w) = \mathsf{dist}(v, w)$. Otherwise

$$\begin{aligned}
\mathsf{dist}_{i(v,w)}(v, w) &\leq \mathsf{dist}(v, w) + (1+\varepsilon)^{i(v,w)} \cdot \mathsf{hop}(v, w) \\
&\leq \mathsf{dist}(v, w) + (1+\varepsilon)^{\log_{1+\varepsilon}\left(\frac{\varepsilon \cdot \mathsf{dist}(v,w)}{\mathsf{hop}(v,w)}\right)} \cdot \mathsf{hop}(v, w) \\
&\leq \mathsf{dist}(v, w) + \varepsilon \cdot \mathsf{dist}(v, w)
\end{aligned}$$

This proves part 1. For part 2.

$$\begin{aligned}
\mathsf{dist}(v, w) &= \frac{\mathsf{hop}(v, w)}{\varepsilon} \cdot \frac{\varepsilon \cdot \mathsf{dist}(v, w)}{\mathsf{hop}(v, w)} \\
&\leq \frac{\mathsf{hop}(v, w)}{\varepsilon} \cdot (1+\varepsilon)^{i(v,w)+1} \\
&\in O((1+\varepsilon)^{i(v,w)} \cdot \mathsf{hop}(v, w))
\end{aligned}$$

□

Theorem 21. *For any constant $\varepsilon > 0$, there is a distributed algorithm that computes a $(1+\varepsilon)$-approximation for APSP in $O(n \log n)$ rounds.*

Proof. Approach: Determine largest edge weight, locally round edge weights, apply PipeBF algorithm *sequentially for all rounded graphs* G_i, for $i = 0, \ldots, i_{\max}$.

Consider the graphs G_i:

- Replace edge e of weight $k \cdot (1+\varepsilon)^i$ by virtual path of k edges with weight 1.
- Results in unweighted graph $\tilde{G_i}$
- Suppose we apply PipeBF, let $L_{i,v}(h, k)$ the list for $\tilde{G_i}$
- By Lemma above: In $L_{i(v,w),v}(O(\mathsf{hop}(v, w)), n) = L_{i(v,w),v}(O(n), n)$ there is an entry (d, w) such that $(1+\varepsilon)^{i(v,w)} \cdot d \leq (1+\varepsilon) \cdot \mathsf{dist}(v, w)$.
- For every i we have

1. $\mathsf{dist}(v, w) \leq (1 + \varepsilon)^i \cdot d$
2. $(d, w) \in L_{i,v}(O(n), n)$ (since weights get rounded up)
3. $i(v, w) \leq i_{max}$ (since $\varepsilon \cdot \mathsf{dist}(v, w)/\mathsf{hop}(v, w) \leq \omega_{max}$)

- This implies

$$\begin{aligned}\mathsf{dist}(v, w) &\leq \min_{i=0,\ldots,i_{max}} \{(1 + \varepsilon)^i d \mid (d, w) \in L_{i,v}(O(n), n)\} \\ &\leq (1 + \varepsilon) \cdot \mathsf{dist}(v, w)\end{aligned}$$

Hence, if we run the algorithm for all i, we will find a distance value and a path that respresent a $(1 + \varepsilon)$-approximation for the true distance. The number of graphs G_i we need to consider is bounded logarithmically by the largest edge weight, which is by assumption:

$$i_{max} = \lceil \log_{1+\varepsilon} \omega_{max} \rceil \leq \log_{1+\varepsilon} n^c \in O(\log n)$$

Overall, $\tilde{G}_i$ can have up to n^{c+2} virtual nodes, but we need to run PipeBF only to compute the list entries (d, w) with $d \in O(n)$ for original nodes from G. Hence, PipeBF needs only $O(n)$ rounds. $\square$

Chapter 8

Packet Routing

(Synchronous) **Store-and-Forward Packet Routing**:

- Each packet p has a source node s_p and a target node t_p. Initially located at source s_p
- Full Duplex: Every round, an edge can send at most one packet in each direction
- Minimize number of rounds until last packet is delivered

Two Problems:

Path Selection Problem: Given a set of packets with sources and targets, select an s_p-t_p-path for every packet p

Packet Scheduling Problem: Given a set of packets and a collection $\mathcal{P}$ of paths (one for each packet), determine which packet to forward on which edge in which round.

Clearly, time to route packets critically depends on trivial bottlenecks

- Maximum distance of any chosen path for any packet p
- Maximum number of packets delivered from/to a single node

To uncover structural bottlenecks in the network, we focus on **Permutation Routing**:

- n packets, every node source of exactly one packet and target of exactly one packet
- Routing problem specified by a permutation $\pi : V \to V$: Packet at source node v to be delivered to target node $\pi(v)$

Let's consider a warm-up: **Permutation Routing on Mesh Networks**.

Mesh Network $M(\ell, d)$ (d-dimensional mesh of side length ℓ)

- $M(\ell, d) = (\{0, 1, ..., \ell - 1\}^d, E)$ with

$$E = \{\{a, b\} \mid \exists i \in \{0, 1, ..., d - 1\} : |a_i - b_i| = 1 \text{ and } a_j = b_j, \text{ for } j \neq i\}$$

- $M(\ell, 1)$: Path of ℓ nodes; $M(\ell, 2)$: $\ell \times \ell$ grid; $M(\ell, 3)$: $\ell \times \ell \times \ell$ cube
- $M(\ell, d)$: ℓ copies of $M(\ell, d - 1)$. The ℓ copies of the same node of $M(\ell, d - 1)$ form a path in dimension d.
- $n = \ell^d$ nodes, $d \cdot \ell^d - d \cdot \ell^{d-1}$ edges, diameter is $d \cdot (\ell - 1)$
- Observe: $M(2, d)$ is d-dimensional hypercube

A simple and attractive strategy: **Dimension-by-dimension routing**

- All packets are routed in parallel along the paths in dimension 0 to target node in dimension 0. Then all packets routed in parallel along the paths in dimension 1 to target node in dimension 1. Then dimension 2, 3, 4 etc.
- If several packets to be routed on one edge in same direction: **Farthest-first routing** – the packet that has the longest distance to target in the current dimension gets routed first.
- For hypercubes, this results in **bit-fixing paths**

[Pic: Example Grid, Hypercube]

Dimension-by-dimension routing is a simple and decentralized procedure for solving both path selection (repeatedly route along path of dimension i to target submesh, for $i = 0, 1, 2, \ldots$) and packet scheduling (farthest first) problems. It works well, however, only for small-dimensional meshes.

Lemma 48. *For every permutation routing problem on meshes $M(\ell, 1)$ and $M(\ell, 2)$, dimension-by-dimension routing terminates in $O(Diam(G))$ rounds. For mesh $M(\ell, 3)$ there are permutations such that dimension-by-dimension routing takes $\Omega(Diam(G)^2)$ rounds.*

Proof. Exercise. □

Centralized routing algorithms can be much faster. The following result relies on fundamental insights on mesh networks, sorting and matching to choose appropriate routing paths.

Theorem 22. *For any permutation routing problem on $M(\ell, d)$ there are centralized algorithms to solve path selection and packet scheduling such that the routing terminates in $O(\ell \cdot d) = O(Diam(G))$ rounds.*

8.1 Deterministic Oblivious Routing

Oblivious Routing:
- Routing with local control
- Path chosen for each packet depends only on its own source and target
- Does not depend on sources or destinations of other packets
- Specify a **path system** $\mathcal{W}$ that contains a path $P_{u,v}$ from u to v, for every pair $u, v \in V$.
- Every packet with source u and target v is sent along the path $P_{u,v}$
- Example: Dimension-by-dimension routing and bit-fixing paths – every routing path is entirely determined by IDs of source and target

Here paths given by $\mathcal{W}$ must be used for **every permutation routing problem** on the network – impossible to adjust paths **based on the permutation** (as in Theorem 22).

Theorem 23. *Let $G = (V, E)$ be any graph, Δ the maximum degree of any node in G, and $\mathcal{W}$ be any path system. Then there exists a permutation π and an edge $e^* \in E$ such that at least*

$$\sqrt{\frac{n}{2(\Delta^2)}} \quad = \quad \Omega\left(\frac{\sqrt{n}}{\Delta}\right)$$

of the paths selected by π from $\mathcal{W}$ contain e^*.

Very bad news about **deterministic oblivious routing**: For graphs with small degree, time is polynomial in n. Even a small diameter, say, logarithmic in n does not help.

Example d-dimensional Hypercube $M(2, d)$:
Theorem 22: Non-oblivious deterministic routing in $O(Diam(G)) = O(d) = O(\log n)$
Theorem 23: Deterministic oblivious routing needs $\Omega(\sqrt{n}/\Delta) = \Omega(2^{d/2}/d) = \Omega(\sqrt{n}/\log n)$

Proof of Theorem 23. Notation/Definition:

- For $v \in V$, let $\mathcal{W}_v = \{P_{v,u} \mid u \in V\}$ the set of all paths starting in v
- Consider $t > 0$, node $v \in V$ and edge $e \in E$. We say e **is t-popular for** v if at least t paths from $\mathcal{W}_v$ contain e.

Three Proof Steps:

1. Lemma 49: For any $v \in V$, there are "many" edges that are "quite popular" for v.
2. Using Lemma 49: There is $e^* \in E$ "quite popular" for many nodes, i.e., e^* is t-popular for t different nodes, with $t = \Omega(\sqrt{n}/\Delta)$.
3. Given this, construct permutation π such that t of paths selected by π contain e^*.

Step 1:
For $t > 0$, define a 0-1-matrix $A(t)$ with n rows and $|E|$ columns. For $v \in V$ and $e \in E$:

$$A_{v,e}(t) = \begin{cases} 1 & \text{if } e \text{ is } t\text{-popular for } v\text{, and} \\ 0 & \text{otherwise.} \end{cases}$$

- For $v \in V$ we denote the row sum of v by
$$A_v(t) = \sum_{e \in E} A_{v,e}(t)$$
- For $e \in E$ we denote the column sum of e by
$$A_e(t) = \sum_{v \in V} A_{v,e}(t)$$

Lemma 49. *For all $v \in V$ and $t \leq (n-1)/\Delta$ we have $A_v(t) \geq \frac{n}{2\Delta t}$*

Proof of Lemma: Consider nodes connected to v by "popular paths":

- $Q \subseteq V$ is the set of nodes to which there is a path from v that contains only edges that are t-popular for v.
- $L = V - Q$ and $B = E \cap (L \times Q)$
- B are edges connecting a node in Q (to which a path of t-popular edges exists) to a node in L.

[Pic: Schema Q, L and B]

Note that
$|L| \leq (t-1) \cdot |B|$: For each node $u \in L$, the path $P_{u,v}$ leads through at least one edge in B. These edges are not t-popular, each of them contained in at most $t-1$ paths from $\mathcal{W}_v$.
$|B| \leq \Delta |Q|$: Each node in Q has at most Δ incident edges

Combining this gives $\Delta|Q|(t-1) \geq |L| = n - |Q|$, which implies $\Delta|Q|t \geq n$ and, thus,

$$|Q| \geq \frac{n}{\Delta t}$$

If $|Q| \leq 2A_v(t)$, the lemma is proved, because

$$A_v(t) \geq \frac{|Q|}{2} \geq \frac{n}{2\Delta t} \quad .$$

Let E' be all t-popular edges for v. We will show $|Q| \leq 2|E'| = 2A_v(t)$.

Observe that the lemma requires $t \leq (n-1)/\Delta$. Then $E' \neq \emptyset$.

- v has at most Δ incident edges, $\mathcal{W}_v$ contains $n-1$ paths.
- At least one of the incident edges must be used by at least $(n-1)/\Delta$ paths from $\mathcal{W}_v$
- Hence, if t is so small, then there would be at least one t-popular edge.

There is at least one t-popular edge. Each node in Q is incident to an edge in E'. Every edge in E' is incident to at most two nodes from Q. Hence, $|Q| \leq 2|E'| = 2A_v(t)$. This proves the lemma. $\square$

Step 2:
Now show that there is e^* that is t-popular for t nodes, with $t = \Omega(\sqrt{n}/\Delta)$. Observe that

$$\sum_{e\in E} A_e(t) = \sum_{e\in E}\sum_{v\in V} A_{v,e}(t) = \sum_{v\in V}\sum_{e\in E} A_{v,e}(t) = \sum_{v\in V} A_v(t) \geq \frac{n^2}{2\Delta t}$$

The last step is due to Lemma 49. Pigeonhole principle implies there is $e^* \in E$ with

$$A_{e^*}(t) \geq \left\lceil \frac{n^2}{|E| \cdot 2\Delta t} \right\rceil \geq \left\lceil \frac{n}{2\Delta^2 t} \right\rceil .$$

The last step uses $|E| \leq \Delta n$.

Choose $t = \frac{n}{2\Delta^2 t}$, i.e., $t = \sqrt{n}/(\sqrt{2}\Delta)$. For any $n \geq 2$, we have $t \leq (n-1)/\Delta$, so Lemma 49 can be applied. Plugging in t into the above inequality, we see

$$A_{e^*}(t) \geq \left\lceil \frac{n}{2\Delta^2\sqrt{n}/(\sqrt{2}\Delta)} \right\rceil = \left\lceil \frac{\sqrt{n}}{\sqrt{2}\Delta} \right\rceil = \lceil t \rceil.$$

e^* is $\lceil t \rceil$-popular for $\lceil t \rceil$ nodes, where $t = \sqrt{n}/(\sqrt{2}\Delta)$.

Step 3:
Construct bad permutation π such that $\lceil t \rceil$ paths selected by π contain e^*:

- V' denotes set of $\lceil t \rceil$ nodes for which e^* is $\lceil t \rceil$-popular, wlog $V' = \{1, \ldots, \lceil t \rceil\}$
- For every $v \in V'$, there is subset $U_v \subseteq V$ with $|U_v| = \lceil t \rceil$ such that, for every $u \in U_v$ the path $P_{v,u}$ contains e^*
- For $v = 1$ to $\lceil t \rceil$ set $\pi(v) = u$ with u chosen arbitrarily from $U_v \setminus \{\pi(1), \ldots, \pi(v-1)\}$
- For $v = \lceil t \rceil$ to n set $\pi(v) = u$ with u chosen arbitrarily from $V \setminus \{\pi(1), \ldots, \pi(v-1)\}$

By construction, π and e^* satisfy properties postulated in the theorem. $\square$

8.2 Randomized Oblivious Routing

How to improve upon this? **Randomized Oblivious Routing**!
For every pair $u, v \in V$ of nodes:

- Path system $\mathcal{W}$ contains set $\mathcal{W}_{u,v}$ of paths from u to v
- Probability distribution $\mathcal{D}_{u,v} : \mathcal{W}_{u,v} \to [0, 1]$

For every packet to be routed from u to v choose the routing path $P_{u,v}$ from $\mathcal{W}_{u,v}$ by drawing independently at random from $\mathcal{D}_{u,v}$.

Example: For every pair of nodes there are two possible paths, i.e., $|\mathcal{W}_{u,v}| = 2$ for all $u, v \in V$. Consider uniform distributions $\mathcal{D}_{u,v}$. When sending a packet from u to v, each of the two possibel paths is chosen with probability $\mathcal{D}_{u,v}(P) = 1/2$.

We will design **path selection algorithms** and **packet scheduling policies**.
Simple Examples for scheduling policies:

- FCFS (first-come-first-serve)
- FTG (farthest-to-go)
- Random Rank (defined later)

Greedy (no-wait) policies: Packet p waits at round t before using next edge e on its path only because other packet p' is using e in round t. Then we say p is **delayed** by p' at e in round t.

Two important parameters for a collection of paths $\mathcal{P}$:
Dilation D of $\mathcal{P}$ is length (number of edges) on the longest path in $\mathcal{P}$.
Congestion C of $\mathcal{P}$ is maximum number of paths of $\mathcal{P}$ sharing the same edge (in the same direction).
For simplicity: Replace every undirected edge by two directed edges in opposite direction.
For (directed) edge e let $C(e)$ be number of paths from $\mathcal{P}$ using e. Then $C = \max_{e \in E} C(e)$.

Initial Observations:
Lower Bound: Every scheduling policy needs at least $\max\{C, D\} = \Omega(C + D)$ steps.

- There is a packet with path length D, needs at least D rounds to target
- There is an edge that needs to forward at least C packets, requires at least C rounds.

Upper Bound: Every greedy scheduling policy needs at most $C \cdot D$ steps.

- Each packet can be delayed for at most $C - 1$ rounds on each edge of the path.

8.2.1 Path Selection for the Hypercube

First we study **permutation routing on $\mathbf{M(2, d)}$**, i.e., the d-dimensional hypercube.

Path Selection with Valiant's Trick:

- For each packet p, pick an intermediate target node v_p **independently uniformly at random** from V.
- Route p from source s_p to intermediate target v_p via bit-fixing paths
- Route p from intermediate target v_p to target t_p via bit-fixing paths

Observe: Path selection according to the paradigm of randomized oblivious routing.

Simplify the analysis by analyzing two-step process:
Phase 1 : All packets routed from sources to intermediate targets
Phase 2 : All packets routed from intermediate targets to targets

Transforms "worst-case permutation routing problem" into two "random routing problems":
Phase 1 : Random destination nodes
Phase 2 : Random source nodes

We **analyze Phase 1**, same analysis can be done for Phase 2.

Lemma 50. *The congestion C in Phase 1 (Phase 2) is $O(\log n / \log\log n)$ whp.*

Proof. Consider e, an edge of dimension i, i.e., e flips i-th bit ($i = 0, \ldots, d-1$)

- $IN(e)$: set of nodes from which e is reachable by a bit-fixing path.
- $OUT(e)$: set of nodes that are reachable from e by a bit-fixing path.
- Sizes: $|IN(e)| = 2^i$ and $|OUT(e)| = 2^{d-i-1}$.
- Fix any node in $v \in IN(e)$.
- Path of packet starting at v contains e in Phase 1 $\Leftrightarrow$ intermediate target in $OUT(e)$.
- Since they are chosen uniformly at random

$$\Pr[v\text{'s packet traverses } e] = \frac{|OUT(e)|}{n} = \frac{2^{d-i-1}}{2^d} = 2^{-i-1} .$$

[Pic: Example Bit-Fixing Paths arriving at edge e]

Consider any subset $X \subseteq IN(e)$.

- $A(X, e)$ denotes event that paths of all packets starting from X contain e
- $C(e)$ random variable, congestion at edge e (i.e., number of paths containing e)
- Let $k \in \mathbb{N}_0$, then

$$\begin{aligned}
\Pr[C(e) \geq k] &= \Pr[\exists X \subseteq IN(e), |X| = k : A(X,e)] \\
&\underset{\text{Union Bound}}{\leq} \sum_{X \subseteq IN(e), |X|=k} \Pr[A(X,e)] \\
&= \sum_{X \subseteq IN(e), |X|=k} (2^{-i-1})^k \\
&= \binom{|IN(e)|}{k} \cdot (2^{-i-1})^k
\end{aligned}$$

Estimation of binomial coefficients using $\mathsf{e} = 2.71\ldots$ the Eulerian number:

$$\left(\frac{a}{b}\right)^b \leq \binom{a}{b} \leq \left(\frac{\mathsf{e} \cdot a}{b}\right)^b$$

This implies

$$\Pr[C(e) \geq k] \leq \binom{|IN(e)|}{k} \cdot (2^{-i-1})^k \leq \left(\frac{\mathsf{e} \cdot 2^i}{k}\right)^k \cdot (2^{-i-1})^k = \left(\frac{\mathsf{e}}{2k}\right)^k .$$

Congestion $C = \max\{C(e) \mid e \in E\}$, so

$$\begin{aligned}\Pr[C \geq k] &= \Pr[\exists e \in E : C(e) \geq k] \underset{\text{Union Bound}}{\leq} \sum_{e \in E} \Pr[C(e) \geq k] \\ &\leq |E| \cdot \left(\frac{\mathsf{e}}{2k}\right)^k \leq n^2 \left(\frac{1}{2}\right)^k .\end{aligned}$$

The last step uses $|E| \leq dn \leq n(n-1)$ and $\frac{\mathsf{e}}{2k} \leq \frac{1}{2}$, where we assume $k \geq 3$.

Now choose k large enough such that $\Pr[C \geq k] \leq n^{-\alpha}$ for constant $\alpha > 0$. In particular, with $k = \lceil(\alpha + 2)\log n\rceil \geq 3$ we get

$$\Pr[C \geq k] \leq n^2 \cdot 2^{-(\alpha+2)\log n} \leq n^2 n^{-(\alpha+2)} = n^{-\alpha}.$$

This shows $C = O(\log n)$ whp.
For $C = O(\log n / \log\log n)$ choose k more clever. With

$$k = \max\left\{\frac{\mathsf{e}}{2}\sqrt{d},\ 2(\alpha+2) \cdot \frac{d}{\log d}\right\} = O\left(\frac{\log n}{\log\log n}\right)$$

we get

$$\begin{aligned}\Pr[C \geq k] &\leq n^2 \cdot \left(\frac{\mathsf{e}}{2k}\right)^k \leq n^2 \cdot \left(\frac{1}{\sqrt{d}}\right)^k \leq n^2 \left(\left(\frac{1}{\sqrt{d}}\right)^{\frac{2}{\log d}}\right)^{(\alpha+2)d} \leq n^2 \left(\frac{1}{2}\right)^{(\alpha+2)d} \\ &= n^2 n^{-(\alpha+2)} = n^{-\alpha}.\end{aligned}$$

□

This shows that Valiant's trick gives low congestion for permutation routing problems.

More general routing: **h-relation**. Every node is source of at most h packets and target of at most h packets. Permutation routing is a 1-relation.

Lemma 51. *Using Valiant's Trick for routing an arbitrary h-relation on the hypercube, the congestion is $C = O(\log n + h)$ whp.*

Proof. Exercise. □

8.2.2 Packet Scheduling for the Hypercube

Using Valiant's trick, we can do good path selection on the hypercube. How to solve the packet scheduling problem?

Theorem 24. *Suppose we are given a set of packets and a set $\mathcal{P}$ of bit-fixing paths, one for each packet. Let C denote the congestion of $\mathcal{P}$. The RandomRank Protocol is a distributed, randomized scheduling policy that delivers all packets in time $O(C + \log n)$ whp.*

Together with Valiant's Trick, this shows

Algorithm 15: RandomRank Protocol

$R \leftarrow$ sufficiently large integer value (speficied below)
$r(p) \leftarrow$ rank of packet p, assigned indep. uniformly at random from $\{1, 2, \ldots, |R|\}$
repeat
  In every round, each node forwards as many packets along their paths as possible
  if *two or more packets want to enter edge e in round t* **then**
    if *two or more of these packets have smallest rank* **then**
      packet with smallest ID among the ones with smallest rank enters e
    else
      packet with smallest rank enters e
    Remaining packets wait until next round
until *until all packets reach target*

Corollary 12. *There is a distributed algorithm that routes any h-relation on the hypercube in time $O(h + \log n)$ whp.*

The proof of Theorem 24 uses a "witness structure".

A **delay sequence (DS)** of length s consists of

- a **delay path** $P = (e(1), \ldots, e(L))$, $1 \le L \le d$ with edges of decreasing dimension (like a bit-fixing path in reverse order)
- s numbers $\ell_1, \ldots, \ell_s \in \{1, \ldots, L\}$ with $\ell_1 \le \ell_2 \le \cdots \le \ell_s$
- $s + 1$ distinct **delay packets** $p_0, p_1, \ldots, p_s$ such that, for $1 \le i \le s$, edge $e(\ell_i)$ is contained in the paths of packets p_{i-1} and packet p_i
- $s + 1$ numbers $k_0, k_1, \ldots, k_s \in [R]$ with $k_0 \ge k_1 \ge \cdots \ge k_s$.

- A DS is **active** if $r(p_i) = k_i$ for $0 \le i \le s$.

We first show that a long execution of RandomRank gives rise to a long active DS.

Lemma 52. *If RandomRank needs $T > d$ steps, then there exists an active DS of length at least $T - d$.*

Proof. We construct a path by travelling backwards through time:

- Consider a packet p_0 arriving in round $T > d$. p_0 must have been delayed.
- Follow path of p_0 backwards through time to first edge e, where p_0 was delayed. Here a packet p_1 delays p_0.
- Follow path of p_1 to an edge (possibly still e) where p_1 was delayed. Here a packet p_2 delays p_1.
- Repeat. Finally, we reach packet p_s that was not delayed before. Follow p_s to source.
- Tour backward through time covers T steps. We saw s time steps where a packet got delayed. Let L be the number of edges on the recorded path.
- Every step: One more edge or observed delay. Thus $T = L + s$ and $s = T - L \ge T - d$.

Based on this path computed via reverse time-travel, we can now construct an active DS:

1. The sequence of edges we have recorded gives us the delay path $P = (e(1), \dots, e(L))$. Since we follow bit-fixing paths backwards in time, our sequence of edges (i.e., the reverse path) traverses edges in decreasing order of dimension.
2. Packets $p_0, \dots, p_s$ are the delay packets. By construction, they are distinct (Why?)
3. For $1 \leq i \leq s$, we choose $\ell_i \in \{1, \dots, L\}$ so that $e(\ell_i)$ is the edge on which p_{i-1} was delayed by p_i
4. Observe that both the paths of p_{i-1} and p_i traverse $e(\ell_i)$, and $\ell_1 \leq \ell_2 \leq \dots \leq \ell_s$.
5. For $0 \leq i \leq s$, we set $k_i = r(p_i)$. Observe this yields $k_0 \geq k_1 \geq \dots \geq k_s$ as packet p_{i-1} is delayed by packet p_i, and RandomRank prefers packets with small rank.

□

Let's get some more insights on active DS. We first count the number of active DS of some maximum length s.

Lemma 53. *The number of delay sequences of length at most s is at most*

$$n^2 \cdot \binom{d-1+s}{s} \cdot \binom{R+s}{s+1} \cdot C^{s+1} \ .$$

Proof. **Step 1:** Counting delay paths.

Delay path moves monotonically through dimension, i.e., corresponds to a bit-fixing path (in reverse order). Number of paths determined by number of distinct source/target nodes, at most $n(n-1) \leq n^2$ paths.

Step 2: Counting the ways to choose ℓ_i's and k_i's.

Bound the number to choose integers $\ell_1, \dots, \ell_s$ such that $\ell_1 \leq \ell_2 \leq \ldots \leq \ell_s \leq d$.

- Encode integers as binary strings:

$$0^{\ell_1-1}10^{\ell_2-\ell_1}10^{\ell_3-\ell_2}1\ldots10^{\ell_s-\ell_{s-1}}10^{d-\ell_s} \ .$$

- String contains s ones. Number of zeros is

$$\ell_1 - 1 + \left(\sum_{i=2}^{2}(\ell_i - \ell_{i-1})\right) + d - \ell_s = d - 1 \ .$$

- One-to-one mapping between ℓ_i's and binary strings with $d-1$ zeros and s ones. Number of such strings is

$$\binom{d-1+s}{s}$$

- Similarly: Number of ways to choose $k_0, \dots, k_s \in [R]$ such that $k_0 \geq k_1 \geq \ldots \geq k_s$ given by number of binary strings with $R-1$ zeros and $s+1$ ones:

$$\binom{R+s}{s+1}$$

Step 3: Counting the ways to choose delay packets.

Suppose delay path P and ℓ_i's are fixed.

- For each delay packet, we know an edge that is contained in its path: p_i uses edge $e(\ell_i)$ (for $1 \leq i \leq s$) and p_0 uses edge $e(\ell_1)$.
- Each edge contained in at most C paths of packets
- At most C possibilities to choose a packet p_i that goes through a given edge.
- At most C^{s+1} possibilities to choose all delay packets $p_0, \ldots, p_s$.

$\square$

In addition to counting, we also see that a long DS is quite unlikely to be active.

Lemma 54. *The probability that a given DS of length s is active is $(1/R)^{s+1}$.*

Proof. For every delay packet: Probability of rank k_i is $1/R$, since ranks chosen uniformly at random from $[R]$. Hence, probability all $s+1$ delay packets have prescribed rank is $1/R^{s+1}$, since ranks are chosen independently. $\square$

Finally, we compose the three lemmas to prove Theorem 24.

Proof of Theorem 24. Lemma 52 shows: Algorithm needs $T = d + s$ steps $\Rightarrow$ exists active DS with length at least s. Cut the sequence after packet p_s. This gives active DS of length *exactly* s.

$\mathcal{DS}(s)$ is set of all possible delay sequences of length s.

$$\begin{aligned}
\Pr[T \geq d+s] \quad &\leq \quad \Pr[\exists DS \in \mathcal{DS}(s) : DS \text{ is active}] \\
&\leq \quad \sum_{DS \in \mathcal{DS}(s)} \Pr[DS \text{ is active}] \\
&\underset{\text{Lemma 54}}{=} \quad \sum_{DS \in \mathcal{DS}(s)} \frac{1}{R^{s+1}} \\
&\underset{\text{Lemma 53}}{\leq} \quad n^2 \cdot \binom{d-1+s}{s} \cdot \binom{R+s}{s+1} \cdot \left(\frac{C}{R}\right)^{s+1}
\end{aligned}$$

Now use $\binom{a}{b} \leq 2^a$ and $\binom{a}{b} \leq \left(\frac{\mathsf{e}a}{b}\right)^b$ and derive

$$\begin{aligned}
\Pr[T \geq d+s] \quad &\leq \quad n^2 \cdot 2^{d-1+s} \cdot \left(\frac{\mathsf{e}(R+s)}{s+1}\right)^{s+1} \cdot \left(\frac{C}{R}\right)^{s+1} \\
&\leq \quad n^3 \cdot \left(\frac{2\mathsf{e}C(R+s)}{(s+1)R}\right)^{s+1} \quad .
\end{aligned}$$

Choosing $R \geq s$ yields $R + s \leq 2R$ and, thus,

$$\Pr[T \geq d+s] \quad \leq \quad n^3 \cdot \left(\frac{4\mathsf{e}C}{s+1}\right)^{s+1} \quad .$$

Now choose $s = \lceil \max\{8\mathsf{e}C, (\alpha+3)\ \log n\}\rceil - 1 = O(C + \log n)$. This gives

$$\Pr[T \geq d+s] \ \leq\ n^3 \cdot \left(\frac{1}{2}\right)^{s+1} \ \leq\ n^3 \cdot \left(\frac{1}{2}\right)^{(\alpha+3)\log n} \ \leq\ n^{-\alpha} \ .$$

With probability at least $1 - n^{-\alpha}$, RandomRank delivers all packets in at most $d + s - 1 = O(C + \log n)$ rounds. □

8.2.3 Packet Routing in General Networks

We route packets in arbitrary networks $G = (V, E)$.
Path Selection: Every packet p routed along a shortest s_p-t_p-path.
Packet Scheduling: GrowingRank (same as RandomRank, differs only in Lines 1 and 11)

Algorithm 16: GrowingRank Protocol

1 $R \leftarrow$ sufficiently large integer, being a multiple of D (dilation of chosen paths $\mathcal{P}$)
2 $r(p) \leftarrow$ rank of packet p, assigned indep. uniformly at random from $\{1, 2, \ldots, |R|\}$
3 **repeat**
4   In every round, each node forwards as many packets along their paths as possible
5   **if** *two or more packets want to enter edge e in round t* **then**
6   **if** *two or more of these packets have smallest rank* **then**
7   packet with smallest ID among the ones with smallest rank enters e
8   **else**
9   packet with smallest rank enters e
10   Remaining packets wait until next round
11   Every packet that moves over some edge increases its rank by R/D
12 **until** *until all packets reach target*

Theorem 25. *Suppose we are given a set of $N \geq n$ packets, and for each packet p a shortest s_p-t_p path in $\mathcal{P}$. Let C and D denote the congestion and dilation of $\mathcal{P}$, resp. The GrowingRank Protocol is a distributed, randomized scheduling policy that delivers all packets in time $O(C + D + \log N)$ whp.*

Observation/Notation:

- Initial rank at most R, at most D times forwarded, each time rank grows by R/D
- Final rank at most $2R$
- $r_e(p) \in [2R]$: rank of packet p in time steps, where p contends for being forwarded along edge e

Adapted definition of **delay sequence**. A delay sequence (DS) of length s consists of

- a **delay path** $P = (e(1), \ldots, e(L))$, for $1 \leq L \leq 2D$, where P is a path in G
- s numbers $\ell_1, \ldots, \ell_s \in \{1, \ldots, L\}$ with $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_s$

- $s+1$ distinct **delay packets** $p_0, p_1, \ldots, p_s$ such that, for $1 \leq i \leq s$, edge $e(\ell_i)$ is contained in the paths of packets p_{i-1} and packet p_i
- $s+1$ numbers $k_0, k_1, \ldots, k_s \in [2R]$ with $k_0 \geq k_1 \geq \cdots \geq k_s$.

- A DS is **active** if $r_{e(\ell_i)}(p_i) = k_i$ for $0 \leq i \leq s$ and $r_{e(\ell_1)}(p_0) = k_0$.

We conduct similar analysis steps as for the RandomRank Protocol.

Lemma 55. *If GrowingRank needs $T \geq 2D$ steps, then there exists a DS of length at least $T - 2D$.*

Proof. The proof is quite similar to the proof of Lemma 52 We again construct the path by travelling backwards through time:

- Consider a packet p_0 arriving in round $T > d$. p_0 must have been delayed.
- Follow path of p_0 backwards through time to first edge e, where p_0 was delayed. Here a packet p_1 delays p_0.
- Follow path of p_1 to an edge (possibly still e) where p_1 was delayed. Here a packet p_2 delays p_1.
- Repeat. Finally, we reach packet p_s that was not delayed before. Follow p_s to source.
- Tour backward through time covers T steps. We saw s time steps where a packet got delayed. Let L be the number of edges on the recorded path.

Based on this path computed via reverse time-travel, construct active DS:

1. Path we have recorded in reverse order is delay path $P = (e(1), \ldots, e(L))$.
2. Packets $p_0, \ldots, p_s$ are delay packets.
3. For $1 \leq i \leq s$, we choose $\ell_i \in \{1, \ldots, L\}$ so that $e(\ell_i)$ is the edge on which p_{i-1} was delayed by p_i
4. For $1 \leq i \leq s$, set $k_i = r_{e(\ell_i)}(p_i)$ and $k_0 = r_{e(\ell_1)}(p_0)$.

Exercise: Show that packets $p_0, \ldots, p_s$ are distinct, i.e., no packet appears more than once in the delay sequence. This is the only part of the analysis where we need to assume that paths of packets are **shortest paths in** G.

Observe $k_0 \geq k_1 \geq \cdots \geq k_s$ as ranks of delay packets non-increasing on our tour:

- Switch from p_i to p_{i+1} on the tour, then rank of p_{i+1} is not larger than rank of p_i (protocol prefers packets with smaller rank)
- Add edge to delay path and follow this edge, then rank of currently observed packet is decreased (by R/d) since we proceed backwards in time.

Remains to show: $L \leq 2D$ and $s \geq T - 2D$

- Final rank of p_0 is at most $2R$
- During travel backward in time, observed ranks non-increasing
- Add edge to delay path: Rank of current packet that we follow decreases by R/D
- Rank of p_s at source at most $2R - L \cdot (R/D)$.
- Ranks non-negative: $2R - LR/D \geq 0$, so $L \leq 2R/(R/D) = 2D$.
- $T = L + s \Rightarrow s = T - L \geq T - 2D$.

□

The next step is again to count the number of DS of a given length.

Lemma 56. *The number of delay sequences of length s is at most*

$$\binom{2D-1+s}{s} \cdot \binom{2R+s}{s+1} \cdot NC^s \ .$$

Proof. Similar to the analysis of Lemma 53 above, the number of ways to choose ℓ_i's and k_i's can be bounded by

$$\binom{2D-1+s}{s} \cdot \binom{2R+s}{s+1} \ .$$

Given ℓ_i's, count number of choices for delay packets and edges on delay path.

- N possibilities to choose packet p_0
- When p_0 fixed, start construction of delay path from $e(1)$ to $e(\ell_1)$ backwards from t_{p_0}
- Path of p_1 contains $e(\ell_1)$, so at most C possibilities to choose p_1
- When p_1 fixed, construct delay path up to $e(\ell_2)$
- Path of p_2 contains $e(\ell_2)$, so at most C possibilities to choose p_2
- And so on. Number of possibilities to choose delay packets and path: At most NC^s.

□

The last lemma again bounds the probability that a DS of a given length is active.

Lemma 57. *The probability that a DS of length s is active is at most* $({}^1\!/\!_R)^{s+1}$

Proof. Let $e(\ell_i)$ be the j-th edge on the path of packet p_i. Suppose rank of p_i at $e(\ell_i)$ is $k_i = k_i' + (j-1) \cdot R/D$. This happens with probability $1/R$ (if and only if initial rank is k_i'). Hence, probability that rank of p_i at $e(\ell_i)$ is k_i is at most $1/R$. Thus, probability that *all* $s+1$ delay packets have the given ranks k_i is at most $(1/R)^{s+1}$. □

Proof of Theorem 25. We assemble the insights from the lemmas as before.

$$\begin{aligned}
\Pr[T \geq 2D+s] &\leq \Pr[\exists DS \in \mathcal{DS}(s) : DS \text{ is active}] \\
&\leq \sum_{DS \in \mathcal{DS}(s)} \Pr[DS \text{ is active}] \\
&\leq \binom{2D-1+s}{s}\binom{2R+s}{s+1} NC^s R^{-(s+1)} \\
&\leq 2^{2D-1+s}\left(\frac{\mathsf{e}(2R+s)}{s+1}\right)^{s+1} NC^s R^{-(s+1)} \\
&\leq 2^{2D}\left(\frac{6C\mathsf{e}}{s+1}\right)^{s+1} N
\end{aligned}$$

The last step assumes $R \geq s$.

Finally, let $s = \lceil \max\{12\mathsf{e}C, (\alpha+1)\log N + 2D\}\rceil = O(C + D + \log N)$. This gives

$$\Pr[T \geq 2D+s] \ \leq\ 2^{2D} N \left(\frac{1}{2}\right)^{s+1} \ \leq\ 2^{2D} N \left(\frac{1}{2}\right)^{(\alpha+1)\log N + 2D} \ \leq\ N^{-\alpha} \leq n^{-\alpha}$$

using $n \leq N$. Thus, with probability at least $1 - n^{-\alpha}$, GrowingRank needs at most $2D + s - 1 = O(C + D + \log N)$ rounds. □

Chapter 9

Rumor Spreading

Simple protocols to broadcast a message/update/virus/rumor/etc. in a network.
Motivated, for instance, by

Lazy Updates in Distributed Databases: Dataset is mirrored at several locations in the network. Suppose an update happens at one node. How to spread the updates through the network with low message complexity?

Epidemics and Infection Processes: Message interpreted as a virus that spreads like an epidemic though a network. Model and understand (randomized) infection process and the resulting properties of the spreading

Rumor Spreading: Piece of news that spreads though an online social network. How long does it take to reach everyone?

Several protocols studied in the literature. Initially, a single node v_0 in G has a message.
In each round, ...

Push Protocol: ...every *informed node* that holds the message sends it to a neighbor chosen uniformly at random.

Pull Protocol: ...every *uninformed node* that has no message asks a neighbor chosen uniformly at random whether there are some news.

Push-Pull Protocol: ...every informed node applies Push; every uninformed node applies Pull.

We concentrate on the Push protocol. In each round, every informed node makes a **random phone call** to a neighbor to tell her the message.

How to measure the spreading time in a graph with n nodes?

1. Consider for each integer T the **probability that after T rounds all nodes are informed**. Here we assume a worst-case starting node v_0. We usually give a bound on T as a function of n so that $\Pr[\text{all informed at time } T] = 1 - o(1)$ (with asymptotics in n).
2. Consider the round T_{all} at which the last node gets informed. We sometimes consider bounds on the **expected time to inform all nodes** $\mathbb{E}[T_{all}]$. Again, we assume a worst-case starting node.

9.1 Stars and Cliques

Theorem 26. *For the n-node star graph, the expected number of rounds to inform all nodes is $\mathbb{E}[T_{all}] = \Omega(n \log n)$. For the n-node complete graph, all nodes are informed after $T = O(\log n)$ rounds with probability $1 - o(1)$.*

Proof. **Star Graph:** Exercise.

Complete Graph:
Completely symmetric, cluster time into phases based on the number of informed nodes:
Phase 1: 0 to $x_1 = n^{0.4}$ informed nodes. True doubling: Whp all calls reach an uninformed node. Number of informed nodes doubles in each round ($\rightarrow$ Birthday Paradox)
Phase 2: x_1 to $x_2 = n/(\log\log n)^2$ informed nodes. Exponential growth: Each call still has probability $1 - o(1)$ to reach an informed node. Number of informed nodes almost doubles, increases by factor $2 - o(1)$ each round
Phase 3: x_2 to $x_3 = n \cdot (1 - 1/(\log\log n)^2)$ informed nodes. Short intermediate phase.
Phase 4: x_3 to n informed nodes. Exponential shrinking of uninformed nodes by factor $\mathsf{e} - o(1)$ each round ($\rightarrow$ Coupon Collection)

Assumption: Nodes call random neighbor including themselves.
Slows down process, adds symmetry, eases writing ($1/n$ instead of $1/(n-1)$)

Phase 1: Exact doubling

- Consider the first x_1 calls. During each such call, at most x_1 informed nodes.
- Pr[call reaches informed node] $\leq x_1/n$
- Union Bound: Pr[$\exists$ call that reaches informed node] $\leq x_1 \cdot x_1/n = 1/n^{0.2} = o(1)$.
- With probability $1 - 1/n^{0.2}$, the first $x_1 = n^{0.4}$ calls all reach uninformed nodes.
- Every call informs a new node $\Rightarrow$ Number of informed nodes doubles in every round.
- Let $x_1 = 2^{T_1} - 1$, then $T_1 = \log_2(n^{0.4} + 1) = (0.4 + o(1)) \log n$

With probability $1 - o(1)$, Phase 1 takes $T_1 = (0.4 + o(1)) \log n$ rounds.

Phase 4: Exponential shrinking of uninformed nodes

- For each round, at least $x_3 = n - o(n)$ informed nodes, so at least x_3 calls
- Consider fixed uninformed node v.

$$\Pr[v \text{ still uninformed after } T_4 \text{ rounds in Phase 4}] \leq \left(1 - \frac{1}{n}\right)^{x_3 T_4} \leq \mathsf{e}^{-x_3 T_4/n} \leq \frac{1}{n}$$

  for $T_4 = (n/x_3) \ln n = (1 + o(1)) \ln n$.
- Note: $1 + x \leq e^x$ for all $x \in \mathbb{R}$.
- Union Bound: Pr[$\exists$ uninformed node after T rounds] $\leq (n - x_3) \cdot 1/n = n/(\log\log n)^2) \cdot 1/n = o(1)$

With probability $1 - o(1)$, Phase 4 takes $T_4 = (1 + o(1)) \ln n$ rounds.

Phase 3: Short intermediate phase

- For each round, at least $x_2 = n \cdot f$ informed nodes, where $f = o(1)$.

- Consider $T_3 = 1/(f \cdot g)$ rounds, where $g = o(1)$. Consider fixed uninformed node v.

$$\Pr[v \text{ still uninformed after } T_3 \text{ rounds in Phase 3}] \leq \left(1 - \frac{1}{n}\right)^{x_2 T_3} \leq \mathsf{e}^{-x_2 T_3/n} = \mathsf{e}^{-1/g}$$

- Expected number of uninformed nodes after T_3 rounds at most $n \cdot \mathsf{e}^{-1/g}$
- Markov inequality:
  $\Pr[\text{more than } ng \text{ uninformed nodes after } T_3 \text{ rounds}] \leq n\mathsf{e}^{-1/g}/ng = \frac{1/g}{\mathsf{e}^{1/g}} = o(1)$.
- Note that f and g depend on choice of x_2 and x_3. We choose $f = g = 1/(\log\log n)^2$.
- Hence, $x_2 = n/(\log\log n)^2$ and $x_3 = n - n/(\log\log n)^2$

With probability $1 - o(1)$, Phase 3 takes $T_3 = (\log\log n)^4$ rounds.

Phase 2: How long does it take from $x_1 = n^{0.4}$ to $x_2 = n/(\log\log n)^2$ informed nodes? Number of informed nodes "almost" doubles in each round. Reasons for inexact doubling: Call goes to an already informed node, two nodes call the same informed node.

- Consider single round in Phase 2. Suppose there are i informed nodes.
- Order the i informed nodes and assume they make their calls sequentially
- Indicator variable:

$$X_k = \begin{cases} 1 & k\text{th node calls uninformed node} \\ 0 & \text{otherwise} \end{cases}$$

- At the end of round i, number of newly informed nodes is $X = \sum_{k=1}^{i} X_k$
- Since X_k are indicator variables

$$\mathbb{E}[X] = \sum_{k=1}^{i} \mathbb{E}[X_k] = \sum_{k=1}^{i} \Pr[X_k = 1] \geq \sum_{k=1}^{i} \frac{n - i - (k-1)}{n} \geq i\left(1 - \frac{3i}{2n}\right)$$

Single round with i informed nodes ends with expected number of at least $i\left(2 - \frac{3i}{2n}\right)$ nodes.

Thought experiment: An "ideal" Phase 2.
Every round always ends having exactly the expected number of informed nodes.

- Ideal Phase 2a: From $n^{0.4}$ to $n/(\log n)^2$ informed nodes
- Each round, number of informed nodes grows by factor at least $2\left(1 - \frac{3}{4(\log n)^2}\right)$
- Consider $T_{21} = \frac{4}{3} \cdot \log(n/x_1)$ rounds. Then number of informed nodes grows to

$$i\left(2\left(1 - \frac{3}{4(\log n)^2}\right)\right)^{T_1} \geq n\left(1 - \frac{3T_1}{4(\log n)^2}\right) \geq n/(\log n)^2$$

  where we use the Bernoulli inequality: $(1-x)^t \geq 1 - xt$
- Ideal Phase 2b: From $n/(\log n)^2$ to $n/(\log\log n)^2$ informed nodes
- Similar argument as above replacing (roughly) all $\log n$ by $\log\log n$
- $T_{22} = 2\log\log n$ rounds do the job.

The ideal Phase 2 needs $\frac{4}{3} \cdot \log n^{0.6} + 2\log\log n$ rounds.

Consider the changes to this ideal process. The probability that a change happens in any one of the rounds is small. We apply the following standard concentration result.

Theorem 27 (Azuma, McDiarmid, Hoeffding,...)**.** *Let $X_1, \ldots, X_m$ be independent random variables taking values in some sets $A_1, \ldots, A_m$. Let $f : A_1 \times \cdots \times A_m \to \mathbb{R}$ with $|f(a) - f(b)| \le c_j$ whenever a and b differ only in the jth component. Then*

- $\Pr[f(X_1, \ldots, X_m) \le \mathbb{E}[f(X_1, \ldots, X_m)] - \lambda] \le \exp(-2\lambda^2 / \sum_{j=1}^m c_j^2)$
- $\Pr[f(X_1, \ldots, X_m) \le \mathbb{E}[f(X_1, \ldots, X_m)] + \lambda] \le \exp(-2\lambda^2 / \sum_{j=1}^m c_j^2)$

Use this theorem as follows:

- Independent random variables: Our indicator variables X_k for successful calls
- Sets $A_k = \{0, 1\}$.
- $f = \sum_{k=1}^i X_k$ is the number of newly informed nodes after the round.
- Clearly, f differs by at most $c_k = 1$ when result of call X_j changes
- Apply theorem with $m = i$, all $c_k = 1$:

$$\Pr[X \le \mathbb{E}[X] - i^{0.75}] \le e^{-2\sqrt{i}}$$

Now apply concentration to see that it's very unlikely to take much longer than the ideal process. We only sketch the proof:

- Using $i \le n/(\log\log n)^2$ we have for $n \ge 2^{2^3}$ that $3i/2n \le 1/2$ and $\mathbb{E}[X] \ge i/2$.
- Then,

$$\Pr[X \le \mathbb{E}[X](1 - 2i^{0.25})] \le \Pr[X \le \mathbb{E}[X] - i^{0.75}] \le \mathsf{e}^{-2\sqrt{i}} \le \mathsf{e}^{-2n^{0.2}}$$

  where we use $i \ge n^{0.4}$ in the end
- Union Bound: With probability $1 - o(1)$ we have $X \ge \mathbb{E}[X](1 - 2i^{0.25})$ in all of the $2\log n$ rounds starting with at most $n/(\log\log n)^2$ informed nodes.
- With probability $1 - o(1)$ we have the good case, in which the ideal phase 2 applies – only progress dimished by a factor $1 - 2i^{0.25}$ in every round.
- When the phase runs for a logarithmic number of rounds, total progress in informed nodes dimished to a factor of $(1 - 2n^{-0.1})^{\Theta(\log n)} \ge 1 - o(1)$.
- Hence, we need to run the process a bit longer the overcome this loss in progress. The mulitplicative growth in one additional round, however, directly recovers this small multiplicative loss.

With probability $1 - o(1)$, Phase 2 takes at most $\frac{4}{3} \cdot \log n^{0.6} + 2\log\log n + 1$ rounds. □

9.2 General Graph Topologies

Let's consider more general network topologies. The complete graph and star graph are indeed the extreme network topologies for the Push protocol.

Theorem 28. *Let T be an integer such that in round T of the Push protocol with probability $1 - 1/n$ every node is informed. For every connected graph G it holds $T = O(n \log n)$ and $T = \Omega(\log n)$.*

Proof. **Upper Bound**:
Consider a shortest path $P = (v_0, v_1, \ldots, v_k)$ to some node v_k.

- Each round: Probability that v_i informs v_{i+1} is $1/\deg(v_i)$.
- Expected number of rounds until v_i informs v_{i+1} at most $\deg(v_i)$
- Expected number of rounds until v_k gets informed at most $\sum_{i=0}^{k-1} \deg(v_i)$
- $w \notin P$ connected to at most three nodes v_i, v_{i+1} and v_{i+2} (since P shortest path).
- Thus, $\sum_{i=0}^{k-1} \deg(v_i) \leq 3n$.
- For each node $u \neq G$, expected time until u informed is at most $3n$.
- Markov inequality: $\Pr[u$ uninformed after $6n$ rounds $] \leq 1/2$.

[Pic: Path]

We say phase 1 are rounds $1, \ldots, 6n$. If u remains uninformed after phase 1, overestimate the time by assuming that the whole process starts at v_0 again.

- Phase 2 is rounds $6n+1, \ldots, 12n$. We assume rumor again starts at v_0.
- Same analysis as above:
  $\Pr[u$ uninformed after $12n$ rounds $\mid u$ uninformed at the end of round $6n] \leq 1/2$.
- Thus, $\Pr[u$ uninformed after $12n$ rounds$] \leq 1/4$.
- Repeat. $\Pr[u$ uninformed after $i \cdot 6n$ rounds$] \leq 1/2^i$.
- With $i = 2\log n$: $\Pr[u$ uninformed after $12n \log n$ rounds$] \leq 1/n^2$
- Union bound: $\Pr[\exists u$ uninformed after $12n \log n$ rounds$] \leq 1/n$

Hence, after $T = 12n \log n$ nodes, all nodes are informed with probability at least $1 - 1/n$.

Lower Bound: Exercise. □

More generally: Same approach shows bounds wrt. maximum degree Δ and diameter $Diam(G)$.

Lemma 58. *Let $P = (v_0, v_1, \ldots, v_k)$ be any path of length k in G and $\Delta = \max_{i=0,1,\ldots,k} \deg(v_i)$ be the maximum degree of vertices in P. For any $k' \geq k$, after $2k'\Delta$ rounds the whole path is informed with probability at least $1 - e^{-k'/4}$.*

Proof. Consider modified process:
Every round, each informed node v_i calls v_{i+1} with probability *exactly* $1/\Delta$.

- Modified process obviously slower to inform v_k than real process
- $i(t)$ is largest index j such that $v_0, \ldots, v_j$ are informed at start of round t
- Define random variable X_t in round t:
  If $i(t) < k$ and $v_{i(t)+1}$ becomes informed in round t, then set $X_t = 1$.
  If $i(t) = k$ (i.e. all nodes informed), then set $X_t = 1$ with probability $1/\Delta$, independently from all other random decisions.
  In all other cases, set $X_t = 0$.
- All X_t are independent and satisfy $\Pr[X_t = 1] = 1/\Delta$.
- Consider $X = \sum_{t=1}^{T} X_t$. Observe: v_k informed after T rounds $\Leftrightarrow X \geq k$
- Note $\mathbb{E}[X] = T/\Delta = 2k'$
- Use Chernoff bounds: v_k still uninformed after T rounds with probability at most

$$\Pr[X < k] \quad \leq \quad \Pr[X < k'] \quad = \quad \Pr\left[X < \frac{1}{2} \cdot \mathbb{E}[X]\right] \quad \leq \quad e^{-\mathbb{E}[X]/8} \quad = \quad e^{-k'/4}$$

□

Use the lemma for bounds based on diameter and maximum degree.

Theorem 29 (Degree-Diameter Bound). *Let T be an integer such that in round T of the Push protocol with probability $1-1/n$ every node is informed. For every connected graph G it holds $T = O(\Delta \cdot \max\{Diam(G), \log n\})$.*

Proof. For every vertex v, consider shortest path from v_0 to v.

- Apply previous lemma with $k' = \max\{Diam(G), 8 \ln n\}$.
- This implies v gets informed after $T \leq 2k'\Delta = O(\Delta \cdot \max\{Diam(G), \log n\})$ rounds with probability $1 - e^{-k'/4} \geq 1 - 1/n^2$.
- Thus, probability that v is uninformed after T rounds is at most $1/n^2$.
- Apply union bound: Probability that at least one v is uninformed after T rounds is at most $1/n$.

□

Some Applications:

***k*-ary trees** Tree where every internal node has exactly k children.

- Diameter is $Diam(G) = O(Depth(T))$
- Maximum degree is $\Delta = k + 1$
- Number of nodes is $n = O(k^{Depth(T)})$.

Degree-Diameter Bound: $T = O(k \max\{Depth(T), \log n\}) = O(Depth(T) \cdot k \log k)$ rounds, all nodes informed w. prob. at least $1 - 1/n$

Mesh $M(\ell, d)$

- Diameter is $Diam(G) = d(\ell - 1)$
- Maximum degree $\Delta = 2d$
- Number of nodes is $n = \ell^d$.

Degree-Diameter Bound: $T = O(d \max\{d(\ell-1), \log n\}) = O(d^2\ell)$ rounds, all nodes informed w. prob. at least $1 - 1/n$.

Hypercube Degree-Diameter Bound: $T = O(\log^2 n)$ rounds, all nodes informed w. prob. $1 - 1/n$. Can be improved via different analysis to $O(\log n)$ rounds.

9.2.1 Random Geometric Graphs $G(n, r)$

Stylized model for wireless sensor networks detecting events in an area.

- Area modeled by unit square $[0,1]^2$
- Sensors are nodes: $v_1, \ldots, v_n \in [0,1]^2$ chosen uniformly at random
- Edges: $\{v_i, v_j\} \in E$ if and only if $dist(v_i, v_j) \leq r$. Equivalently: Put two disks with radius $r/2$ centered at v_i and v_j. Edge if and only if disks intersect.
- There is a **well-connected regime:** If $r \geq C\sqrt{(\ln n)/n}$, for C large enough, then G is connected w.h.p.

[Pic: Schema]

How long to spread a rumor via the Push protocol in a well-connected $G(n, r)$?
Analysis technique: Discretize the unit square.

- Partition unit square $[0, 1]^2$ into $\Theta(1/r^2)$ squares of side length $\ell = r/(2\sqrt{2}) = \Theta(r)$
- Two squares are adjacent if they touch (vertical, horizontal, diagonal)
- Note: vertices in adjacent squares are adjacent in G

[Pic: Grid partition of unit square]

Claim: Each square S contains a similar number of vertices.

- Number X^S of vertices in S is sum of independent binary random variables X_i with $\Pr[X_i = 1] = \ell^2 = \Theta(r^2) \geq C^2(\ln n)/n$.
- Hence, $\mathbb{E}[X^S] = n\ell^2 \geq C^2 \ln n$ and $\Pr[|X^S - \mathbb{E}[X^S]| \geq 0.25\mathbb{E}[X^S]] \leq n^{-2}$ when C sufficiently large.

This implies several properties in the well-connected regime (all hold whp):

Diameter is $O(1/r)$: Graph of squares has diameter $\Theta(1/r)$, each square contains at least one vertex, there is an edge between any two vertices in neighboring squares.

All degrees in $O(nr^2)$: All vertices in the (usually) 8 neighboring squares are neighbors, all neighbors lie in the (usually) 48 squares of distance at most 3. Hence, $\deg(v)$ is sum of a constant number of X^S

Degree-Diameter Bound: All nodes informed after time $T = O(nr)$ w. prob. at least $1 - 1/n$

Better Bound: Two-Stage Argument

- Square is informed: Contains at least one informed vertex. Rumor spreads among squares similar to 2D grid $M(\ell, 2)$ (gives time $O(1/r)$)
- When all squares informed, argue that inside every square rumor spreads similar to a clique (gives time $O(\log n)$)

The following result is given without proof.

Theorem 30. *Let T be an integer such that in round T of the Push protocol with probability $1 - 1/n$ every node is informed. A random geometric graph in the well-connected regime is structured w.h.p. such that $T = O(Diam(G) + \log n)$.*

9.2.2 Preferential Attachment Graphs

Very popular class of random graphs, captures some special properties of real-world networks (e.g., social networks, citation networks, etc.)

- Small diameter
- Non-uniform degree distribution
- Few nodes high degree (hubs), many nodes small (constant) degree
- Power Law: Number of nodes of degree d is proportional to $d^{-\beta}$, where β a constant, often in [2,3]

In the formal model, we use n as number of vertices, vertex set $\{v_1, \ldots, v_n\}$, and a constant density parameter $m \geq 2$. **PA graph** G^n is recursively defined:

- G^1 single vertex with m self-loops
- G^n obtained from G^{n-1} by adding new vertex v_n
- One after another, the new vertex chooses m neighbors
- Probability that some vertex v_x is chosen is proportional to
  - current degree of v_x, if $v_x \neq v_n$
  - "1 + the current degree of v_x" if $v_x = v_n$

  (self-loop probability takes into account current edge starting in v_n)

[Pic: PA Graph construction]

Properties of PA-Graphs (w.h.p., for constant $m \geq 2$):

- Diameter $\Theta(\log n / \log\log n)$
- Power law degree distribution: For $d \leq n^{1/5}$, expected number of vertices having degree d is proportional to d^{-3}
- Clustering coefficient (roughly probability that two neighbors of some node are connected by an edge) is $\Theta(1/n)$

Rumor Spreading Results:

- Push-Protocol: After n^α rounds with α a small constant, with constant probability there is still an uninformed node
- Push-Pull-Protocol: $\Theta(\log n)$ rounds, then all nodes informed with probability $1 - 1/n$.
- Can be improved to $\Theta(\log n / \log\log n)$ when contacts are chosen excluding the neighbor contacted in most recent round.

Chapter 10

Wireless Networks

Wireless networking has been a great success story over the last decades. From a distributed computing perspective, in some sense easier to analyze than general message passing systems: Nodes are often computationally restricted devices; also, they cannot form arbitrary network topologies, are restricted w.r.t. underlying geometry of the problem. On the other hand, wireless networks create additional challenges – no individual communication between nodes, **collision and interference problems**.

We study medium access control (MAC) protocols for channel access. Basic questions:

- How long until the first node has successfully sent a message? (Leader Election)
- How long until all nodes have sent at least one message? (Coloring)
- How to maximize the number of messages sent in a single round? (Maximum Independent Set)

10.1 Leaders, Initialization and the ALOHA Protocol

We first concentrate on a very simple network model:

- There are n devices, all located close to each other.
- Each device can decide in each round to (1) transmit or (2) listen and not transmit.
- If two or more devices decide to transmit in the same round, **they interfere with each other**. We call this case a **collision**.
- Transmissions that collide are unsuccessful. If a node transmits alone, it is successful.

How to find a leader, i.e., how long until **a single node can transmit alone**?

Easy if nodes have IDs – each node v simply waits $ID(v)$ many rounds until it transmits. Depending on the IDs this might be very slow.

Algorithm 17: Slotted ALOHA

```
repeat
    In each round, transmit with probability 1/n
until a node has transmitted alone
```

Lemma 59. *For $n \to \infty$, the expected number of rounds until the ALOHA protocol allows one node to transmit alone (i.e., become a leader) is* e.

Proof. Let X be the number of transmitting nodes. Probability that in a single round a node transmits alone:

$$\Pr[X = 1] = n \cdot \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1} \to \frac{1}{\mathsf{e}}$$

for $n \to \infty$. Hence, expected time until this happens is e. □

Remarks:
- Origin of the name: ALOHAnet, developed at the University of Hawaii.
- How does the leader know that it is the leader? "Distributed acknowledgment": Nodes continue ALOHA, including ID of leader in their transmission. Next time a node transmits alone, the leader learns that it is the leader.
- Then node v that managed to transmit the acknowledgment (alone) is the only remaining node which does not know that the leader knows that it is the leader. Now the leader can acknowledge v's successful acknowledgement.
- Unslotted time model: Two messages that overlap partially will interfere and no message is received. ALOHA also works here, with a factor 2 penalty, i.e., the probability for a successful transmission will drop from $1/\mathsf{e}$ to $1/(2\mathsf{e})$. Essentially, each slot is divided into t small time slots with $t \to \infty$ and whenever a node is not transmitting, it starts a new t-slot long transmission with probability $1/(2nt)$.

10.1.1 Initialization

A more involved task is **Initialization**: Number nodes from 1 to n.

Lemma 60 (Non-Uniform Initialization)**.** *If all nodes know n, the expected number of rounds for initialization is in $O(n)$.*

Proof. Simply apply ALOHA to elect a leader repeatedly. Each leader election step takes expected $O(1)$ time. □

What if nodes do not know n? We first assume nodes can do **collision detection**:
- Two or more nodes that transmit simultaneously are called a **collision**
- A **receiver** that hears the collided signals cannot detect any message. Thus, the channel appears like regular noise without any transmitted signal.
- A receiver **with collision detection** knows when a collision occured and can distinguish regular noise (no transmission) from collision (two or more transmissions).

Consider **Algorithm InitCD**:
- Iteratively apply a binary partition to the nodes and build a binary tree until only a single node in a partition remains.
- Once single node is identified, it receives the next free initialization number.

- Line 14: Transmitting node needs to know whether it was the only one transmitting. Achievable via acknowledgement rounds: Insert an additional acknowledgement round for each of the rounds $r+1$ and $r+2$. To notify v that it has transmitted alone in round $r+1$, every silent node transmits in the ack-round of $r+1$, while v is silent. If v hears message or collision in ack-round of $r+1$, it knows it transmitted alone in round $r+1$. Similarly for round $r+2$.

Algorithm 18: InitCD

```
1  nextInitNum ← 0, myBitstring ← 'x'
2  bitstringsToSplit ← ['x']                                  // Queue
3  while bitstringsToSplit is not empty do
4      b ← bitstringsToSplit.pop()
5      repeat
6          if b = myBitstring then
7              choose r uniformly at random from {0, 1}
8              In the next two rounds: Transmit in round r + 1, listen in other round
9          else
10             just listen in both rounds
11     until there was at least 1 transmission in both slots
12     if b = myBitstring then myBitstring ← myBitstring + r
13     for r ∈ {0, 1} do
14         if some node u transmitted alone in round r + 1 then
15             ID(u) ← nextInitNum, u becomes passive
16             nextInitNum ← nextInitNum + 1
17         else
18             bitstringsToSplit.push(b + r)
```

Theorem 31 (Uniform Initialization)**.** *Algorithm InitCD with collision detection correctly initializes n nodes in expected time $O(n)$.*

Proof. Consider a **successful split**: A split in which both subsets are non-empty.

- We build a binary tree with n leaves and $n-1$ inner nodes. Hence, there must be exactly $n-1$ successful splits. If we always make successful splits, we need $O(n)$ time.
- Problematic are unsuccessful splits. Then the repeat loop must be executed again.
- In an unsuccessful split of a set of size $k \geq 2$, there are 0 or k nodes transmitting in round $r+1$. The probability is

$$\Pr[X \in \{0,k\}] = \frac{1}{2^k} + \frac{1}{2^k} \leq \frac{1}{2}.$$

Thus, in expectation we have only $O(1)$ unsuccessful splits until a successful split occurs. □

For initialization without collision detection:

- First elect a leader ℓ. Suppose set S wants to transmit.
- Split every round in Algorithm InitCD in two rounds
- Use leader to distinguish between silence and noise
- First round: Every node from S transmits; Second round: $S \cup \{\ell\}$ transmits.
- This gives enough information to distinguish all cases. In the following table, X is silence/noise/collision, and $\surd$ is a successful transmission

	nodes in S transmit	nodes in $S \cup \{\ell\}$ transmit
$\lvert S\rvert = 0$	X	$\surd$
$\lvert S\rvert = 1, S = \{\ell\}$	$\surd$	$\surd$
$\lvert S\rvert = 1, S \neq \{\ell\}$	$\surd$	X
$\lvert S\rvert \geq 2$	X	X

Indeed, this implies that in general we can replace the assumption of collision detection with the assumption that a leader exists. In particular, this shows

Corollary 13. *Given a leader ℓ, Algorithm InitCD can be implemented without collision detection to correctly initialize n nodes in expected time $O(n)$.*

10.1.2 Leader Election

Given that we can omit collision detection if we have computed a leader, let us return to the leader election problem. The ALOHA protocol also delivers a whp guarantee:

Lemma 61. *The ALOHA protocol elects a leader in $O(\log n)$ rounds w.h.p.*

Proof. Exercise. □

What about uniform leader election, i.e., when the nodes do not know n?

Algorithm 19: UniformLeadElect

```
for k = 1, 2, 3, ... do
    for i = 1 to c · k do
        Transmit with probability p = 1/2^k
        if v was the only node which transmitted then v becomes the leader
```

Theorem 32. *If n is unknown, Algorithm UniformLeadElect can be used to elect a leader in time $O(\log^2 n)$ w.h.p.*

Proof. Nodes transmit with probability 2^{-k} for $c \cdot k$ rounds, for $k = 1, 2, 3, \ldots$.

- First p will be too high, lots of collisions
- When $k \approx \log n$, then nodes transmit with probability approximately $1/n$
- For simplicity let $n = 2^x$, a power of 2. Then after $O(\log n)$ iterations, we have $p = 1/n$.

- Previous lemma shows: At this point we can elect a leader w.h.p. in $O(\log n)$ rounds, i.e., within the corresponding execution of the second for-loop
- We have to try $\log n$ estimates until $k \approx \log n$, the total runtime until we reach $k = x$ becomes $O(\log^2 n)$ w.h.p.

□

Faster uniform leader election with collision detection?

Algorithm 20: ULE-CD

repeat
  Transmit with probability 1/2
  if *at least one node transmitted* **then**
    all nodes that did not transmit leave the protocol
until *one node transmits alone*

Theorem 33. *If n is unknown, Algorithm ULE-CD can be used to elect a leader in time $O(\log n)$ w.h.p.*

Proof. Exercise. □

We can be even faster. Consider **Algorithm Fast-ULE-CD**. In Phase 1, the algorithm computes a rough estimate of $\log n$. This is further refined using a binary search in Phase 2. Finally, the estimate for $\log n$ is made even more precise in the last phase using a biased random walk. Throughout, the algorithm is assumed to stop immediately as soon as a single node transmits alone (and, hence, a leader is found).

Let X denote the number of nodes that transmit in a single round. We first consider bounds on the probability that more than a single node or no node at all transmits, since based on these events we develop our estimate of $\log n$ in all three phases. We analyze these values in each of the three phases, where transmission is governed by i, j and k that estimate $\log n$.

Lemma 62. *If $j > \log n + \log\log n$, then* $\Pr[X > 1] \le \frac{1}{\log n}$.

Proof. Each node transmits with probability $1/2^j < 1/2^{\log n + \log\log n} = \frac{1}{n \log n}$. Hence, the expected number of nodes transmitting is $E[X] = \frac{n}{n\log n} = \frac{1}{\log n}$. Using Markov inequality:

$$\Pr[X > 1] \le \Pr[X > E[X] \cdot \log n] \le \frac{1}{\log n}.$$

□

Corollary 14. *If $i > 2\log n$, then* $\Pr[X > 1] \le \frac{1}{\log n}$.

Lemma 63. *If $j < \log n - \log\log n$, then* $\Pr[X = 0] \le \frac{1}{n}$.

Algorithm 21: Fast-ULE-CD

`Exponential Growth - Phase 1`

$i \leftarrow 1$
repeat
    $i \leftarrow 2 \cdot i$
    Transmit with probability $1/2^i$
until *no node transmitted*

`Binary Search - Phase 2`

$l \leftarrow i/2$
$u \leftarrow i$
while $l + 1 < u$ **do**
    $j \leftarrow \lceil (l+u)/2 \rceil$
    Transmit with probability $1/2^j$
    if no node transmitted **then** $u \leftarrow j$ **else** $l \leftarrow j$

`Biased Random Walk - Phase 3`

$k \leftarrow u$
repeat
    Transmit with probability $1/2^k$
    if no node transmitted **then** $k \leftarrow k-1$ **else** $k \leftarrow k+1$
until *exactly one node transmitted*

Proof. Each node transmits with probability $1/2^j > 1/2^{\log n - \log\log n} = \frac{\log n}{n}$. Hence, the probability that a node is at most $1 - \frac{\log n}{n}$. The probability for a silent slot is

$$\Pr[X = 0] \leq \left(1 - \frac{\log n}{n}\right)^n \leq \mathsf{e}^{-\log n} = \frac{1}{n}.$$

□

Corollary 15. *If* $i < \frac{1}{2}\log n$, *then* $\Pr[X = 0] \leq \frac{1}{n}$.

Lemma 64. *Let* y *be such that* $2^{y-1} \leq n \leq 2^y$, *i.e.,* $y \approx \log_2 n$. *If* $k > y + 2$ *then* $\Pr[X > 1] \leq 1/4$.

Proof. Markov inequality:

$$\Pr[X > 1] = \Pr\left[X > \frac{2^k}{n}\mathbb{E}[X]\right] < \Pr\left[X > \frac{2^k}{2^y}\mathbb{E}[X]\right] < \Pr[X > 4\mathbb{E}[X]] \leq \frac{1}{4}.$$

□

Lemma 65. *If* $k < y - 2$ *then* $\Pr[X = 0] \leq 1/4$.

Proof. A similar analyis is possible to upper bound the probability that a transmission fails if the estimate is too small. Since $k < y - 2$, we obtain

$$\Pr[X = 0] = \left(1 - \frac{1}{2^k}\right)^n < \mathsf{e}^{-n/2^k} < \mathsf{e}^{-2^{y-1}/2^k} < \mathsf{e}^{-2} < \frac{1}{4}.$$

□

Lemma 66. *If $y - 2 \le k \le y + 2$, then $\Pr[X = 1]$ is constant.*

Proof. Transmission probability is $p = \frac{1}{2^{y \pm \Theta(1)}} = \Theta(1/n)$. The lemma follows with an adaptation of Lemma 59. □

Lemma 67. *Let $|u - \log n| \le \log\log n$. Then with probability $1 - 1/\log n$ we find a leader in Phase 3 in $O(\log\log n)$ rounds.*

Proof. Sketch of the argument:

- Suppose u is already close to $\log n$. Then for any k, Lemmas 64 and 65 show that the random walk has a good bias towards an interval where a single transmission per round is likely.
- One can show: In $O(\log\log n)$ rounds we get $\Omega(\log\log n)$ rounds with $k \in [y-2, y+2]$ the interval around n.
- Lemma 66 then shows that the expected number of rounds with exactly one transmission is $O(\log\log n)$.
- Chernoff bounds imply that with probability $1 - 1/\log n$ there is at least one such round, i.e., we elect a leader.

□

Theorem 34. *If n is unknown, Algorithm Fast-ULE-CD can be used to elect a leader with probability at least $1 - O(\frac{\log\log n}{\log n})$ in time $O(\log\log n)$.*

Proof.

- Phase 1 should end with estimate of $\frac{1}{2}\log n \le i \le 2\log n$ in $O(\log\log n)$ rounds.
- Probability that Phase 1 terminates too early or too late (i.e., an error occurs in a round) is at most $1/\log n$ (Corollaries 14 and 15)
- Phase 2 should end with estimate of $\log n - \log\log n \le j \le \log n + \log\log n$ that is exact up to $\log\log n$-terms. Phase 2 is binary search on an interval of length $O(\log n)$, so this takes $O(\log\log n)$ rounds.
- Probability that Phase 2 terminates too early or too late (i.e., an error occurs in a round) is at most $1/\log n$ (Lemmas 14 and 15)
- By union bound: Probability that no error occurs in $O(\log\log n)$ rounds of Phases 1 and 2 is at most $O(\log\log n/\log n)$.
- If no error occurs, estimate u for $\log n$ is at most $\log\log n$ away. Probability that this happens at least $1 - O(\frac{\log\log n}{\log n})$.
- Apply previous lemma to show that Phase 3 terminates with a leader. Overall algorithm needs $O(\log\log n)$ rounds with probability at least $1 - O(\frac{\log\log n}{\log n})$.

□

With a more involved analysis, one can slightly improve the bound to the following near-optimal trade-off:

Corollary 16. *If n is unknown, Algorithm Fast-ULE-CD can be used to elect a leader with probability at least $1 - \frac{1}{\log n}$ in time $\log\log n + o(\log\log n)$.*

10.2 Modeling Interference

Crucial aspect for MAC protocols: **Modeling interference and collision**

A first example: **Disk Graph Model**.
Simple model, used extensively for algorithm design in the 80/90s:

- There are n base stations. Each base station has associated mobile devices, wants to transmit a message to them
- Mobile devices located in an area around the base station
- Suppose mobile device of station i is close to several base stations $\{i, j, k, ...\}$. If stations i and j transmit simultaneously, signals interfere and mobile device cannot decode any message

Formally, we model this using a conflict graph based on disks:

- Nodes V are base stations. Node v_i is a point in the Euclidean plane
- Mobile devices are represented by a disk with radius $r_i \geq 1$ around v_i
- Disks around v_i and v_j intersect $\Rightarrow$ Base stations are in conflict since mobile devices are close to several stations $\Rightarrow$ add an edge $\{v_i, v_j\}$ to E
- Resulting conflict graph $G = (V, E)$ is a **disk graph** – captures all pairwise conflicts among base stations
- Special case: **Unit-Disk Graph**, all radii $r_i = 1$.
- A subset $I \subseteq V$ is **conflict-free** if there is no edge among vertices in I. A conflict-free set of nodes is an **independent set** in G.

[Pic: Disk Graph]

Problem: Disk graphs can only express symmetric conflicts. In many cases conflicts are asymmetric, e.g., with directed communication and different sender/receiver locations.

More elaborate: **Protocol Model**

- There are n links. Link ℓ_i has sender s_i and receiver r_i. All senders and receivers are points in the Euclidean plane
- s_i wants to transmit message only to r_i, but also reaches other receivers r_j
- Suppose for r_j, sender s_i is roughly at the same distance or closer than sender s_j:

$$\mathsf{dist}(s_i, r_j) \leq (1 + \delta) \cdot \mathsf{dist}(s_j, r_j),$$

  where dist is distance in the plane and $\delta > 0$ is a constant. Then link ℓ_j is in conflict with link ℓ_i (but possibly not vice versa).
- Intuitively: Long links likely to be in conflict, short links are robust
- **Directed conflict graph** $G = (V, E)$: For each link ℓ_i add a node v_i to V. Directed edge $(v_i, v_j) \in E$ iff ℓ_j is in conflict with ℓ_i.
- A subset of links $I \subseteq V$ is **conflict-free** if no vertex in I has an incoming edge from any other vertex in I. A conflict-free set of nodes is an **independent set** in G.

[Pic: Senders, Receivers, Conflicts, Conflict Graph]

Problem: Conflicts are binary ($\exists$ incoming edge or not). In reality, a conflict arises only if the amount of communication and interfering signals on a channel becomes too large.

Even more elaborate: **SINR Model**

- There are n links. Link ℓ_i has sender s_i and receiver r_i. All senders and receivers are points in the Euclidean plane
- s_i wants to transmit message only to r_i, but also reaches other receivers r_j
- Sender s_i with **transmission power** p_i. Signal strength decays exponentially over distance. Received power at distance d from sender is p_i/d^α, where $\alpha > 1$ is a constant.
- Consider receiver r_i. Incoming signal from s_i has strength

$$\frac{p_i}{(\mathsf{dist}(s_i, r_i))^\alpha}$$

  Incoming interfering signals from the other senders $\{s_j \mid j \neq i\}$ have total strength

$$\sum_{j \neq i} \frac{p_j}{(\mathsf{dist}(s_j, r_i))^\alpha}$$

  Inherent noise in the channel is a constant $\gamma > 0$.
- The **signal-to-interference-plus-noise-ratio (SINR)** is

$$SINR_i = \frac{p_i \big/ (\mathsf{dist}(s_i,r_i))^\alpha}{\gamma + \sum_{j \neq i} p_j \big/ (\mathsf{dist}(s_j,r_i))^\alpha}$$

- Link ℓ_i is in conflict if $SINR_i$ is too small, i.e., $SINR_i \leq \beta$, for some constant β (common assumption in many systems: $\beta \geq 1$).
- Again long links quickly in conflict, short links are robust
- Depends also on power: Links with silent senders quickly in conflict, links with loud senders are more robust

[Pic: Links, Power, SINR, Conflict Graph]

Suppose transmission powers p_i are given and fixed. Then the SINR model yields a **directed conflict graph** with **edge weights**

- Suppose all parameters are > 0.
- For every directed pair of nodes (v_j, v_i), define a suitable edge weight

$$w(v_j, v_i) = \min\left\{1, \frac{\beta \cdot (\mathsf{dist}(s_i, r_i))^\alpha \cdot p_j}{p_i \cdot (\mathsf{dist}(s_j, r_i))^\alpha \cdot \left(1 - \frac{\gamma \cdot \beta \cdot (\mathsf{dist}(s_i,r_i))^\alpha}{p_i}\right)}\right\}$$

  that captures the "damage" that sender s_j causes at receiver r_i.
- Link ℓ_i is in conflict $\Leftrightarrow$ Incoming damage is too large, i.e., indegree greater than 1:

$$SINR_i \leq \beta \qquad \Longleftrightarrow \qquad \sum_{j \neq i} w(v_j, v_i) \geq 1$$

since

$$\begin{aligned}
& & \beta &\geq SINR_i \\
&\Leftrightarrow & \beta \cdot \left(\sum_{j \neq i} \frac{p_j}{(\mathsf{dist}(s_j, r_i))^\alpha} \right) &\geq \tfrac{p_i}{(\mathsf{dist}(s_i,r_i))^\alpha} \\
&\Leftrightarrow & \beta \cdot \tfrac{(\mathsf{dist}(s_i,r_i))^\alpha}{p_i} \left(\gamma + \sum_{j \neq i} \frac{p_j}{(\mathsf{dist}(s_j, r_i))^\alpha} \right) &\geq 1 \\
&\Leftrightarrow & \tfrac{\beta(\mathsf{dist}(s_i,r_i))^\alpha}{p_i} \cdot \gamma + \sum_{j \neq i} \frac{\beta(\mathsf{dist}(s_i, r_i))^\alpha p_j}{p_i(\mathsf{dist}(s_j, r_i))^\alpha} &\geq 1 \\
&\Leftrightarrow & \sum_{j \neq i} \frac{\beta(\mathsf{dist}(s_i, r_i))^\alpha p_j}{p_i(\mathsf{dist}(s_j, r_i))^\alpha} &\geq 1 - \tfrac{\gamma\beta(\mathsf{dist}(s_i,r_i))^\alpha}{p_i} \\
&\Leftrightarrow & \sum_{j \neq i} \frac{\beta(\mathsf{dist}(s_i, r_i))^\alpha p_j}{p_i(\mathsf{dist}(s_j, r_i))^\alpha \left(1 - \tfrac{\gamma\beta(\mathsf{dist}(s_i,r_i))^\alpha}{p_i}\right)} &\geq 1 \\
&\Leftrightarrow & \sum_{j \neq i} \min \left\{ 1, \frac{\beta(\mathsf{dist}(s_i, r_i))^\alpha p_j}{p_i(\mathsf{dist}(s_j, r_i))^\alpha \left(1 - \tfrac{\gamma\beta(\mathsf{dist}(s_i,r_i))^\alpha}{p_i}\right)} \right\} &\geq 1 \\
&\Leftrightarrow & \sum_{j \neq i} w(v_j, v_i) &\geq 1
\end{aligned}$$

- We used in the derivation above that

  $$1 - \frac{\gamma\beta(\mathsf{dist}(s_i, r_i))^\alpha}{p_i} > 0,$$

  which is equivalent to the natural condition

  $$\frac{p_i / (\mathsf{dist}(s_i,r_i))^\alpha}{\gamma} > \beta,$$

  i.e., the power of every link is sufficient to make the link successful and make the signal come through the noise (when there is no interference from others).
- A subset of links $I \subseteq V$ is **conflict-free** if for every vertex in I the indegree from other vertices in I is at most 1:

  $$\sum_{j \in I, j \neq i} w(v_j, v_i) < 1 \qquad \text{for all } v_i \in I.$$

  We call I an **independent set**.

[Schema: SINR model → weighted conflict graph]

For the subsequent algorithms, we will **forget about all details of the Disk-Graph, Protocol, or SINR model**. We simply assume we are **given a directed, weighted conflict graph** with edge weights in $[0, 1]$. Unweighted conflict graphs that result from Disk-Graph or Protocol model are expressed using binary edge weights in $\{0, 1\}$.

10.3 Coloring

Suppose every node has a single message and wants to make one successful (i.e., conflict-free) transmission.

Goal: Given a conflict graph, **color** the vertices with a set of colors. Every color class $I \subseteq V$ must be **conflict-free**, i.e., an **independent set**. Minimize the number of colors.

Color classes might be time slots. Therefore, coloring problem is sometimes termed **latency minimization** (minimize time until every node has transmitted once), or **link scheduling**.

Consider **Algorithm LogColor**, an ALOHA-style procedure in which all nodes randomly attempt to make successful transmissions. The probability for transmission attemps decays exponentially over the time phases.

Algorithm 22: LogColor

for $k = 1, 2, 3, \ldots$ **do**
  for $i = 1$ *to* $(4 \cdot 2^k) \cdot c \cdot \ln n$ **do**
    Transmit with probability $p = 1/2^k$
    if v transmitted and was conflict-free **then** v leaves the protocol

Running time of LogColor governed by the following **density measure** of the instance. Consider a weighted, directed conflict graph $G = (V, E, w)$ with edge-weights $w(e) \in [0, 1]$. The **max-average indegree** $MaxAvg(G)$ is given as follows. Consider for every induced subgraph the average indegree of the nodes, and take the maximum average indegree of all induced subgraphs:

$$MaxAvg(G) = \max_{G'=(V',E')\subseteq G} \sum_{v \in V'} \frac{\deg^-_{G'}(v)}{|V'|} \quad .$$

[Example: Graph, MaxAvg indegree]

Theorem 35. *Algorithm LogColor terminates after $O((1 + MaxAvg(G)) \cdot \log n)$ rounds w.h.p.*

Proof. Let the **critical value** k be such that $2^{k-1} \leq 4 \cdot (1 + MaxAvg(G)) \leq 2^k$.

- **Phase**: Consists of all rounds in the second for-loop with the same value of k
- **Critical phase**: The phase in which k has critical value
- Phases keep doubling in length $\Rightarrow$ takes only $O((1 + MaxAvg(G)) \cdot \log n)$ rounds until critical phase starts
- Length of critical phase is $O((1 + MaxAvg(G)) \cdot \log n)$ rounds

We consider the critical phase and number the rounds of this phase $t = 1, 2, 3, \ldots$

- Let X_v^t be the random variable s.t. $X_v^t = 1$ if v transmits in round t and 0 otherwise. Note $\mathbb{E}[X_v^t] = p = 1/2^k$.
- Consider subgraph $G_t = (V_t, E_t)$ induced by all nodes that are still participating in the protocol in the beginning of round t (i.e., V_t are all nodes still in need of a successful transmission)

- Let $n_t = |V_t|$ the number of remaining nodes in round t
- For the overall expected indegree:

$$\begin{aligned}
\mathbb{E}\left[\sum_{v_i \in V_t} \deg^-_{G_t}(v_i)\right] &= \mathbb{E}\left[\sum_{v_i \in V_t} \sum_{v_j \neq v_i, v_j \in V_t} w(v_j, v_i) \cdot X^t_{v_j}\right] \\
&= \sum_{v_i \in V_t} \sum_{v_j \neq v_i, v_j \in V_t} \frac{w(v_j, v_i)}{2^k} \\
&\leq \sum_{v_i \in V_t} \sum_{v_j \neq v_i, v_j \in V_t} \frac{w(v_j, v_i)}{4(1 + MaxAvg(G))} \\
&= \frac{n_t}{4} \cdot \frac{1}{1 + MaxAvg(G)} \cdot \sum_{v_i \in V_t} \frac{\deg^-_{G_t}(v_i)}{n_t} \\
&\leq \frac{n_t}{4}
\end{aligned}$$

- Consider nodes v with expected indegree $\mathbb{E}[\deg^-_{G_t}(v)] \leq 1/2$. Denote their set by V^s_t.
- There must be at least $n_t/2$ nodes in V^s_t – if not the total expected indegree would be larger than $n_t/4$.
- $v \in V^s_t$ encounters lots of interference only with small probability, since, by Markov inequality,

$$\Pr[v \text{ is in conflict}] = \Pr[\deg^-_{G_t}(v) \geq 1] = \Pr[\deg^-_{G_t}(v) \geq 2 \cdot \mathbb{E}[\deg^-_{G_t}(v)]] \leq 1/2$$

- v being in conflict depends only on which subset of other nodes decide to transmit. This is independent of v's own tranmission decision:

$$\begin{aligned}
\Pr[v \text{ successful in round } t] &= \Pr[v \text{ transmits and } v \text{ not in conflict}] \\
&= \Pr[X^t_v = 1] \cdot \Pr[v \text{ not in conflict}] \geq \frac{1}{2^k} \cdot \frac{1}{2}
\end{aligned}$$

Now consider how many nodes leave the protocol over time.

- In round t every node in V^s_t successful with probability at least $1/(2 \cdot 2^k)$.
- Expected number of successful nodes in round t is at least $|V^s_t|/(2 \cdot 2^k) \geq n_t/(4 \cdot 2^k)$. Expected number of nodes that remain for round $t+1$ is, thus,

$$\begin{aligned}
\mathbb{E}[n_{t+1}] &\leq \sum_{h=0}^{\infty} \left(1 - \frac{1}{4 \cdot 2^k}\right) h \cdot \Pr[n_t = h] \\
&= \left(1 - \frac{1}{4 \cdot 2^k}\right) \sum_{h=0}^{\infty} h \cdot \Pr[n_t = h] \\
&= \left(1 - \frac{1}{4 \cdot 2^k}\right) \mathbb{E}[n_t]
\end{aligned}$$

- Recursive application gives $\mathbb{E}[n_{t+1}] \leq n \cdot \left(1 - \frac{1}{4 \cdot 2^k}\right)^t$. Using $t = (4 \cdot 2^k) \cdot c \cdot \ln n$, we see

$$\mathbb{E}[n_{t+1}] \leq n \cdot \mathsf{e}^{-c \ln n} = 1/n^{c-1}$$

- n_{t+1} is a non-negative integer, so $\Pr[n_{t+1} > 0] \leq \mathbb{E}[n_{t+1}] = 1/n^{c-1}$. At the end of the critical phase, $\Pr[n_{t+1} = 0]$, i.e., the probability that all nodes have successfully transmitted, is at least $1 - 1/n^{c-1}$.

$\square$

How does $MaxAvg(G)$ relate to **chromatic number** $\chi(G)$, i.e., the **optimum number of colors** in the best coloring?

First consider disk graphs to outline the argument.

Theorem 36. *For any disk graph G the chromatic number $\chi(G) = \Omega(MaxAvg(G))$. Algorithm LogColor computes an $O(\log n)$-approximation.*

Proof. Consider a disk graph G and an optimal coloring of G.

- Treat G as weighted graph with symmetric weights $w(v_i, v_j) = w(v_j, v_i)$, where $w(v_i, v_j) = w(v_i, v_j) = 1$ if $\{v_i, v_j\} \in E$ and 0 otherwise.
- Consider node v_i and all neighbors v_j with **larger disk radius** $r_j \geq r_i$
- Geometry: Every independent set I can contain at most 5 neighbors of v_i with larger disk radius. Otherwise, at least two neighbor disks would intersect and could not be both in I.

[Pic: Disk, Neighborhood, at most 5 conflict-free neighbors with larger radius]

We denote the maximum indegree from independent larger-disk neighbors by $\rho(G) \leq 5$. We prove a lower bound on the optimal number of colors $\chi(G)$:

- Consider subgraph $G'(V', E')$ with the max-average indegree.
- Consider the total indegree from nodes with higher disk radius. Each color class contributes at most $\rho(G)$ to this indegree. Hence, for every node $v_i \in V'$

$$\sum_{v_j \in V', r_j \geq r_i} w(v_j, v_i) \leq \rho(G)\chi(G)$$

- On the other hand,

$$\begin{aligned}
MaxAvg(G) &= \sum_{v_i \in V'} \frac{\deg^-_{G'}(v_i)}{|V'|} \\
&= \frac{1}{|V'|} \sum_{v_i \in V'} \sum_{v_j \in V'} w(v_j, v_i) \\
&\leq \frac{1}{|V'|} \sum_{v_i \in V'} \sum_{v_j \in V', r_j \geq r_i} w(v_j, v_i) + w(v_i, v_j) \\
&\leq \frac{1}{|V'|} \sum_{v_i \in V'} 2 \cdot \rho(G) \cdot \chi(G) \\
&= 2\rho(G) \cdot \chi(G)
\end{aligned}$$

Hence $\chi(G) = \Omega(MaxAvg(G)/\rho(G))$. The theorem follows since $\rho(G) \leq 5$. $\square$

More generally, consider the **inductive independence number** $\rho(G)$ determined as follows.

- First define symmetric weights:

$$\bar{w}(v_i, v_j) = \bar{w}(v_j, v_i) = \frac{1}{2}(w(v_j, v_i) + w(v_i, v_j))$$

  Note for disk graphs $\bar{w}(v_i, v_j) = w(v_i, v_j)$.
- Consider an ordering π of the nodes (e.g., decreasing disk radius). For each ordering π and node v_i, let $\Gamma_\pi(v_i)$ be the set of nodes that come before v_i in the ordering (e.g., all nodes with higher radius).
- Consider any independent set $I \subseteq V$ and any node v_i. We compute the indegree w.r.t. symmetric weights from earlier nodes of I (e.g., indegree from independent disks with higher radius):

$$\sum_{v_j \in \Gamma_\pi(v_i) \cap I} \bar{w}(v_j, v_i)$$

- The ordering number $\rho_\pi(G)$ of G is the maximum indegree of any node from earlier nodes in any independent set I:

$$\rho_\pi(G) = \max_{v_i \in V} \max_{I \text{ independent set}} \sum_{v_j \in \Gamma_\pi(v_i) \cap I} \bar{w}(v_j, v_i)$$

- For disk graphs G and decreasing-disk-radius ordering π we saw above $\rho_\pi(G) \leq 5$.
- In general, the **inductive independence number** $\rho(G)$ is the best ordering number:

$$\rho(G) = \min_{\pi \text{ ordering of nodes}} \rho_\pi(G)$$

Corollary 17. *For every weighted, directed conflict graphs G, the chromatic number $\chi(G) = \Omega(MaxAvg(G)/\rho(G))$. Algorithm LogColor computes an $O(\rho(G) \cdot \log n)$-approximation.*

Proof. The proof follows by using the inductive independence number $\rho(G)$ in Theorem 36.

- Consider subgraph $G'(V', E')$ with max-average indegree.
- Consider total indegree w.r.t. symmetric weights $\bar{w}$ from nodes that are earlier in the optimal ordering π. Each color class contributes at most $\rho(G)$ to this indegree. Hence, for every node $v_i \in V'$

$$\sum_{v_j \in V', \pi(v_j) \leq \pi(v_i)} \bar{w}(v_j, v_i) \leq \rho(G)\chi(G)$$

- On the other hand,

$$\begin{aligned} MaxAvg(G) &= \sum_{v_i \in V'} \frac{\deg^-_{G'}(v_i)}{|V'|} \\ &= \frac{1}{|V'|} \sum_{v_i \in V'} \sum_{v_j \in V'} w(v_j, v_i) \\ &\leq \frac{1}{|V'|} \sum_{v_i \in V'} \sum_{v_j \in V', \pi(v_j) \leq \pi(v_i)} w(v_j, v_i) + w(v_i, v_j) \end{aligned}$$

$$
\begin{aligned}
&= 2 \cdot \frac{1}{|V'|} \sum_{v_i \in V'} \sum_{v_j \in V', \pi(v_j) \le \pi(v_i)} \bar{w}(v_j, v_i) \\
&\le 2 \cdot \frac{1}{|V'|} \sum_{v_i \in V'} \rho(G) \cdot \chi(G) \\
&= 2\rho(G) \cdot \chi(G)
\end{aligned}
$$

Hence $\chi(G) = \Omega(MaxAvg(G)/\rho(G))$. $\square$

For many interference models, the resulting conflict graphs have small inductive independence numbers. Small upper bounds can be shown even when for simple node orderings. Algorithm LogColor computes approximately-optimal colorings in all these models.

Model	Ordering	Upper Bound on $\rho(G)$
Disk Graphs	Disk Radius	5
Protocol Model	Link Length	$\left\lceil \frac{\pi}{\arcsin \frac{\delta}{2(\delta+1)}} \right\rceil - 1$
IEEE 802.11 model	Link Length	23
Distance-2-Matching	Disk Radius	$O(1)$
Distance-2-Coloring	Disk Radius	$O(1)$
SINR, Length-Monotone Powers	Link Length	$O(\log n)$
SINR, Sqrt-Power	Link Length	$O(1)$
SINR, Power Control.	Link Length	$O(1)$

10.3.1 Acknowledgements

Consider a link-based model, e.g., the Protocol or the SINR model. How does the sender s_i realize it was successful, i.e., the receiver successfully received the message? We consider acks and assume **bidirectional communication** – receiver r_i becomes sender, sender s_i becomes receiver. If r_i successfully received the message, it sends an ack to s_i.

Consider **Algorithm LogColorAck**. Every second round, every sender that is still in the protocol waits for ack from its receiver. Only if receiver successfully gets the message, it sends an ack to s_i in the next round with same probability. Sender keeps on transmitting every second round with probability p until receives the ack.

Transmitting acks is essentially sending a message in a **dual instance**:

- Senders become receivers and vice versa. The dual confict graph $\tilde{G}$ has the same vertices, all directed edges, and weights $\tilde{w}$ with $\tilde{w}(v_j, v_i)$ determined by roles of senders and receivers interchanged for each link ℓ_i.
- Let $MaxAvg(G, \tilde{G}) = \max\{MaxAvg(G), MaxAvg(\tilde{G})\}$ be the maximum of the max-average indegrees of both G and $\tilde{G}$.

Algorithm 23: LogColorAck

for $k = 1, 2, 3, \ldots$ **do**
  for $i = 1$ *to* $(4 \cdot 2^k)^2 \cdot c \cdot \ln n$ **do**
    In the first round:
      Sender transmits message with probability $p = 1/2^k$
      Receiver listens
    In the second round:
      If receiver successfully received message in first round,
        it transmits ack with probability $p = 1/2^k$
      Sender listens
    if ack successfully received **then** v leaves the protocol

Theorem 37. *Algorithm LogColorAck terminates after* $O((1 + MaxAvg(G, \tilde{G}))^2 \cdot \log n)$ *rounds w.h.p.*

Proof. Adjust proof of Theorem 35. Define the critical phase as the one with k such that $2^{k-1} \leq 4(1 + MaxAvg(G, \tilde{G})) \leq 2^k$. Consider iterations $t = 1, 2, 3, \ldots, 16 \cdot 2^{2k} \cdot c \cdot \ln n$ of the critical phase.

- Note $MaxAvg(G, \tilde{G}) \geq MaxAvg(G)$ and repeat the arguments above. Thus, the expected indegree for senders is $\mathbb{E}\left[\sum_{v_i \in V_t} \deg^-_{G_t}(v_i)\right] \leq \frac{n_t}{4}$ and the expected number of successful senders in round t is at least $\frac{1}{4 \cdot 2^k} \cdot n_t$.
- Hence, for the number of successfully received messages
$$\mathbb{E}[\text{number of receivers that get msg.} \mid n^t \text{ senders each send w. prob. } 1/2^k] \geq \frac{1}{4 \cdot 2^k} \cdot n_t.$$
- Since $MaxAvg(G, \tilde{G}) \geq MaxAvg(\tilde{G})$, the same analysis applies to receivers and the dual conflict graph. Hence, for any number of h receivers that transmit with probability $p = 1/2^k$ each, expected number of successfully received acks is
$$\mathbb{E}[\text{number of senders getting ack} \mid h \text{ receivers each send w. prob. } 1/2^k] \geq \frac{1}{4 \cdot 2^k} \cdot h.$$
- However, only $\mathbb{E}[h] = \frac{1}{4 \cdot 2^k} \cdot n_t$ receivers actually attempt to transmit an ack in the second round. Thus,
$$\mathbb{E}[\text{number of senders that get ack in iteration } t] \geq \left(\frac{1}{4 \cdot 2^k}\right)^2 \cdot n_t.$$
- Remaining analysis as above: Expected number of nodes that remain for iteration $t+1$
$$\mathbb{E}[n_{t+1}] \leq \left(1 - \frac{1}{16 \cdot 2^{2k}}\right) \mathbb{E}[n_t]$$
so $\mathbb{E}[n_{t+1}] \leq n \cdot \left(1 - \frac{1}{16 \cdot 2^{2k}}\right)^t$. Use $t = 16 \cdot 2^{2k} \cdot c \cdot \ln n$. Hence, in the critical phase with probability at least $1 - 1/n^{c-1}$ all senders successfully transmit and receive the ack.

□

But wait – how does a receiver r_i realize it was successful, i.e., the sender successfully received the ack? Ack-ack? :)

10.4 Maximum Independent Set

Consideration of acks is possible but tedious, so we assume in this section that **nodes do realize** whether their transmission attempt was **successful**.

Goal: Find a good independent set, i.e., maximize **number of successful transmissions**. We use **learning algorithms** that achieve a good number of successful transmissions **on average over time**. This task also termed **throughput** or **capacity maximization**.

Note: **No fairness guarantees** – some nodes might be successful all the time, other nodes never successful. We simply try to make a lot of successful communication attempts overall. Algorithms with **provable trade-offs of fairness and throughput**: Open problem!

10.4.1 Online Learning

We apply **online learning algorithms** to steer transmission attemps. The basic online learning scenario is a simple, round-based reward maximization process:

- T rounds, K actions in each round
- In round $t = 1, \ldots, T$: The decider chooses one action randomly
- After action is chosen, decider sees a reward in $[0, 1]$ for the chosen action
- Decider updates its probability distribution for choice of action, then next round starts.
- Goal: Decider tries to maximize its total reward of chosen actions

We model our domain here as a **special case**:

- Each node uses some algorithm for online learning to steer transmission over T rounds
- In each round t, node v_i has **2 actions** to choose from: Transmit/Not transmit
- Define $x_i^t = 1$ if node v_i decides to transmit in round t, $x_i^t = 0$ otherwise.
- We assume node gets a **utility** $u_i(x^t)$ in round t for chosen action:

$$u_i(x^t) = \begin{cases} 1 & x_i^t = 1 \text{ and } v_i \text{ successful} \\ -1 & x_i^t = 1 \text{ and } v_i \text{ not successful} \\ 0 & x_i^t = 0 \end{cases}$$

  Why exactly this utility? Because *that's how we do things around here*!
  Also, we can prove Lemma 68 below.
- Actual rewards shall be in $[0, 1]$, define reward $r_i(x^t) = (u_i(x^t) + 1)/2 \in \{1, 0, 0.5\}$
- Note: Reward $r_i(x^t)$ for v_i depends on complete vector x^t of **actions chosen by all nodes**, since it depends on other actions if i's transmission attempt is (un-)successful

Algorithm Exp3-WN is an application of the general-purpose online learning algorithm Exp3 to our special case. It is a carefully designed **exploration/exploitation** trade-off:

- Most of the time, exploit your experience from the past.
- Action weights w_0, w_1 adjusted to express past success experiences. Choose actions with probabilities proportional to weight.
- Every once in a while: Be crazy and explore (non-)transmission with probability 1/2.
- Exploration probability η chosen carefully based on total time interval T

Algorithm 24: Exp3-WN

Set $\eta \leftarrow \min\left\{1, \sqrt{\frac{2}{e-1} \cdot \frac{1}{T}}\right\}$ `// exploration probabililty`

$w_0 \leftarrow 1$, $w_1 \leftarrow 1$ `// weights ~ previous success experience`

for $t = 1, \dots, T$ **do**

- Draw random number $x \in [0, 1]$
- **if** $x < \eta$ **then** $p \leftarrow 1/2$ **else** $p \leftarrow \frac{w_1}{w_0+w_1}$ `// Exploration or Exploitation`
- Transmit with probability p
- **if** *transmitted and successful* **then**
  - $x \leftarrow 1 \cdot \frac{w_0+w_1}{w_1(2/\eta-1)+w_0}$
  - $w_1 \leftarrow w_1 \cdot e^x$ `// increases transmission prob.`
- **if** *not transmitted* **then**
  - $x \leftarrow \frac{1}{2} \cdot \frac{w_0+w_1}{w_0(2/\eta-1)+w_1}$
  - $w_0 \leftarrow w_0 \cdot e^x$ `// decreases transmission prob.`

Why even consider using Exp3? What kind of guarantee do we get from using it?

Consider a **history of actions** for all nodes $x = (x^1, \dots, x^T)$. We define the **regret** of node i in this history as

$$R_i(x) = \max_{y=0,1} \sum_{t=1}^{T} r_i(y, x_{-i}^t) - \sum_{t=1}^{T} r_i(x^t)$$

Here (y, x_{-i}^t) is the vector $(x_1^t, \dots, x_{i-1}^t, y, x_{i+1}^t, \dots, x^t)$, i.e., all actions as in x^t, only the one for v_i replaced by y. $R_i(x)$ captures the maximum gain in reward that node v_i would get when **always transmitting** or **always not transmitting**.

[Pic: Sequence of chose actions, Sequence never-transmit, Sequence always-transmit, Regret]

Exp3-WN is a **no-regret algorithm**. Suppose node v_i is using Exp3-WN to compute x_i^t. The other nodes can have arbitrary behavior. Then for v_i it is known that the **average regret over time goes to 0**

$$R_i(x)/T \to 0 \qquad \text{for } T \to \infty$$

Rest of the chapter: Simplify matters a bit. Asume all nodes use Exp3 and history x will be such that **every node has 0 average regret** $R_i(x)/T \leq 0$ (and, thus, $R_i(x) \leq 0$). This will be the key property for our analysis – all results hold similarly when nodes use **any other no-regret algorithm**.

We defined the rewards as above to show the following condition.

Lemma 68. *Suppose a history x is such that node v_i has regret $R_i(x) \leq 0$. Then at least half of v_i's transmission attempts have been successful.*

Proof. Note that

$$
\begin{aligned}
0 \geq R_i(x) &= \max_{y=0,1} \sum_{t=1}^{T} r_i(y, x_{-i}^t) - \sum_{t=1}^{T} r_i(x^t) \\
&= 2\left(\max_{y=0,1} \sum_{t=1}^{T} u_i(y, x_{-i}^t) - \sum_{t=1}^{T} u_i(x^t)\right) \\
&\geq 2\left(0 - \sum_{t=1}^{T} u_i(x^t)\right)
\end{aligned}
$$

Thus, $\sum_{t=1}^{T} u_i(x^t) \geq 0$. Hence, for every unsuccessfuly attempt with $u_i(x^t) = -1$, there is at least one other successful attempt. □

10.4.2 Learning in Bounded-Independence Graphs

In general, the no-regret property by itself does not guarantee good throughput!

Consider an unweighted conflict graph that is a star with $n-1$ leaves. Construct a no-regret history x as follows:

- Suppose the star center transmits the whole time, i.e., $x_1^t = 1$ for all $t = 1, \dots, T$
- Suppose leaf nodes never transmit, i.e., $x_i^t = 0$, for all $i = 2, \dots, n$ and $t = 1, \dots, T$
- Every node has 0 regret!
- Average number of successful transmissions is 1. Optimal would be $n-1$. By always transmitting, the star center "kills" a large independent set.

[Pic: Star, bad independent set, explain no-regret property]

We restrict attention to graphs, in which "no vertex can kill a large independent set". For unweighted graphs G consider the **independence number** $\alpha(G)$:

- Consider the maximum indegree caused by any node at any independent set:

$$
\alpha(G) = \max_{v_i \in V} \max_{I \text{ independent set}} \sum_{j \in I} w(v_i, v_j)
$$

- Note: $\alpha(G) = k$ means that there is no induced subgraph that is a star with $k+1$ or more outgoing edges from the center.
- Note: Consider any independent set I. If any node $v \notin I$ decides to join I, it causes a conflict for at most $\alpha(G)$ other nodes of I.

Examples: There are disk graphs with $\alpha(G) = n - 1$. For *unit*-disk graphs, $\alpha(G) \leq 5$. (Why?)

For edge-weighted conflict graphs, we generalize this definition as follows. A conflict graph G is c**-independent** if for every independent set $I \subseteq V$ there is a subset $I' \subseteq I$ s.t.

- for every node $v_i \in V$ the total indegree of I' received from v_i satisfies

$$
\sum_{v_j \in I'} w(v_i, v_j) \leq c.
$$

- I' is not too small: $|I'| \geq |I|/2$.

If an unweighted graph G is c-independent, it has independence number $\alpha(G) = O(c \cdot \log n)$. (Exercise)

Theorem 38. *Consider a c-independent conflict graph. Suppose there is a history x such that all nodes v_i have $R_i(x) \leq 0$. Then the average number of successful transmissions is an $O(c)$-approximation of the optimum.*

Proof. In the optimum we simply assign a maximum independent set I^* to transmit in every round $t = 1, \ldots, T$, for a maximum number of $|I^*|T$ successful transmissions.

How many successful transmissions do the nodes make in x? Lemma 68: At least half of all transmission attempts are successful, so consider **total number of attemps**.

- Consider $v_i \in I^*$. Let $t_i = \sum_{t=1}^{T} x_i^t$ be number of i's attempts. If $t_i \geq T/2$, then great!
- Suppose at least half the nodes in $|I^*|$ have $t_i \geq T/2$. Then at least $|I^*|T/4$ attempts in total. With Lemma 68 this implies an 8-approximation.

What if only few nodes of I^* do attempt transmission frequently?

- Suppose at most half the nodes in $|I^*|$ have $t_i \geq T/2$. Consider the infrequent nodes, i.e, $I_0^* \subseteq I^*$ are all nodes from I^* with $t_i < T/2$. Note: $|I_0^*| \geq |I^*|/2$
- For $v_i \in I_0^*$, let $\mathcal{T}_0 = \{t \mid x_i^t = 0\}$ be the rounds where v_i decided to stay silent.
- No regret: $\sum_{t=1}^{T} u_i(1, x_{-i}^t) - \sum_{t=1}^{T} u_i(x) \leq 0$. This implies

$$\sum_{t \in \mathcal{T}_0} u_i(1, x_{-i}^t) \leq 0.$$

- Similar argument as in Lemma 68 shows that v_i would have been unsuccessful in at least half of the rounds of $\mathcal{T}_0$
- Consider total indegree of v_i from transmitting nodes in all T rounds. This at least $|\mathcal{T}_0|/2 = (T - t_i)/2 > T/4$ since $t_i < T/2$.
- Graph is c-independent: Every attempt causes indegree at most c on $I' \subseteq I_0^*$ with $|I'| \geq |I_0^*|/2 \geq |I^*|/4$.
- Total indegree of all nodes in I' in all rounds at least $|I'| \cdot T/4$. This implies that there were at least $|I'|T/(4c)$ attempts of all nodes in total.
- Total number of attempts is $|I'| \cdot T/(4c) \geq |I^*| \cdot T/(16c)$ With Lemma 68 this implies a $32c$-approximation.

□

10.4.3 Jamming-Resistant Learning

In many applications, a system is not the only one using a channel or frequency band. Multiple subsystems in the same channel give rise to **jamming**. Can we use learning algorithms to maximize the throughput even in channels with (adversarial) jamming?

We consider is a standard approach for modeling jamming conditions.

- The system runs for T rounds.

- A $(T', 1-\delta)$**-jammer** can decide to make a node unsuccessful. It can decide this individually for each node.
- For $v \in V$ and every **subinterval of** T' **rounds**, the jammer can render at most a $(1-\delta)$**-fraction** of rounds unsuccessful for v. It every round t, the jammer can make a jamming decision for each node even **after** it knows the transmission decisions of all nodes in round t.
- Node v does not learn if unsuccessful transmission is due to (successful) attempts of other nodes or jamming.

[Pic: Jamming, Time Interval, Fraction of Rounds]

We again consider a c-independent conflict graph and nodes using no-regret learning.
- A **phase** is an interval of k rounds. Learning will be applied to phases.
- During each phase, node v executes the same action in all rounds, i.e., transmit in all k rounds or not
- Phases of other nodes do not need to be synchronized
- Phase R is labeled **successful** if a **fraction** ν of the rounds in the phase were successful.

We define some desirable properties inspired by the proof of Theorem 38. Consider a history x of transmission decisions by all nodes.
- q_v is the fraction of phases where v attempted transmission, w_v is the fraction of successful phases
- x is γ**-successful** if for every node v

$$q_v \leq \frac{2w_v}{\gamma}.$$

  The number of attempted transmissions is roughly the number of successful ones.
- x is η**-blocking** if for every node with $q_v \leq \frac{1}{4}\eta$ we have for the fraction of phases f_v that are unsuccessful due to other nodes

$$f_v \geq \frac{1}{4}\eta \qquad \text{and} \qquad \sum_{u \in V} w(u,v) q_u \geq \frac{1}{8}\eta.$$

  If a node did not attempt many transmission phases, this was because other nodes made a lot of phases unsuccessful, and the average indegree was large.

It would be great if the algorithms will compute a history x that is γ-successful and η-blocking for large parameters of γ and η, because these conditions can be used to show that there is a lot of throughput.

Theorem 39. *Suppose the algorithms implemented by all nodes compute a history x which is γ-successful and η-blocking. Against $(T', 1-\delta)$-jammers the average throughput of x guarantees an approximation factor of*

$$O\left(\frac{\max(1,c)}{\nu \cdot \gamma \cdot \eta}\right)$$

Proof. The proof uses duality of linear optimization problems. Consider the maximum independent set I^* and the set $I' \subseteq I^*$ as in the definition of c-independence. It represents a feasible solution of the following **linear program (LP)** by setting $x_v = 1$ iff $v \in I'$.

$$\begin{array}{rrcll}
\text{Maximize} & \displaystyle\sum_{v \in V} x_v & & & \\
\text{subject to} & \displaystyle\sum_{v \in V} w(u,v) x_v & \leq & c & \forall u \in V \\
 & x_v & \in & [0,1] & \forall v \in V.
\end{array}$$

Now consider the system when we have $(T', 1-\delta)$-jammers.

- Due to individual jamming of nodes, each round t has a possibly different maximum independent set I_t^*.
- Consider the subset $I'_t \subseteq I_t^*$ as in the definition of c-independence
- Let x_v be the fraction of times when v is in I'_t

  $$x_v = \frac{|\{t \mid v \in I'_t\}|}{T}$$

  As each I'_t satisfies the indegree constraint from c-independence, the solution is again feasible for the LP.
- The objective function value is at least half (since using I'_t instead of I_t^*) of the optimal average number of successful transmissions that would have been possible under the jamming pattern chosen by the jammers.

Strong duality of linear programs – the essentials:

- For every LP there is a **dual LP**.
- The optimum objective function value of an LP **equals the optimum objective value of its dual**.
- Every feasible solution of an LP has **less objective value** than every feasible solution of the dual.

The dual for our LP is the following Dual-LP

$$\begin{array}{rrcll}
\text{Minimize} & \displaystyle\sum_{v \in V} c \cdot y_v + \sum_{v \in V} z_v & & & \\
\text{subject to} & \displaystyle\sum_{u \in V} w(u,v) y_u + z_v & \geq & 1 & \forall v \in V \\
 & y_v, z_v & \geq & 0 & \forall v \in V
\end{array}$$

Construct a feasible dual solution from the history x computed by the algorithms, which is γ-successful and η-blocking.

- Set $y_v = \frac{1}{\eta} \cdot 8q_v$ and $z_v = \frac{1}{\eta} \cdot 4q_v$. Solution fulfills all the constraints:
- If $q_v \geq \frac{1}{4}\eta$, constraint is fulfilled since $z_v \geq 1$.

- If not, η-blocking yields $\sum_{u \in V} w(u,v) q_u \geq \frac{1}{8}\eta$. Plugging in shows constraint fulfilled.

How different are the objective function values? Since we constructed feasible solutions for both LP and Dual-LP, strong duality implies

$$\sum_{v \in V} \frac{|\{t \mid v \in I'_t\}|}{T} \leq \sum_{v \in V} c \cdot \frac{8}{\eta} \cdot q_v + \frac{4}{\eta} \cdot q_v \quad .$$

Since history x is γ-successful,

$$\sum_{v \in V} \frac{|\{t \mid v \in I'_t\}|}{T} \leq \sum_{v \in V} \max(1,c) \cdot \frac{24}{\eta \cdot \gamma} \cdot w_v \quad .$$

Remember that a phase is of length k.

- In a successful phase node v is successful in at least νk rounds.
- Hence, w_v and total number of successful rounds are related by a factor of ν.
- This yields a factor of $O(\max(1,c)/(\eta\gamma\nu))$ difference between the objective function values of our solution for Dual-LP (based on the history x) and LP (for at least half of the optimum).
- The solution computed in history x is only this factor worse than the optimum.

$\square$

We apply no-regret algorithms to phases in the following way:

- Phase length: $k = T'$.
- Success fraction for phases: $\nu = \frac{1}{2}\delta$
- Action chosen in the beginning of phase and fixed throughout the phase
- Utililty obtained for phase R

$$u_i^R(x) = \begin{cases} 1 & v_i \text{ transmitted in } R \text{ and } w_i^R \geq \frac{1}{2}\delta \\ -1 & v_i \text{ transmitted in } R \text{ and } w_i^R < \frac{1}{2}\delta \\ 0 & v_i \text{ did not transmit in } R \end{cases}$$

If the algorithms compute a history with no regret, such a history is 1-successful and 1-blocking.

Corollary 18. *If all nodes use no-regret algorithms with the above given parameters, they compute a $O(1/\delta)$-approximation in systems with $(T', 1-\delta)$-jammers.*

The framework in Theorem 39 is very flexible and allows many more aspects to be incorporated. One more example: What if T' is unknown to the nodes?

- Phase length: $k = 1$, i.e., no phases, single rounds
- Utility obtained for phase/round R

$$u_i^R(x) = \begin{cases} 1 & v_i \text{ transmitted in } R \text{ and successful} \\ -\frac{\delta}{2-\delta} & v_i \text{ transmitted in } R \text{ and unsuccessful} \\ 0 & v_i \text{ did not transmit in } R \end{cases}$$

If the algorithms compute a history with no regret, such a history is $\frac{\delta}{2}$-successful and δ-blocking.

Corollary 19. *If all nodes use no-regret algorithms with the above given parameters, they compute a $O(1/\delta^2)$-approximation in systems with $(T', 1-\delta)$-jammers and unknown T'.*

Chapter 11

Blockchain and Consenus

11.1 Cryptocurrencies, Trust, and Consensus

Bitcoin is a **cryptocurrency**, i.e., a decentralized currency based on cryptographic features.

- We discuss some basic design principles and distributed computing aspects.
- Many more aspects: Protocols, incentives, crypto aspects, etc.
- Only a high-level exposition and details on some aspects of distributed computing, especially consensus protocols

Key feature of a currency: **Trust**

- Ancient times: Trade via direct exchange, one could directly see which goods to give and which ones to receive, immediate and direct negotiation
- Development of **currencies**: Exchange goods for coins and bills,
- Advantage: Flexibility in time and location, sell stuff here today, buy other stuff tomorrow over there, easier to collect taxes!
- **Trust** that everyone else will exchange them for goods and services
- Origin of trust: **Country, government, economy** etc. Trust generated, e.g., by total value of money being backed by large amounts of gold or goods
- Trust breaks down: Currency becomes worthless, replaced by other means (e.g., cigarettes in Germany directly after WWII)

Modern currencies mostly abstract and digital

- Less people use coins and bills, just abstract numbers in our accounts
- Governments often do not keep large gold reserves
- Trust? Generated by a system of (more or less) trusted parties, like government, federal reserve, banks, credit card companies
- Recent example for significance of trust: Einlagensicherung

What about a **decentral currency without a trusted party** standing behind it?

- More anonymity, no central control instance
- Enables access to global economy and many more trade possibilities
- Cheaper transactions, no intermediators
- Possibly less secure and easier to use for illegal activities

- What happens if something breaks – who is responsible to fix things?
- No manipulation of exchange rate to help the economy
- **How to generate trust in such a system?**

Bitcoin system, proposed in 2008 by "Satoshi Nakamoto" (pseudonym, real author(s) unknown). Key idea: **The bank is everyone** – everyone keeps a record of all bank accounts

More concretely:
- There is a peer-to-peer system of all users that maintains the transactions of all bank accounts
- Transactions are blocks attached to the history ($\rightarrow$ blockchain).
- Suppose Alice wants to send some money to Bob
- She sends a message to the network: "I give 50$ to Bob", signed with her private key
- Network needs to update, takes some time

Is this a good design? Suppose Alice tries a **Double Spending Attack**:
- Send message again to different part of the network with slightly different content: "I give 50$ to Chales", signed with her key
- Which message is accepted? Can she spend her money twice?
- Solution with **centralized trusted party**: Checks if Alice has the money, issues transaction number (TAN), a key that can be used only once, generated based on content of message.
- Trusted party sends TAN to the network, update is executed.
- But we want a **distributed system**! Idea: Network plays the role of trusted party and issues the "TAN"
- Network decides based on majority vote which meassage is the truth and which transaction is accepted into the blockchain.
- Trust might be low for a single participant of the P2P network, but trust is high in the system of all nodes. If there is fraud in the system, the overall trust suffers, currency drops in value, everyone loses.

Update of blockchain via **consensus**:
- Each coin has a unique ID
- Alice sends message to Bob: "I give the coin with ID x to Bob", signs it.
- Bob gets message and checks based on his copy if Alice has the coin to do so
- Bob sends to everyone in the network that he accepts the transaction
- Open transactions are collected by everyone in a block
- If majority agrees with the contents of a block, it is appended to the blockchain and the transactions take effect
- If Alice sends concurrent message "I give the coin with ID x to Chales", content of block becomes ambigious, finally block gets discarded

Eventually, this solution needs **consensus** among a majority of network nodes on the content of a block and where the coin of ID x will end up.

11.2 Fault Tolerance and Byzantine Generals

Byzantine Fault Tolerance

- Distributed system composed of normal/regular/honest items, **faulty** items
- Faulty items produce any sort of unpredictable output
- There is work on faulty nodes, faulty edges, faulty memory cells, faulty ...
- Usually poses super hard challenges.

Byzantine Generals Problems:

- Basic scenario to study **consensus against faults**
- Several armies are located outside of a city. Each division commanded by a different general. Generals try to coordinate on an attack plan
- Some parts of the army (and possibly even some generals!) are corrupted by the enemy.
- Consensus problem: (How) can the loyal generals agree on consistent attack plans?

Faulty Communication: **Two Generals Problem**

- Two generals know that they both decided to attack, need to agree on the same time
- Communication is faulty: Messages might be lost.
- Can they reach consensus on an attack time and both know that they both agreed?

Theorem 40. *There is no consensus algorithm for the Two Generals Problem.*

Proof. Consider a communication protocol P that solves the problem, i.e., at some point the generals agree on a time to attack and know that they both agreed.

- Consider last message m sent in P, w.l.o.g. sent to general 1.
- W.l.o.g. m is needed to convince general 1 that both generals agree, i.e., before m arrives, general 1 is not sure that both agreed to an attack time.
- Since general 2 cannot know if m arrives, he must be sure that both agreed before sending m
- But general 1 was not sure of that before he received $m \rightarrow$ contradiction.

□

Faulty Nodes: **Byzantine Generals Problem**

- n generals try to coordinate on an action, some generals are traitors.
- Devise consensus algorithm for generals to agree on attack (A) or retreat (R)
- Initially, every general has personal opinion
- Possible messages $\{A, R\}$, arrive correctly, sender/receiver known & correct, absence of msgs can be detected, no crypto
- No change of content, origin, destination of msgs, no manipulation by absence of msgs
- **Goal**: All loyal generals should decide to take the same action

Reduction to Commander/Lieutenant Case

- Loyal generals broadcast their true opinion. Suppose a traitor also sends same (manipulated) opinion to every loyal general
- Then all loyal generals execute same protocol on same input and reach consensus in final decision.

- Hence, traitors must send different messages to different generals. However, it is *not necessary to identify traitors*, we just want consensus of final decisions among loyal generals
- Rephrase as **equivalent problem**: One general sends orders to all others. Design algorithm and use it once for every general in the role as commander to send its orders (= initial opinion) to everyone else

$\rightarrow$ One **commander** C, $n-1$ **lieutenants** L1, L2,...
Interactive consistency constraints:
(IC1) All loyal lieutenants must agree on an order.
(IC2) Loyal lieutenant must follow order of a loyal commander.

Lemma 69. *There is no consensus algorithm for the Three Generals Problem.*

Proof. Consider the case for one commander and two lieutenants L1, L2. The following two scenarios are equivalent for a loyal L1:

1. C is traitor, sends A to L1, R to L2. L2 sends R to L1.
2. C is loyal, sends A to both L1,L2. Traitor L2 sends R to L1.

Same situation for L1, but different actions are needed $\rightarrow$ solution impossible. $\square$

[Pic Three Generals Scenarios]

Lemma 70. *There is no consensus algorithm for m traitors and $m+1 < n \leq 3m$.*

Proof. Simulation argument. Suppose $n = 3m$ and assume a consensus algorithm for $3m$-Generals Problem exists (Albanian generals). Use it to solve the Three Generals Problem (Byzantine generals):

- Byzantine commander $\rightarrow$ Albanian commander, $m-1$ Albanian lieutenants.
- Byzantine lieutenant $\rightarrow$ m Albanian lieutenants.
- At most one Byzantine general is traitor $\rightarrow$ at most m Albanian generals are traitors
- Use consensus algorithm to solve the Albanians instance
- All loyal lieutenants reach same decision in the end
- Loyal Byzantine generals read off the decision from their Albanian lieutenants, implies IC1 and IC2, i.e., consensus for the Byzantine Three Generals Problem

$\rightarrow$ Contradiction.

More traitors only make the problem harder (until there is just 1 loyal agent $\rightarrow$ trivial) $\square$

[Pic $3m$ generals, m traitors]

For $n \geq 3m+1$ consider the **Algorithm Oral-Messages OM(m, S, v_S)**. It is a recursive procedure that uses the number m of traitors as input. It relies on a majority vote among the lieutenants and shows inductively that their consistent votes can steer the loyal generals to a consistent decision.

[Example: 4 Generals, 1 Traitor. C is traitor: all Li get same msgs. Li is traitor: all Li receive at least two msgs v]

Theorem 41. *The OM(m, S, v_S) Algorithm solves the Byzantine Generals Problem for m traitors and $n \geq 3m+1$ in time $O(n^m)$.*

Algorithm 25: OM(m, S, v_S) for subset of loyal generals

Input: Estimated num. traitors m, set S of lieutenants, possibly different commander message for each lieutenant $v_S = (v_i)_{i \in S}$

if $m = 0$ **then**
  Set $v'_i \leftarrow v_i$ (or $v'_i \leftarrow R$ if no msg v_i received), for every $i \in S$
else
  for *every* $i \in S$ **do**
    Loyal i chooses values $v^i_{S\setminus\{i\}} = (v^i_j)_{j \in S\setminus\{i\}}$ by $v^i_j = v_i$ for all $j \in S, j \neq i$
    Receives vector $w_{-i} = (w^i_j)_{j \in S\setminus\{i\}} \leftarrow OM(m-1, S \setminus \{i\}, v^i_{S\setminus\{i\}})$
    w_{-i} has entry for every $j \in S, j \neq i$. Set $w^i_j \leftarrow R$ if no entry is received.
    Set $w^i_i \leftarrow v_i$
    Set $v'_i \leftarrow$ majority value of $(w_j)_{j \in S}$ `// among all` w^i_j `and` $w^i_i = v_i$

return *vector of values* $(v'_i)_{i \in S}$

Proof. We first prove condition IC2 holds whenever OM(m, S, v_S) for any set S of $2k + m$ generals and at most k traitors. IC2 applies only when C is loyal. Assume loyal C sends v to all Li in S.

Induction. Start: $m = 0$ and loyal commander (i.e., all v_S are same). Then the vector of decisions returned by OM$(0, S, v_S)$ yields IC2. Hypothesis: We get IC2 for $m - 1$. Prove IC2 for m.

- Consider invocation of OM$(m-1, S \setminus \{i\}, v_{S\setminus\{i\}})$. Since $|S| > 2k+m$, we have $|S|-1 > 2k + m - 1 \geq 2k$.
- By hypothesis: Every loyal Li receives $w^i_j = v$ for each loyal Lj.
- At most k traitors, more than k loyal ones among the $|S| - 1$ Lj's
- Majority vote gives consistent action for loyal lieutenants $i \in S$.

For the proof that both IC1 and IC2 hold, we again apply an induction. No traitors: OM(0) gives IC1 and IC2. Hence, assume IC1 and IC2 hold for OM$(m-1, S \setminus \{i\}, v_{S\setminus\{i\}})$ and prove it for OM(m, v_S, S):

- Case 1: C is loyal. Let $k = m$ above, OM(m, S, v) satisfies IC2. IC1 follows.
- Case 2: C is traitor. At most m traitors, C is one, at most $m - 1$ traitor lieutenants
- At least $3m - 1$ lieutenants and $3m - 1 > 3(m - 1)$.
- Apply induction hypothesis: OM$(m - 1, S \setminus \{i\}, v_{S\setminus\{i\}})$ when called by Li satisfies IC1 and IC2.
- For each j, any two loyal Li and Li' receive same $w^i_j = w^{i'}_j$ from a loyal lieutenant Lj, due to IC1 and IC2 of the respective calls of OM$(m - 1, S \setminus \{i\}, v_{S\setminus\{i\}})$ and OM$(m - 1, S \setminus \{i'\}, v_{S\setminus\{i'\}})$
- Hence, loyal lieutenants get sufficiently many consistent values from other loyal lieutenants to arrive at consistent decision in the majority step.
- This proves IC1 and the theorem.

□

11.3 Proof-of-Work Consensus in Bitcoin

In practice, consensus conditions are even harder to achieve. Communication might be asynchronous, and/or messages might be lost. One can show that even in **shared-memory systems with asynchrony**, consensus can be **impossible to achieve** even when there is **at most 1 traitor**.

Thus, in principle, obtaining consensus is hopeless. Then again, there are protocols that work reasonably well in practice ...

Nakamotos Idea for Bitcoin: **Proof-of-Work**

- Nodes in P2P network are called **miners**
- In every step a "random" miner is allowed to decide the consensus action and extend blockchain by a block
- More precisely, every miner can add a block to the blockchain as long as
  (1) he is working on longest chain known in the system
  (2) he is the first to solve a computationally hard puzzle
- if two miners solve the puzzle simultaneously, the chain "forks" (splits) and two concurrent chains are being built
- Unlikely to happen. Also every subsequent block must be added to the longest chain known to the miner, so separate chains will not live long

The computational puzzle is based on **cryptographic hash functions**. Such a function

- maps input data to a key of fixed length
- computationally easy to verify the mapping for given input
- computationally superhard to invert
- hashes nicely: even usage of keys, neighboring data yields very different keys

The puzzle: If miner wants to add a block, needs to find a **nonce** for the block to be added

- Nonce: integer number s.t. hash key of the pair (block, nonce) has x leading 0s
- The larger x, the more difficult to find a valid nonce
- Essentially impossible to solve without testing all integers
- x is adjusted based on current hardware/software technology: Single miner should be able to find a nonce on average only every 10 minutes
- If nonce is found, miner adds (block, nonce) to chain, broadcasts result to everyone.
- This process is called **mining**.

[Schema: Hash function, block, nonce]

Reward for mining

- If a nonce is found, **miner allowed to give himself a reward** (i.e., some amount of bitcoin) for computing it
- Reward decreases by factor of 2 every 4 years, stops at 10^{-8} ₿ (= 1 "Satoshi"). Afterwards, there will be no further mining reward.
- This is the only way new bitcoins are created, limits total number to roughly 21m.

Forks and Gamblers Ruin

- Suppose two miners solved the puzzle simultaneously. Others have started to extend one chain. Can a miner overthrow the consensus, make the shorter chain catch up, and turn this fork into the longest one?
- Suppose miner has probability of p to be the fastest one to solve a puzzle
- Probability that he is for k times the fastest solver is only $\left(\frac{p}{1-p}\right)^k$, exponentially small
- Gamblers Ruin: Same calculation as for a gambler that wants to recover a suffered loss in a game with bad odds...
- It is suggested to wait for 6 subsequent blocks (ca. 1h) to consider a block really valid.

[Pic: Fork, longer/shorter chain, probabililty of catching up]

Further aspects and issues:

- Instead of proof of work, there exist alternative approaches that can be used for consensus in blockchains (such as, e.g., **proof of stake**, where permission to add a block is drawn at random with probabililty corresponding to total money of the miner)
- P2P network and bitcoin accounts are **entirely anonymous**. Participants are listed using public keys. Each agent uses her private key to execute transactions, access the money, mining, etc.
- You lose your private key → your money is GONE! Nobody can access the bitcoins. They stay listed in the database, though.
- Lots of computing overhead wasted for generating nonces and bitcoins. Does it make sense? Then again, normal currencies also have overhead...
- What does it mean for a currency that only 21m units exist? Will people keep on mining and investing time/energy of their machines when there is no reward?
- How can a country collect taxes if the currency system is decentral and anonymous?

Blockchains are not currencies – blockchains are **decentralized databases**. Can also be used for contracts, health data, voting procedures, and many more

- There are many "transactions" that need a formal approval by a trusted party
- Example: Contracts. In Germany, many contracts need formal approval by a "Notar" (e.g., when selling/buying houses, inheritance, etc.)
- There are blockchains allowing **smart contracts**, where the approval of the trusted party is generated by majority consensus
- Same principle, blocks contain details about contracts.
- In essence, each block is a (collection of) small programs that implement contract details and, e.g., execute transfers of money, access rights, etc. once they find the prerequisites laid down in the contract to be fulfilled.