Collected Lecture Notes on
Algorithms for Distributed Systems

Sebastian Forster
Department of Computer Science
University of Salzburg

With contributions from Simon Bauer, Ivica Brezovic, András Czuczi, Cansu Demir,
Christian Edelmayer, Florian Feik, Falco Gödde, Sebastian Landl,
Mirna Mrazovic, Michael Nening and Sara Seidl

November 9, 2026

This work is licensed under the Creative Commons Attribution 4.0 International License.

Contents

1 Leader Election in Rings **3**
- 1.1 Problem Statement . . . 3
- 1.2 Communication Model . . . 3
- 1.3 Simple Clockwise Algorithm . . . 4
- 1.4 General Clockwise Algorithm . . . 6
- 1.5 Wait-Time Algorithm . . . 7
- 1.6 Anonymous Networks . . . 8
- 1.7 Radius Growth Algorithm . . . 9
- 1.8 Randomized Algorithm . . . 11
- 1.9 Conclusion . . . 15

2 Introduction to the CONGEST Model **16**
- 2.1 Broadcast . . . 16
- 2.2 Spanning Tree via Breadth-First Search . . . 17
- 2.3 Distance Computation . . . 18
- 2.4 Broadcast via Upcast and Downcast . . . 19
- 2.5 Pipelining . . . 20
- 2.6 Queuing . . . 22

3 Synchronization **24**
- 3.1 Asynchronous Model . . . 24
- 3.2 Synchronizer Framework . . . 24
- 3.3 Synchronizer α . . . 25
- 3.4 Synchronizer β . . . 26
- 3.5 Synchronizer γ . . . 27
- 3.6 Summary . . . 29

4 Computing a Maximal Independent Set **30**
- 4.1 Definitions . . . 30
- 4.2 Sequential Greedy Algorithm . . . 30
- 4.3 Distributed Algorithm . . . 30

5 Graph Spanners **36**
- 5.1 Problem Setting . . . 36
- 5.2 Greedy Spanner . . . 36
- 5.3 Spanner Construction via Clustering . . . 40

6 Computing Shortest Paths **43**
- 6.1 Problem Statement . . . 43
- 6.2 Unweighted Graphs . . . 43
- 6.3 Weighted Graphs . . . 46
- 6.4 Approximation Algorithm for Single-Source Shortest Paths . . . 49

7 Epidemic Information Dissemination **54**
7.1 Introduction . . . 54
7.2 Analysis of the Random-Push Model . . . 55
7.3 Analysis of the Random-Pull Model . . . 60
7.4 Analysis of the Random-Push&Pull Model . . . 60

A Mathematical Background **63**
A.1 Exponents . . . 63
A.2 Logarithms . . . 64
A.3 Miscellaneous . . . 65

B Foundations of Probability Theory **65**
B.1 Discrete Probability Spaces . . . 65
B.2 Random Variables and Expected Values . . . 66
B.3 Statistical Independence and Conditional Probability . . . 67
B.4 Probability Distributions . . . 67
B.5 Concentration Inequalities . . . 71

1 Leader Election in Rings

1.1 Problem Statement

A *ring* (network) consists of n nodes and n edges, where each node has two (ports to) neighbors. One neighbor is located "clockwise" (*Clockwise*) and the other neighbor is located "counter-clockwise" (*Counter-Clockwise*). An example of such a network can be found in Fig. 1.1.

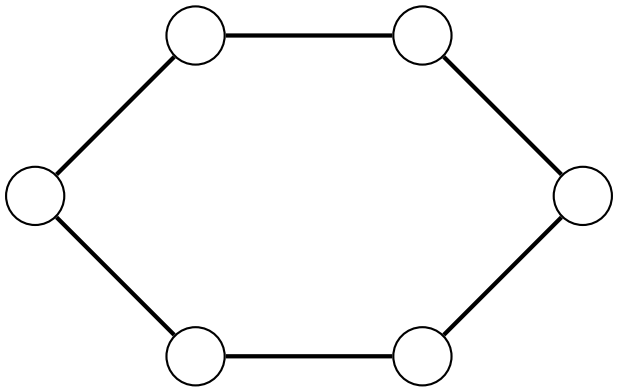


Figure 1.1: Example of a ring with 6 nodes

In the *leader election* problem, the nodes of a network must agree on a single *leader*. All other nodes of the network are then called *followers*. Thus, we are looking for an algorithm that decides for each node whether it is the leader or a follower, with exactly one node being the leader overall. The purpose of leader election can be to allow the leader to perform coordination tasks in a (decentralized) network.

It may initially seem trivial to select a leader, but in a decentralized model this is not so straightforward when every node is initially equivalent and thus every node is suitable as a leader. There is no "better" or "worse" leader in such networks. Therefore, the solution is not unique, and in particular, for a network with n nodes there are n distinct solutions. The difficulty for the nodes of the network is therefore actually to reach a consistent decision for one of these solutions.

1.2 Communication Model

In our message-passing communication model, there are n nodes in the network. Each node knows the number of its neighbors and can send and receive arbitrary messages to and from its neighbors via numbered ports. A port is a connection from one node to a neighboring node. The communication can proceed in the following two ways.

1. *Synchronous:* In the *synchronous* model, communication is round-based. Each round consists of the following steps:

   (a) Receive messages from neighbors (skipped in the first round)

   (b) Local computation

   (c) Send messages to neighbors

   There is a global clock that dictates the beginning of each round.

2. *Asynchronous:* In the *asynchronous* model, communication is event-based. Nodes become active whenever they receive a message or when an external initialization event[1] occurs. Once a node becomes active, it may perform local computations and then send messages to its neighbors, after which it becomes inactive again – until activated by the next event. In the asynchronous model, it is only guaranteed that every message transmission takes finite time. However, in particular, there is no guarantee that every message transmission takes the same amount of time.

The synchronous model can be understood as a special case of the asynchronous model, in which an external initialization event occurs for all nodes simultaneously and every message transmission takes exactly one time unit.

In these models, we consider two measures of complexity, namely *time complexity* and *message complexity*. The time complexity of an algorithm in the synchronous model gives the number of *rounds* until the problem is solved. The time complexity of an algorithm in the asynchronous model gives the number of *time units* from the last initial activation of a node until the problem is solved, assuming every message transmission takes at most one time unit. It is important to emphasize that internal computation time is *not* measured with time complexity, so it is excluded from our analyses. The message complexity of an algorithm gives the total number of messages sent by all nodes in both models.

1.2.1 Further Variants

One can distinguish between a synchronous and an asynchronous model (see above). Similarly, nodes can be anonymous or identifiable. A node is identifiable if it has a unique ID (often called UID for *unique* ID). Typically, such IDs are bit strings, often of length $O(\log n)$. The IDs need not be assigned contiguously; there may be gaps (not all possible numeric values need be assigned). Furthermore, one can also distinguish whether the number of nodes n is global knowledge or not. If the number of nodes n is not global knowledge, then the model is called *uniform*.

1.3 Simple Clockwise Algorithm

For the leader election algorithm that we present below, we leave the setting of anonymous networks and now assume that every node v has a unique ID $\mathrm{ID}(v)$. We first formulate the algorithm for synchronous, non-uniform ring networks.

In the *(simple) clockwise algorithm*, each node v behaves as follows:

Round 1:

1 v sets $L_v := \mathrm{ID}(v)$ (local variable for the smallest ID seen so far)
2 v sends L_v to neighbor clockwise

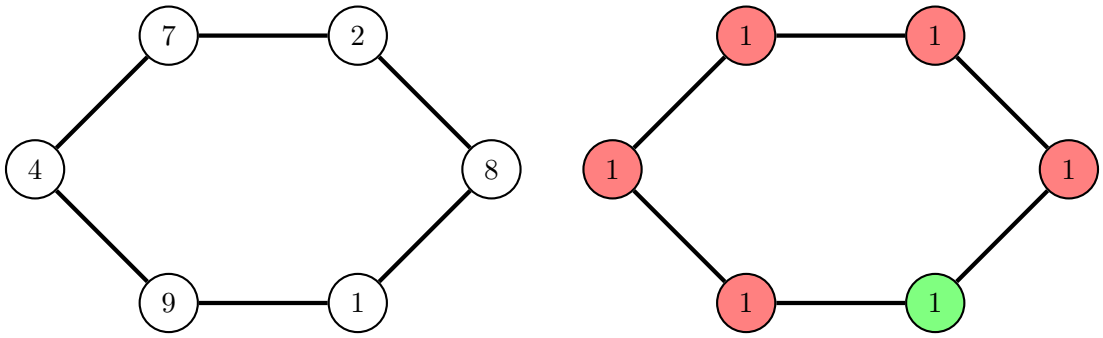
Round $2 \leq r \leq n$:

[1] The occurrence of this external initialization event is necessary for at least one node so that algorithms in this model can start. Once a node has become active for the first time by receiving a message, the external initialization event is no longer necessary for it.

```
if v receives message M then
    if M < L_v then
        v sets L_v := M
        v sends L_v to neighbor clockwise
```

Round $n+1$:

```
if L_v = ID(v) then
    v becomes the leader
else
    v becomes a follower
```



(a) Initially assigned IDs

(b) Assignment to leaders (green) and followers (red)

Figure 1.2: Example of leader election in a ring

We consider the example in Fig. 1.2. The numbers in Fig. 1.2a are the node IDs. These are pre-assigned and unique. In the first round, we initialize for each node the local variable $L_v = \mathrm{ID}(v)$ and send a message to the clockwise neighbor with the content L_v. In the next round, messages are received, and each node that receives a value higher than its own value adopts the higher (i.e., received) value. For example, the node at the lower left, which previously had value 9, adopts the received value 1. All nodes that have changed their value send the new value clockwise to their neighbor. The algorithm continues in this way, and at the end every node has set L_v to 1, and the assignment to leaders and followers is made. In the result in Fig. 1.2b, followers are marked in red and the leader in green. The node with ID 1 became the leader because 1 is the smallest ID in the network.

1.3.1 Analysis

Theorem 1.1. *After $n+1$ rounds, the simple clockwise algorithm makes the node with the smallest ID the leader and all others followers.*

Proof. We want to formalize in the proof the observation that the smallest ID is forwarded in every round. Let z be the node with the (initially) smallest ID. For each $i \geq 0$, let v_i be the node reached from z by traversing i edges clockwise. In other words, node v_i is the node that can be reached from node z by taking i edges clockwise. Node v_0 would be z itself, since 0 edges are needed to reach z from z. Node v_1 would be the clockwise neighbor of z, and so on. We now use the following inductive hypothesis.

Inductive hypothesis: Node v_i sets L_{v_i} to $\mathrm{ID}(z)$ (at the latest) in round $i+1$ (for every $0 \leq i \leq n$).

This statement can be proved by mathematical induction. We will not present this part of the proof here, since it only requires standard arguments.

We now observe that any value assumed by L_v corresponds to an ID in the network.[2] Once a node v receives a message with content $\mathrm{ID}(z)$, then $L_v = \mathrm{ID}(z)$ is set and L_v will not assume any other value thereafter.

Since for every node $v \neq z$ there exists an i with $1 \leq i \leq n-1$ such that $v = v_i$, it follows from the inductive hypothesis that v receives message $\mathrm{ID}(z)$ within $i+1 \leq n$ rounds. As argued just above, from that point onward $L_v = \mathrm{ID}(z) \neq \mathrm{ID}(v)$. In round $n+1$, $L_v = \mathrm{ID}(v)$ holds only for node $v = z$. Therefore, z is correctly designated as the leader and the other nodes as followers. □

Theorem 1.2. *The clockwise algorithm sends at most n^2 messages in total.*

Proof. When are messages sent in the algorithm? A node v sends only immediately after the variable L_v has been changed. That is, node v sends only after L_v has been initialized or increased. Each value assumed by L_v corresponds to an ID in the network. Therefore, L_v can assume at most n distinct values during the algorithm, since there are n distinct nodes. Hence, each node sends at most n messages. Since each node sends at most n messages and there are n nodes, the algorithm thus has a message complexity of n^2. □

1.4 General Clockwise Algorithm

The clockwise algorithm can also be adapted to the asynchronous, uniform model [**Lann77**, **ChangR79**]. As before, we assume that every node v has a unique ID $\mathrm{ID}(v)$. In the asynchronous model, we do not consider rounds but events. There are two possible events: initialization or receipt of a message. An (external) initialization event is necessary to start the algorithm, and multiple nodes can be initialized simultaneously. In contrast to the synchronous and non-uniform model, the nodes no longer know when the smallest ID has completed a full traversal of the ring. Therefore, nodes become followers as soon as they receive an ID smaller than their own. The leader, on the other hand, recognizes itself by the fact that it receives its own ID after this ID has completed a full traversal of the ring. The algorithm for each node v is as follows:

```
1  if v is initialized then
2      v sets L_v := ID(v) (local variable for the smallest ID seen so far)
3      v sends L_v to neighbor clockwise
4  if v receives message M then
5      if v is uninitialized then initialize v
6      if M < L_v then
7          v sets L_v := M
8          v becomes a follower (if not already done so earlier)
9          v sends L_v to neighbor clockwise
10     if M = ID(v) then
11         v becomes the leader
```

[2] One might argue that in a fully formal proof, we would need to prove this statement by induction.

1.4.1 Analysis

Theorem 1.3. *At most $2n-1$ time units after the first activation of a node, the asynchronous clockwise algorithm designates the node with the smallest ID as the leader and all others as followers (assuming each message transmission takes at most one time unit).*

Proof sketch. We start the timing with the first node for which an external initialization event occurs. At this first activation, the node will definitely send a message (its ID). That is, it wakes up the neighboring node. Thus, the neighboring node becomes active and the mutual activation continues. Then at most $n-1$ time units elapse until node z with the smallest ID receives a message and becomes active for the first time. When the node with the smallest ID has become active for the first time, the argument is similar to the synchronous case: after at most n additional time units, every node has received $\mathrm{ID}(z)$ and made a decision. □

1.5 Wait-Time Algorithm

It is clear that every leader election algorithm in a ring requires $\Omega(n)$ rounds, since otherwise relevant information from one part of the network could not reach all nodes. Therefore, an asymptotic time complexity of $O(n)$ is optimal. For message complexity, the trivial lower bound of $\Omega(n)$ also applies, since every node must receive at least one message. The gap between this lower bound and the message complexity of $O(n^2)$ of the clockwise algorithm is very large. One can thus ask whether it is possible to determine a leader with significantly fewer than n^2 messages. It is indeed. The idea behind the solution that we present below is to exploit the presence of synchrony. In particular, it is explicitly interpreted as consent in the algorithm when *no* message is sent.

1.5.1 Algorithm

For the corresponding algorithm [**AttiyaW98**], each node v executes the following instructions in each round:

```
if leader message received then
    Become a follower
    Forward leader message clockwise
if #Rounds = ID(v) · n + 1 and v is not yet a follower then
    Become the leader
    Send leader message to neighbor clockwise
```

A node v thus becomes active when the current round equals $\mathrm{ID}(v) \cdot n + 1$. (The offset +1 is necessary because rounds are counted starting from 1 and IDs start from 0.) If so, the node becomes the leader and sends a message to its neighbor indicating that a leader has been determined. This message is forwarded in subsequent rounds, and every node that receives the message becomes a follower.

1.5.2 Analysis

We divide the rounds into phases of length n, and in each phase (i.e., every n rounds) at most one node decides to become the leader. For the algorithm to be correct, the leader must have informed all other nodes before the next phase begins, i.e., within n rounds. This is guaranteed by forwarding the message around the ring. Once a node decides to be the leader, we know that

before the beginning of the next phase all other nodes have become followers. It follows that the number of messages is $O(n)$. (In fact, it holds that there are at most n messages, since each node sends only once.) The number of rounds is $\Theta(n \cdot \min_v \mathrm{ID}(v))$ and essentially indicates how long we must count up to reach an ID that was assigned in the network.[3]

Remark. Already for IDs that can be interpreted as integers in the range from 0 to $2n-1$, the smallest ID could have the numeric value n. Then the algorithm requires $\Theta(n^2)$ rounds.

1.6 Anonymous Networks

In an *anonymous network*, the nodes have no IDs. A priori distinctions are therefore not possible. It turns out that this restriction is too strong to determine a leader in a ring.

Theorem 1.4 ([**Angluin80**]). *Leader election in anonymous networks is impossible with deterministic algorithms, even in the synchronous ring.*

To prove this theorem, one must of course first define exactly what is meant by *deterministic algorithms* in our distributed model. We define a deterministic algorithm as follows:

A deterministic algorithm is describable by a function f, which is applied repeatedly to each node v. The *input* to this function is the number of nodes n, the number of ports of v, and the entire history of v. The history of a node is the time-ordered protocol of all previously received messages, recording the content of the message and the input port, with in the synchronous model additionally recording the round number at reception. The *output* of the function is the messages to be sent by the node via output ports and, if applicable, the decision of whether v becomes the leader or a follower. We then say that the function solves the problem if for every network after finitely many applications of f, one node is designated as leader and all others as followers.

Deterministic here means that f is a "genuine" function that maps every possible input to a unique output. Non-deterministic would mean that there are inputs with more than one possible output, as can occur with randomized algorithms.

1.6.1 Proof of Theorem 1.4

Proof. We will now prove the impossibility, i.e., the theorem above. For this we need the following inductive hypothesis, which is proved after the impossibility proof itself. For now, we work under the assumption that the inductive hypothesis is correct.

Inductive Hypothesis. *For any deterministic algorithm, in every round the messages received clockwise and counter-clockwise are the same for all nodes.*

From the inductive hypothesis it follows that in every round all nodes have the same history. Moreover, the number of ports is the same for every node, and in every round the input to function f is the same for all nodes. Since f is deterministic and the input is the same for all nodes, it follows that the output of function f is also the same for all nodes. If f makes a decision, then this decision must be consistent for all nodes. This thus means: in every round, f either makes all nodes leaders, all nodes followers, or makes no decision for any node. Ultimately, this implies that f either produces an incorrect result or does not terminate, thereby proving the aforementioned theorem (under the assumption that the inductive hypothesis holds). □

[3] Recall: The IDs are typically bit strings, and the smallest ID in the network does not necessarily have to be 0.

1.6.2 Proof of the Inductive Hypothesis

We will now prove the inductive hypothesis stated in the impossibility proof.

Proof of the inductive hypothesis. Base case: Holds trivially, since no messages are received in the first round.

Inductive step: We now show that the inductive hypothesis holds for every round $k \geq 2$, under the assumption that it holds for all rounds $i \leq k-1$. From the validity of the inductive hypothesis for all previous rounds, it follows that every node has the same history at the beginning of round k. Moreover, all other parameters are the same for all nodes (number of ports, number of nodes n). Thus, the input to function f in round k is the same for all nodes. From the same input it follows the same output, so the output of function f is the same for all nodes. This means that all nodes send the same respective messages clockwise and counter-clockwise in round k. From the ring topology it follows: in round k, all nodes receive the same respective messages clockwise and counter-clockwise. Thus, the inductive hypothesis is proved and the preceding impossibility proof is complete. □

In summary, this means that no node can become a leader or follower without all other nodes making the same decision, which contradicts the goal of designating a single leader. We once again point out that this impossibility does not hold for all networks, but only for anonymous rings.

1.7 Radius Growth Algorithm

So far, we have learned about the clockwise algorithm, which requires $O(n)$ rounds and $O(n^2)$ messages to determine a leader. We have also learned a more message-efficient algorithm, which (under certain assumptions) takes considerably more rounds but requires only $O(n)$ messages. We now present an algorithm that approximately combines the low round count and the low message complexity of these two algorithms, requiring $O(n)$ rounds and $O(n \log n)$ messages.

The radius growth algorithm [**HirschbergS80**] for leader election in rings works on the following principle: each node checks whether it is the leader in a certain neighborhood, which means that it has the smallest ID in that neighborhood. If so, this neighborhood is expanded.

The starting situation of the algorithm is that each of the n nodes is a candidate for leader. Throughout the algorithm, $n-1$ nodes will decide to be followers, and the remaining node will be the leader. Nodes that are not yet followers we call **active**.

The rounds are divided into $\lceil \log n \rceil$ consecutive phases, where phase i lasts $2^{i-1}+1$ rounds. In the first round of each phase, every active node sends its ID to its neighbors in the ring. In all other rounds, when a node receives a message, it forwards it and decides to be a follower if the received ID is larger than its own. The node that is still active at the end becomes the leader.

First round of each phase:

```
1 if v is not yet a follower then
2     v sends ID(v) to neighbors clockwise and counter-clockwise
```

Every other round:

```
if v receives message M from neighbor counter-clockwise (clockwise) then
    if M < ID(v) then
        v becomes a follower (if not already done so earlier)
        if not the last round of the phase then
            v sends M to neighbor counter-clockwise (clockwise)
```

Additionally, in the last round of the last phase:

```
if v is not yet a follower then v becomes the leader
```

1.7.1 Correctness and Runtime

Theorem 1.5. *The radius growth algorithm determines a unique leader.*

Proof. Let z be the node with the smallest ID. Clearly, node z never receives a message with an ID that is smaller than its own. Therefore, it never becomes a follower and remains active until the last round of the last phase, whereupon it becomes the leader. In particular, z is still active in phase $\ell = \lceil \log n \rceil$, the last phase. In this phase, z sends a message with its ID clockwise and counter-clockwise, each reaching $2^{\ell-1}$ nodes. Because

$$2^{\ell-1} = 2^{\lceil \log n \rceil - 1} \geq 2^{\log n - 1} = \frac{2^{\log n}}{2} = \frac{n}{2}$$

in phase ℓ, the message with the ID of z reaches at least $n/2$ nodes in each direction, and thus all other nodes, which then become followers since they have a higher ID than z. Thus, the algorithm determines a valid assignment to leaders and followers. □

Theorem 1.6. *The radius growth algorithm requires $O(n)$ rounds.*

Proof. Phase i requires $2^{i-1} + 1 \leq 2^i$ rounds (generously bounded). Thus, the total number of rounds is: $\sum_{i=1}^{\lceil \log n \rceil} 2^i \leq 2^{\lceil \log n \rceil + 1} \leq 2^{\log n + 2} \leq 4 \cdot 2^{\log n} = 4n$. This follows from the summation formula for geometric series. □

Lemma 1.7. *For every active node, at the end of phase i: all other nodes within distance up to 2^{i-1} are inactive.*

The statement of this lemma is that at the end of phase i in the area of the nearest 2^{i-1} nodes, there is a unique leader.

Proof. We proceed by contradiction. Let v be a node that is still active at the end of phase i. Suppose there exists a node w with distance at most 2^{i-1} to node v that is still active at the end of phase i. Then w was also active at the beginning of phase i.

We fundamentally distinguish between three cases here:

- If $\mathrm{ID}(v) > \mathrm{ID}(w)$, then without loss of generality let w be the node with the smallest ID among all nodes within distance at most 2^{i-1} to node v. Then v receives a message with $\mathrm{ID}(w)$ in phase i and therefore becomes inactive. This contradicts the assumption that v is still active at the end of phase i.

- If $\mathrm{ID}(v) < \mathrm{ID}(w)$, then w receives a message with $\mathrm{ID}(v)$ in phase i and therefore becomes inactive. This contradicts the assumption that w is still active at the end of phase i.
- The case $\mathrm{ID}(w) = \mathrm{ID}(v)$ cannot occur, since the IDs are unique.

□

Lemma 1.8. *For $i \geq 2$, the number of active nodes at the beginning of phase i is at most $n/2^{i-2}$.*

Proof. By Lemma 1.7, we know that at the end of phase i, to each active node we can uniquely assign the 2^{i-2} nearest inactive nodes clockwise (see Fig. 1.3). Let a be the number of active nodes, then it holds that $a \cdot 2^{i-2} \leq n$, because a multiplied by the number of inactive nodes assigned to a cannot exceed the total number of nodes in the ring. Hence, $a \leq n/2^{i-2}$.

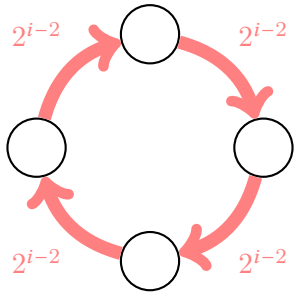


Figure 1.3: Visualization of the unique assignment of inactive nodes to active nodes

□

Theorem 1.9. *The radius growth algorithm sends at most $O(n \log n)$ messages.*

Proof. At the beginning of each phase, every active node starts two messages, one clockwise and one counter-clockwise. These messages are forwarded once in each round of the phase. Thus, the number of messages in phase $i \geq 2$ is at most $\frac{n}{2^{i-2}} \cdot 2 \cdot 2^{i-1} = 4n$ ($\frac{n}{2^{i-2}}$ is the maximum number of active nodes (Lemma 1.8); each active node sends 2 messages; the number of rounds of phase i is $2^{i-1} + 1$, and in each of these rounds, except the last round of the phase, the messages generated at the beginning are forwarded once). In phase 1, the number of messages is clearly $2n$. Thus, in each phase at most $4n$ messages are sent. Since there are $\lceil \log n \rceil$ phases, the message complexity of the radius growth algorithm is in $O(n \log n)$. □

1.7.2 Summary

The node with the smallest ID is always selected as the leader. The time complexity is $O(n)$ rounds, while the message complexity is $O(n \log n)$. The radius growth algorithm can also be formulated as an asynchronous algorithm.

1.8 Randomized Algorithm

At the outset, we established that it is not possible to determine a leader in an anonymous ring using a deterministic algorithm. We will now return to this setting and learn a *randomized* algorithm that determines a leader.

In the anonymous ring, no IDs are assigned a priori. However, we assume that the total number n of nodes in the ring is known to every node. The basic idea of the randomized algorithm [**ItaiR90**, **AttiyaW98**] is now that we work with IDs that have been randomly assigned, and proceed as follows:

1. Each node randomly and uniformly selects an ID from 0 to $2n-1$
2. Nodes execute the simple clockwise algorithm with the chosen IDs
3. Nodes verify the result

The problem with the random assignment of IDs is that there is no guarantee that no ID is assigned multiple times, and thus there could be multiple nodes with the smallest ID. This would lead to an incorrect result, because more than a single, unique leader would be designated. To solve this problem, the nodes must verify in step 3 whether the result, i.e., the assignment to leaders and followers, is correct (i.e., there is exactly one leader and only followers otherwise), which is possible in $O(n)$ rounds and with $O(n)$ messages (see exercise). If the result is not correct, then we must start over from step 1. These steps are repeated until a unique leader has been determined.

1.8.1 Success Probability of an Iteration

An iteration of the algorithm is successful if and only if the smallest assigned ID was assigned exactly once. However, it is somewhat difficult to determine the success probability exactly. Therefore, we apply the following analytic trick: We know that if ID 0 is assigned exactly once, the corresponding iteration is successful. Now we simply disregard the possibility that an ID strictly greater than 0 is assigned exactly once and is the smallest ID, which is not a problem because we thereby do not overestimate our success probability.

We now consider for each node $v \in V$ the random variable

$$X_v = \begin{cases} 1 & \text{if node } v \text{ receives ID } 0 \\ 0 & \text{otherwise} \end{cases}$$

and the random variable $X := \sum_{v \in V} X_v$ for the number of nodes receiving ID 0. Since each node chooses its ID uniformly at random from the range 0 to $2n-1$, we have $\Pr[X_v = 1] = \frac{1}{2n}$. In other words, X_v is Bernoulli-distributed with success probability $p := \frac{1}{2n}$.

Since the assignment of IDs for each node occurs separately, the random variables X_v are stochastically independent. The sum of these random variables, namely the random variable X, is therefore binomially distributed (with parameters n and p). Therefore, $\Pr[X = k] = \binom{n}{k} p^k (1-p)^{n-k}$ for all $0 \leq k \leq n$. In particular, for $k = 1$ we obtain the probability that ID 0 is assigned exactly once. For this probability we now make the following estimate:

$$\Pr[X = k] = n \cdot p(1-p)^{n-1} = \frac{1}{2} \cdot \left(1 - \frac{1}{2n}\right)^{n-1} \geq \frac{1}{2} \cdot \left(1 - \frac{n-1}{2n}\right) \geq \frac{1}{2} \cdot \left(1 - \frac{n}{2n}\right) = \frac{1}{4}.$$

Here we have applied the *Bernoulli inequality*, which in general states that $(1+x)^n \geq 1 + xn$ for $x \geq -1$ and $n \geq 0$ (see (A.16)). Thus, each iteration is successful with probability $q \geq \frac{1}{4}$.

1.8.2 Correctness

Due to the verification step at the end of each iteration, it is clear that the algorithm produces a correct result once it terminates. To prove the correctness of the algorithm, we therefore need to show that it terminates. However, it would be wrong to claim that the algorithm always terminates. It is in principle possible that in every iteration of the algorithm the random choice of IDs was unfavorable, and thus the algorithm never terminates. However, this event is highly unlikely – it has probability 0. Formally, we prove the correctness of our algorithm by showing that it terminates with probability 1.

Theorem 1.10. *The randomized leader election algorithm terminates with probability* 1.

Proof. Let A_k be the event that the algorithm terminates after (exactly) k iterations (for $k \geq 1$). Accordingly, $\bigcup_{k \in \mathbb{N}^{\geq 1}} A_k$ is the event that the algorithm eventually terminates. Since the events A_k are disjoint (for $k_1 \neq k_2$, the algorithm cannot terminate after both k_1 rounds and k_2 rounds), we have:

$$\Pr\left[\bigcup_{k \in \mathbb{N}^{\geq 1}} A_k\right] = \sum_{k \in \mathbb{N}^{\geq 1}} A_k = \sum_{k=1}^{\infty} (1-q)^{k-1} q = q \cdot \sum_{k=1}^{\infty} (1-q)^{k-1} = q \cdot \sum_{k=0}^{\infty} (1-q)^{k}$$
$$= q \cdot \frac{1}{1-(1-q)} = q \cdot \frac{1}{q} = 1\,.$$

The simplification $\sum_{k=1}^{\infty} (1-q)^{k-1} = \frac{1}{1-(1-q)}$ follows from the limit of the geometric series (see (A.15)). □

1.8.3 Runtime

Our analysis so far has shown that this algorithm eventually terminates. Now let us turn to a more detailed analysis of the runtime of the algorithm. First, we want to make a statement about the expected value of the number of rounds of the algorithm. Since each iteration of the algorithm requires $O(n)$ rounds, it suffices to analyze the expected number of iterations of the algorithm.

Theorem 1.11. *The expected number of iterations of the randomized leader election algorithm is* $O(1)$.

Proof. For this analysis, we first define the following random variables Y_i (for $i \geq 1$):

$$Y_i = \begin{cases} 1 & \text{if the } i\text{-th iteration of the algorithm is successful} \\ 0 & \text{otherwise} \end{cases}$$

We also define the random variable Z as the smallest index i for which $Y_i = 1$ holds, i.e., $Z = \min\{i \geq 1 \mid Y_i = 1\}$. Thus, Z corresponds to the number of iterations of the algorithm.

We already know that in each iteration we have a constant success probability of $q \geq \frac{1}{4}$, i.e., $\Pr[Y_i = 1] = q$ for all $i \geq 1$. Since the choice of IDs occurs separately in each iteration, the Y_i's are stochastically independent, and therefore Z is geometrically distributed with parameter q. The expected value of such a random variable is $\mathrm{Ex}[Z] = \frac{1}{q} \leq 4$ (see Theorem B.10). Thus, it follows that the randomized leader election algorithm requires at most $4 = O(1)$ iterations in expectation. □

Sometimes, however, an analysis of the expected value is not sufficient, since there could be a large variance around this expected value and thus many outliers on the high side. Although one has a constant expected number of iterations in expectation, the actual number of iterations can often be significantly larger.

Therefore, we investigate not only how many iterations are necessary with constant probability, but also how many iterations are needed "with high probability." Thus, our measure of success is not a constant probability but an inverse polynomial probability.

Theorem 1.12. *For every $c \geq 1$: With probability at least $1 - \frac{1}{n^c}$ ("with high probability"), the randomized leader election algorithm requires $O(c \log n)$ iterations.*

Proof. Let Z be the random variable indicating how many iterations the algorithm performs. To prove the theorem, we show that for $k = \lceil c \log_{4/3} n \rceil = \lceil \log_{4/3} n^c \rceil$, we have: $\Pr[Z \leq k] \geq 1 - \frac{1}{n^c}$.

The random variable Z is geometrically distributed, with parameter $q \geq \frac{1}{4}$. The probability of the complement event $Z > k$ can therefore be bounded as follows:

$$\Pr[Z > k] = (1-q)^k \leq \left(1 - \frac{1}{4}\right)^k = \left(\frac{3}{4}\right)^k \leq \left(\frac{3}{4}\right)^{\log_{4/3} n^c} = \left(\frac{1}{4/3}\right)^{\log_{4/3} n^c} = \frac{1}{(4/3)^{\log_{4/3} n^c}} = \frac{1}{n^c}.$$

Thus, we obtain $\Pr[Z \leq k] = 1 - \Pr[Z > k] \geq 1 - \frac{1}{n^c}$. □

1.8.4 General Principle

In our randomized leader election algorithm, a general pattern can be observed for employing a faulty randomized algorithm in such a way that it always produces a correct result.

Theorem 1.13 (Monte Carlo → Las Vegas)**.**

- *Let $\mathcal{A}$ be a randomized algorithm for a problem $\mathcal{P}$ that computes a correct solution for every input with probability q.*
- *Let $\mathcal{V}$ be a verifier that can determine for every output of $\mathcal{A}$ whether the solution is correct.*
- *Then there exists an algorithm $\mathcal{B}$ that terminates with probability 1 and computes a correct solution, calling $\mathcal{A}$ and $\mathcal{V}$ in expectation $O(1/q)$ times.*

Definition 1.14. *A randomized algorithm $\mathcal{B}$ whose result is always correct is called a* Las Vegas *algorithm. A randomized algorithm $\mathcal{A}$ whose result is likely to be correct is called a* Monte Carlo *algorithm.*

With a Monte Carlo algorithm, the randomization affects whether the algorithm produces a correct result at the end or not. With a Las Vegas algorithm, the randomization affects the consumed resources (number of rounds needed, message complexity, runtime, memory), and such an algorithm always produces a correct result. Caution: Not every Monte Carlo algorithm can be converted into a Las Vegas algorithm, since it is not always trivial to find a verifier.

1.9 Conclusion

Even in the simple task of determining a leader in a ring, many principles and techniques of distributed algorithms become apparent. First, there is a wide variety of modeling variations. These range from synchronous and asynchronous to anonymous and uniform models. We also distinguish between deterministic and randomized algorithms. Another important aspect is the complexity measures: here we distinguish between the time required and the number of messages sent. We have learned two fundamental ways to break symmetries: on the one hand, through unique IDs, and on the other hand, through randomization. This section is partially based on lectures by Robert Elsässer and Stefan Schmid.

2 Introduction to the CONGEST Model

In the CONGEST model, the network is modeled as an undirected, connected graph. Communication, i.e., the sending of messages between the n nodes along m edges, takes place in synchronized rounds, whereby each node can send at most one message to each neighbor per round. In the CONGEST model, the message size (bandwidth) is limited to $O(\log n)$ bits. Furthermore, every node in the graph has a unique ID consisting of $O(\log n)$ bits. This ID is known to the node itself and to its neighbors. Additionally, every node knows the number of nodes n. Formally, an algorithm is considered terminated as soon as all nodes have transitioned into a special termination state, from which they never leave and in which they no longer send any messages. For the sake of simpler exposition, we will usually not explicitly specify the transition of a node into the termination state in the context of this course, since it is generally obvious when this should take place.[4]

2.1 Broadcast

For the broadcast algorithm, it is assumed that there is exactly one source node s that wishes to send information I of size $O(\log n)$ to all nodes in the graph. The algorithm can be formulated as follows:

---

1 s sends information I to all neighbors in round 1
2 Whenever some node $v \neq s$ receives information I for the first time: v sends I to all neighbors

---

The algorithm begins with the source node s. This node sends I to all immediate neighbors with which it is connected. These nodes in turn send I to their neighbors, until I has also reached the nodes farthest from s. Nodes that have already received the information may ignore a subsequent reception of the information. An example of the execution of one round of the algorithm is given in Fig. 2.1.

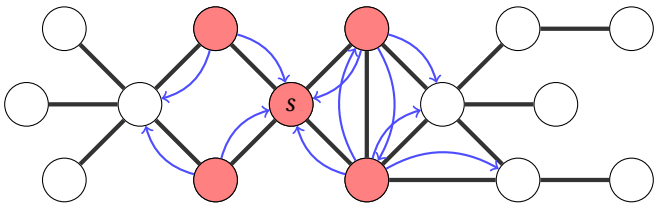


Figure 2.1: In this example, the second round of the broadcast algorithm is executed on the given network.

2.1.1 Correctness

The inductive claim to be proven for correctness is:

For every node $v \neq s$, node v receives at the latest in round $\mathrm{dist}(s, v) + 1$ the information I.

Thus, the round in which v receives the information at the latest depends on the distance of the node to the source node s.

[4] For example, if we have proven that an algorithm has computed the correct result after $R(n)$ rounds, every node can transition into the termination state after $R(n)$ rounds.

Note: The distance from s to v is the length of the shortest path from s to v. In round $r+1$, all nodes that are at r edges from the source node s receive the information I.

The correctness of the algorithm follows from the inductive claim: Since the network is connected, every node has finite distance to s. This means that every node receives I after a finite number of rounds.

To prove the inductive claim, we must prove it for $v \neq s$. For nodes $v \neq s$, it holds that $\mathrm{dist}(s,v) \geq 1$.

Inductive basis:

It holds that $\mathrm{dist}(s,v) = 1$. A node v at distance 1 from s is a neighbor of s. Neighbors of s receive the information in round $2 = \mathrm{dist}(s,v) + 1$.

Inductive step:

We consider the case $\mathrm{dist}(s,v) \geq 2$. Let u be the predecessor of v on the shortest path from s to v. For these nodes: $\mathrm{dist}(s,u) = \mathrm{dist}(s,v) - 1$. Thus, the distance from s to u is strictly smaller than the distance from s to v. We can therefore apply the inductive claim to u.

By the inductive claim, u has received the information I at the latest in round $\mathrm{dist}(s,u)+1$ and then sent it to all neighbors. Since v is a neighbor of u, v receives the information I at the latest in round $\mathrm{dist}(s,u) + 2 = \mathrm{dist}(s,v) + 1$.

2.1.2 Running Time

The algorithm runs as long as – i.e., has exactly as many rounds as – it takes until every node has received the information. Thus, the node farthest from s, let us call it v, must be reached. Each node receives the information after at most $\mathrm{dist}(s,v)+1$ rounds. Therefore, for the maximum number of rounds required: $r_{\max} = \max_v \mathrm{dist}(s,v) + 1$.

The eccentricity of a node u is defined as $\mathrm{Ecc}(u) = \max_v \mathrm{dist}(u,v)$, i.e., the number of edges of the longest shortest path originating from u. The diameter of the network is the maximum distance between a pair of nodes in the network, i.e., $D = \max_{u,v \in V} \mathrm{dist}(u,v)$. It thus specifies how long the shortest path is that exists in the network.

Summarizing, we obtain with these two definitions $D = \max_u \mathrm{Ecc}(u)$, and therefore $\mathrm{Ecc}(u) \leq D$. The diameter is thus equal to the maximum eccentricity of any node, and therefore it follows that the eccentricity of every node is always less than or equal to the diameter of the network. Thus, the running time of the broadcast algorithm is $O(\mathrm{Ecc}(s)) = O(D) = O(n)$ rounds.

2.1.3 Message Complexity

In the algorithm, the information I is forwarded to all neighbors upon first receipt. This means that each node sends I exactly as many times as it has incident edges to its neighbors. This corresponds to the so-called degree of node v, i.e., $\deg(v)$. In total, $\sum_{v \in V} \deg(v) = 2m = O(m)$ messages are sent; the upper bound follows from the fact that the sum of the node degrees equals $2m$.

2.2 Spanning Tree via Breadth-First Search

The goal is to compute a tree that contains all nodes in an arbitrary connected graph. In particular, parent-child relationships must be established without forming any cycles. We achieve our goal by computing a breadth-first search tree, for which any information I is sent from the source node s:

1 s sends information I to all neighbors in round 1
2 Whenever some node $v \neq s$ receives information I for the first time:
3 v stores one of the senders of I as its parent node
4 v sends a "child" message to its parent node and I to all other neighbors

An example of the execution of one round of the algorithm is given in Fig. 6.1. To ensure determinism, for example in line 2 the node with the smaller ID can be preferred as the parent node. After the complete execution of the algorithm, an implicit spanning tree has been created, since every node knows its parent and its children. No cycle has been created since all nodes (except s) in the graph have chosen exactly one parent node with a smaller distance to s.

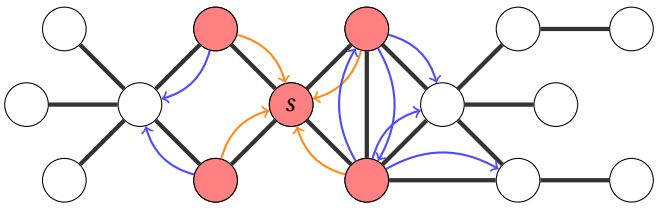


Figure 2.2: In this example, the second round of the algorithm for computing a breadth-first search tree is executed on the given network.

2.3 Distance Computation

For distance computation, the algorithm for computing a breadth-first search tree is slightly modified/extended. Instead of sending a child message, the parent nodes send their children the distance they should assign to themselves.

1 s has distance 0 to itself and in round 1 sends distance information 1 to all neighbors
2 Whenever some node $v \neq s$ receives distance information d for the first time:
3 v stores one of the senders of d as its parent node
4 v sends a "child" message to its parent node and distance information $d+1$ to all other neighbors

Through this algorithm, every node in the graph receives a parent node (except s) and also gets a distance assigned by exactly that parent. If a node that has already been assigned a distance receives another, higher distance, it ignores it. An example of the execution of one round of the algorithm is given in Fig. 2.3.

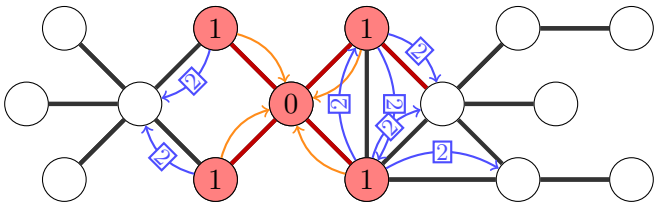


Figure 2.3: In this example, the second round of the distance computation algorithm is executed on the given network.

2.3.1 Summary of Breadth-First Search

The following properties hold for a breadth-first search with source node s:

- *Number of rounds:* $O(\mathrm{Ecc}(s)) = O(D)$

  The algorithm terminates as soon as all nodes have been assigned a parent node and all child messages have arrived at the parent nodes. The maximum distance of a leaf from s in the breadth-first search tree of s is $\mathrm{Ecc}(s)$. Therefore, the running time is $O(\mathrm{Ecc}(s)) = O(D)$.

- *Number of messages:* $O(m)$

  At most one message is sent over each edge, i.e., at most $2m$ messages in total, which is $O(m)$.

- *Every node v knows $\mathrm{dist}(s, v)$, its distance to s:*

  This is the result of the distributed algorithm for distance computation.

- *For the breadth-first search tree:*

  - Every node $u \neq s$ has a parent node v with $\mathrm{dist}(s, v) = \mathrm{dist}(s, u) - 1$
  - Every node knows its parent node and its children in the tree
  - Every breadth-first search tree is also a spanning tree

2.4 Broadcast via Upcast and Downcast

We consider the following initial situation: A breadth-first search tree has already been computed, starting from a source node s, and a node v has information I of size $O(\log n)$. The goal is to make the information I available to all nodes in the graph. This can be done with a combination of upcast and downcast:

```
1 Upcast: I is sent to s via parent nodes
2 Downcast: I is sent from s via children to all nodes
```

In upcast, node v sends the information I to its parent node, which in turn sends it to its parent node, and so on, until the information reaches the source node s. From there, the downcast begins, with each node forwarding I to each child until all leaves have been reached. An example for the upcast phase is given in Fig. 2.4 and for the downcast phase in Fig. 2.5.

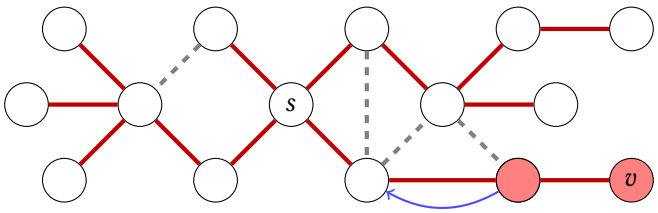


Figure 2.4: In this example, the second round of the upcast phase is executed on the given network.

The number of rounds is $O(\mathrm{Ecc}(s)) = O(D)$, since for both upcast and downcast the maximum distance traveled by a forwarded message is $O(\mathrm{Ecc}(s))$. The upper bound for the number of messages is $O(n)$. In contrast to broadcast, where one to two messages are sent over each edge in the original graph, in upcast and downcast a tree is already present, for which the following property holds:

$$\text{number of nodes} = \text{number of edges} - 1$$

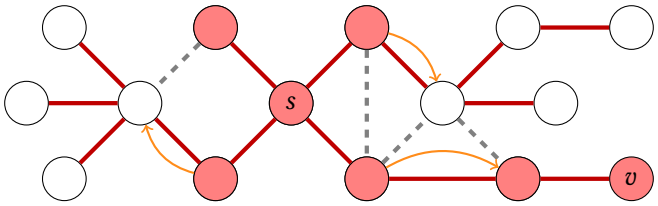


Figure 2.5: In this example, the second round of the downcast phase is executed on the given network.

Therefore, the number of messages is $O(n)$. If several such broadcasts are to be performed sequentially, it is thus more message-efficient to perform upcasts and downcasts in the breadth-first search tree than to repeatedly execute the broadcast algorithm from Section 2.1 each time.

2.5 Pipelining

2.5.1 Multi-Way Downcast

We consider the following initial situation: Node s has k pieces of information $I_1, \dots, I_k$, each of size $O(\log n)$, and a breadth-first search tree, starting from s, has already been computed. The goal is to make the information $I_1, \dots, I_k$ available to all nodes in the graph. The difficulty is that we cannot combine the k pieces of information into a single message, since in the CONGEST model the maximum message size is $O(\log n)$. A naive approach simply executes the regular broadcast algorithm from Section 2.1 k times, which would result in a running time of $O(kD)$. However, this running time can be reduced by the following pipelining algorithm:

---

1 s sends information I_j to all neighbors in round j
2 Whenever some node $v \neq s$ receives information I for the first time: v sends I to all its children

---

In this algorithm, the source node s sends out the information I_j in round j, until $j = k$ holds and all information has been sent. All other nodes immediately forward any information they received for the first time to their children. This makes better use of the available edges and the information I_1 through I_k reach all nodes earlier than with k sequential broadcasts.

An example of the pipelining algorithm is given in Fig. 2.6: After 4 rounds, node v with $\operatorname{dist}(s, v) = 3$ has received (the first) information I_1. (An information is considered received only in the round after receipt.) In the following round, it receives I_2, etc. That is, in round $3 + j$, it receives the last information I_j.

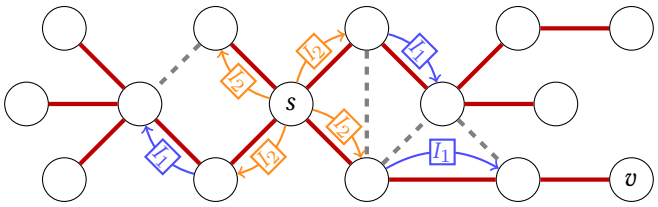


Figure 2.6: In this example, the second round of the pipelining algorithm is executed on the given network.

The inductive claim for the correctness proof is:

For each $1 \leq j \leq k$ and every node $v \neq s$, node v receives at the latest in round $\operatorname{dist}(s, v) + j$ the information I_j.

From this inductive claim it follows that the running time for multi-way downcast via pipelining is $O(\mathrm{Ecc}(s) + k) = O(D + k)$. Roughly speaking, $O(D)$ rounds are needed to reach the farthest node v (from s) and another k rounds until v has also received the last information I_k.

2.5.2 Convergecast

Now consider the situation in which node s wants to ensure that the broadcast or breadth-first search it initiated has actually been completed. We know that this is the case after $O(D)$ rounds, but the actual diameter D is not initially known. Therefore, node s has no simple way of knowing after how many rounds the computation is completed. The solution idea involves sending ACK messages (similar to TCP) from each node to s via upcast. A naive approach would be the following:

```
1 Execute breadth-first search with spanning tree computation from source node s
2 For each leaf: send ACK message to s via upcast in the tree
```

However, the problem here is that high congestion can occur at the parent nodes despite pipelining. In the worst case, the running time is therefore $\Theta(n)$, see Fig. 2.7.

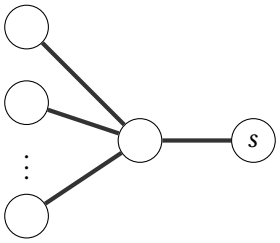


Figure 2.7: Example of high congestion occurring with naive convergecast

A solution to the congestion problem is the aggregation of ACK messages, as described by the following algorithm.

```
1 Execute breadth-first search with spanning tree computation from source node s
2 As soon as a node has received ACK from all its children: send ACK to parent node
3 Leaves in the tree recognize themselves as such when they do not receive a child message
4 s waits until ACK has been received from all children
```

An example of this aggregation algorithm is given in Fig. 2.8:

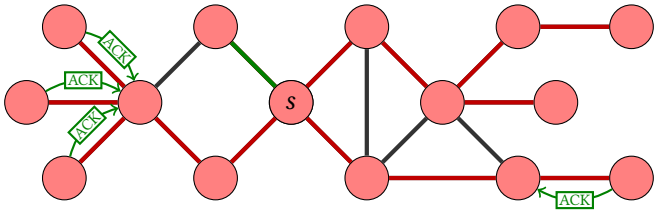


Figure 2.8: In this example, the aggregation algorithm is executed on the given network.

From a functional perspective, the ACK bit of a node is the AND conjunction of the ACK bits of its children. Thus, all ACK messages from the children must arrive first before the node's own ACK message can be formed and sent to the parent node. Due to the tree structure, the bottom-up computation and waiting for the function values of the children is possible. The additional ACK

messages lead to at most a doubling of the time and message complexity. Other aggregate functions for which this technique works include, for example, sum, minimum, and maximum.

2.6 Queuing

2.6.1 Multi-Way Upcast

We consider the following initial situation: A breadth-first search tree has already been computed, starting from a source node s, and several nodes of the network have in total k pieces of information $I_1, \ldots, I_k$, each of size $O(\log n)$. The goal is to make all information $I_1, \ldots, I_k$ available to node s. The information can be distributed throughout the network. In general, it is not the case that one node has all k pieces of information. We perform this type of multi-way upcast with the following algorithm:

```
1 Each node has a queue of unforwarded information
2 In each round, each node sends one piece of information from its queue to its parent node
```

In the algorithm, at the beginning each node's queue consists of the information it already initially knows. In each round, each node sends one of these pieces of information from its queue to its parent node. If the node has sent this information from the queue to the parent node, the information immediately moves into the queue of the parent node. This continues until all queues are empty and s has received all information. An example of the queuing algorithm is given in Fig. 2.9:

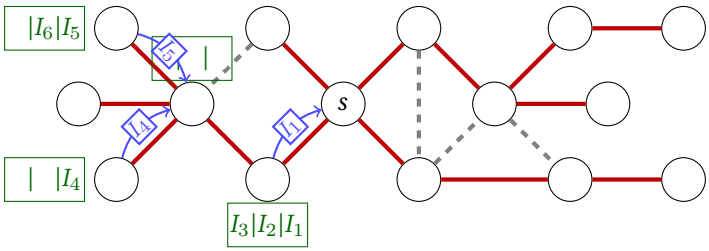


Figure 2.9: In this example, the queuing algorithm is executed on the given network.

2.6.2 Running Time Bound with Auxiliary Lemma

To prove a running time bound for the algorithm, we first need the following auxiliary lemma.

Lemma 2.1. *For each $r \geq 1$ and every node v with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$ the following holds: If v does not receive information in round r, then it does not receive it in round $r+1$ either.*

This lemma states that – from a sufficiently large number of rounds onward – there can no longer be interruptions in the information flow. Thus, it suffices to show a bound for the first (and thus final) interruption of the information flow after this point in time.

Theorem 2.2. *The algorithm for multi-way upcast requires $O(D+k)$ rounds.*

Proof. We consider node s and round $r = \mathrm{Ecc}(s) + 1$. Trivially, the distance from s to itself is 0, i.e., $\mathrm{dist}(s,s) = 0$. Since we chose r as $r = \mathrm{Ecc}(s) + 1$, it holds that $\mathrm{dist}(s,s) = 0 = \mathrm{Ecc}(s) - r + 1$. The inequality $\mathrm{dist}(s,s) \geq \mathrm{Ecc}(s) - r' + 1$ thus holds for all $r' \geq r$.

We next apply Lemma 2.1 to each $r' \geq r$ and obtain the following statement: As soon as s has not received any message for the first time in a round $r' \geq r$, s receives no messages at all. In other words: After round r, no interruptions of the information flow are possible anymore. If s no longer receives any information, then the algorithm terminates.

Overall, there are k pieces of information in the network. Each of the k pieces of information is received by each node (and thus also by node s) at most once. This means that from round r onward, at most k additional rounds can occur in which information is received. Thus, the total number of rounds until termination is at most $\mathrm{Ecc}(s) + k = O(D + k)$. □

Proof of Lemma 2.1. We now prove the auxiliary lemma by induction.

Inductive basis: $r = 1$

Let v be a node for which $\mathrm{dist}(s, v) \geq \mathrm{Ecc}(s) - r + 1 = \mathrm{Ecc}(s)$ holds. Since $r = 1$, it must be the case that $\mathrm{dist}(s, v) \geq \mathrm{Ecc}(s)$. Since $\mathrm{Ecc}(s)$ is the maximum possible distance from s, it also holds that $\mathrm{dist}(s, v) = \mathrm{Ecc}(s)$. The only nodes at distance $\mathrm{Ecc}(s)$ from s are the leaves in the breadth-first search tree of s. Since leaves do not receive any information, the lemma holds for $r = 1$.

Inductive step: $r - 1 \to r$

In the inductive step, $r \geq 2$. We consider a node v for which the inequality $\mathrm{dist}(s, v) \geq \mathrm{Ecc}(s) - r + 1$ holds. The lemma holds trivially if v is a leaf. If v is not a leaf, let u be any child of v in the breadth-first search tree. We assume that v does not receive information in round r. But if v does not receive information, then u did not send any information to v in round $r - 1$. Why did u not send any information to v in round $r - 1$? This can only be the case if u had an empty queue in round $r - 1$ and did not receive any information. Furthermore, we know that since u is a child of v in the breadth-first search tree, $\mathrm{dist}(s, u) = \mathrm{dist}(s, v) + 1$ holds. Now we substitute the above inequality for $\mathrm{dist}(s, v)$ and obtain:

$$\mathrm{dist}(s, u) = \mathrm{dist}(s, v) + 1 \geq \mathrm{Ecc}(s) - r + 1 + 1 = \mathrm{Ecc}(s) - (r - 1) + 1 \ .$$

Now we know that the inductive claim holds for $r - 1$, and we can apply it to node u due to the above inequality $\mathrm{dist}(s, u) \geq \mathrm{Ecc}(s) - (r - 1) + 1$: Since u, as argued above, did not receive information in round $r - 1$, u also does not receive information in round $(r - 1) + 1 = r$. Since the queue of u is also still empty in round r, u does not send any information to v in round r. Since u was an arbitrary child of v in the breadth-first search tree, this statement holds for all children of v. Since none of its children send any information to v in round r, v does not receive any information in round $r + 1$. □

This section is partially based on a lecture by Danupon Nanongkai.

3 Synchronization

The assumption that synchronous rounds exist in distributed systems is idealized, since in real-world applications message transmission does not necessarily occur within constant time. In the CONGEST model, this property is ignored, as all communication proceeds synchronously and without delay. The goal is to simulate synchronous algorithms on an asynchronous network with modest overheads in time and message complexity.

3.1 Asynchronous Model

The greatest difference from the synchronous model is the removal of rounds, i.e., discrete time steps. Instead, there are (as explained in Section 1.2) two distinct types of events:

- The initialization of a node from outside the network.
- The receipt of a message at a node from another node.

Each of these events activates the corresponding node, which can then perform internal computations to process previously received messages and subsequently send a message to each of its neighbors. The duration of message transmission is finite, but may vary from message to message. For runtime analyses, it is generally assumed that each message transmission takes at most one time unit. However, the correctness of an algorithm must hold for arbitrary finite transmission times.

When executing synchronous algorithms in an asynchronous network without additional modifications, inconsistencies can arise if the transmission durations for individual messages differ. An example of the naive execution of the breadth-first search algorithm from Section 2.2 is shown in Fig. 3.1. The gray-colored node is assigned an incorrect distance: The delay in sending the message from the start node to its bottom-right neighbor is large enough that another node with distance message 2 reaches this node earlier. We could now modify the algorithm so that the node sends another distance message to its neighbors upon receiving a smaller distance. However, the existing guarantees for runtime and message complexity would no longer hold.

3.2 Synchronizer Framework

Synchronizers are techniques that enable the simulation of synchronous algorithms in asynchronous networks.

3.2.1 Pulse Generators

In the following, we consider synchronizers based on pulse generator algorithms that locally generate a pulse for each node. However, this pulse is *not* synchronous across different nodes. The pulse can be viewed as analogous to a round in the synchronous model. That is, after the i-th pulse has been generated for a node, it executes round i of the synchronous algorithm to be simulated (receiving, processing, and sending messages). A simulation is correct if pulse $i + 1$ is always generated only after messages from all neighbors for pulse i have been received. The difficulty here is that a node does not necessarily know from which neighbors it must receive messages.

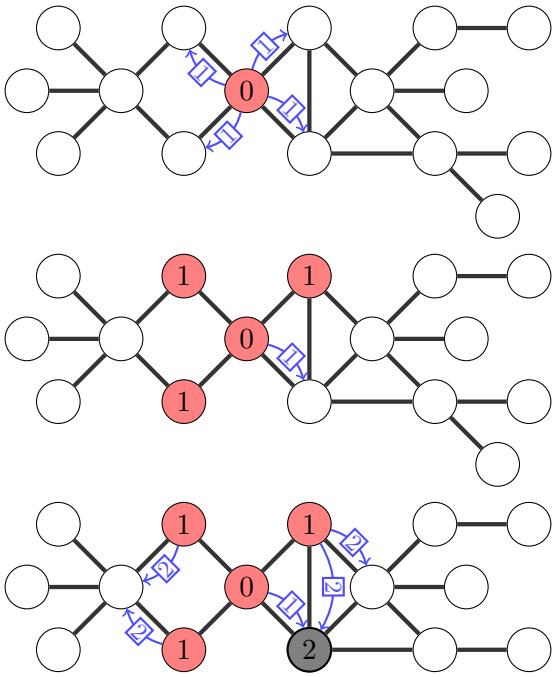


Figure 3.1: Example of naive execution of the breadth-first search algorithm in the asynchronous model

3.2.2 Safety

Safety describes the state of a node for pulse i. Intuitively, a node is considered safe when it has already completed the simulation (receiving, processing, and sending messages) of the current round i.

Definition 3.1. *A node is* safe *for pulse i if all messages it sent during the simulation of round i have reached their respective neighbors.*

Safety can be verified by sending acknowledgment messages after receiving each message. Thus, every receiver of a message sends back an acknowledgment to the sender. This doubles the message and time complexity, since for every message sent, an additional acknowledgment must be sent.

The key observation is that a synchronous algorithm is correctly simulated if for every node, pulse $i + 1$ is always generated only when the node itself and all its neighbors are safe for pulse i. The reason is as follows: If a node itself and all its neighbors are safe for round i, then the node has already received every piece of information needed for round $i + 1$, since it can only receive messages from its neighbors, and those neighbors will not send any more messages because they are already safe. Various strategies can be employed to check safety [**Awerbuch85**].

3.3 Synchronizer α

In synchronizer α, each node decides for itself whether it is safe and can transition to the next pulse. The algorithm works as follows:

- As soon as a node is safe, i.e., it has received all acknowledgment messages for the messages it sent, it sends a safe message to all its neighbors.

- The node immediately recognizes when all neighbors are safe.
- As soon as a node is safe and has received a safe message from all neighbors, it can proceed with the next pulse.

Synchronizer α provides the following guarantees.

Theorem 3.2. *Every synchronous algorithm requiring R rounds and sending M messages can be simulated in asynchronous networks in time $O(R)$ using $O(M + R \cdot m)$ messages.*

The greatest advantage of this algorithm is that it operates purely locally on each node. Sending safe messages introduces only a constant overhead in time complexity. However, the message complexity is significantly increased, since each node must additionally send a safe message to all neighbors in every pulse, which is a major disadvantage of this algorithm.

3.4 Synchronizer β

In synchronizer β, the pulse is generated by a global leader. The algorithm works as follows:

- Setup: A breadth-first search tree has been computed in the network, with the leader of the network located at its root.
- The leader sends each pulse i via downcast to all nodes in the tree.
- Each node executes round i upon receiving pulse i.
- A node is considered safe once it has received all acknowledgment messages.
- The leader learns through aggregated upcast in the tree when all nodes are safe.
- Once all nodes are safe, the leader generates pulse $i + 1$.

Synchronizer β provides the following guarantees.

Theorem 3.3. *Every synchronous algorithm requiring R rounds and sending M messages can be simulated in asynchronous networks in time $O(R \cdot D)$ using $O(M + R \cdot n)$ messages, if a leader and a breadth-first search tree have already been computed.*

In contrast to synchronizer α, synchronizer β has lower message complexity, since exactly two additional messages are sent for each edge in the tree (at most n edges). The major disadvantage, however, is that the runtime increases by a factor of $\Theta(D)$, since both the upcast and the downcast each have a runtime of $\Theta(D)$. Compared to synchronizer α, synchronizer β therefore offers advantages primarily for networks with small diameter. An additional disadvantage of synchronizer β is that a breadth-first search tree and a leader must already have been determined in the asynchronous network.

The required breadth-first search tree can be computed by various algorithms. One possible approach would be to simulate the *synchronous* breadth-first search algorithm using synchronizer α. The synchronous algorithm requires $O(D)$ rounds and has a message complexity of $O(m)$; thus, in an asynchronous network, a breadth-first search tree can be computed in $O(D)$ time units with message complexity $O(mD)$. However, it is also possible to directly develop an algorithm for asynchronous networks that requires $O(D^2)$ time units and has message complexity $O(m + nD)$.

3.5 Synchronizer γ

The goal of the hybrid synchronizer γ is to combine the advantages of synchronizers α and β. This synchronizer works with a special decomposition of the network.

Definition 3.4. *A (ρ, μ)-partition is a decomposition of the nodes V into non-empty, disjoint subsets $P_1, \ldots, P_\ell$ with the following properties:*

1. *Each subset P_i has a designated center, and the nodes in P_i are organized in a tree with distance at most ρ from the center.*
2. *A set F of at most μ edges has been selected such that*
   (a) *every edge in F connects adjacent subsets P_i and P_j, and*
   (b) *for each pair of adjacent subsets, at least one edge in F connects them.*

Ideally, F would contain exactly one edge for each pair of adjacent subsets (i.e., subsets between which an edge exists) that connects these subsets. However, the effort to compute this "minimal" subset is not necessary for our use case. Therefore, we also allow a non-minimal subset F in principle, as long as the maximum size of F is bounded by μ. Fig. 3.2 shows an example of a (ρ, μ)-partition. As is customary in distributed algorithms, it will suffice if nodes only have local information about the (ρ, μ)-partition. In particular, each node must know whether it is a center or not, know its children and possibly its parent in the tree, and know for each of its incident edges whether it is contained in the set F or not.

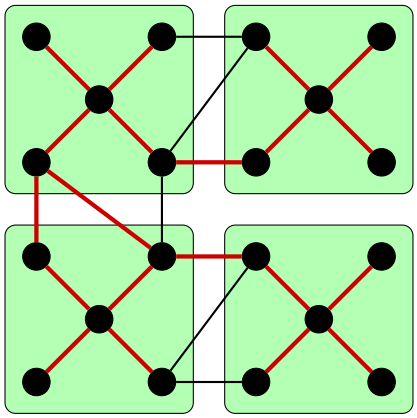


Figure 3.2: Example of a (ρ, μ)-partition with $\rho = 1$ and $\mu = 4$

There are very efficient randomized algorithms for computing a (ρ, μ)-partition.

Theorem 3.5 ([**ElkinN19**, **ForsterGV21**]). *In the synchronous CONGEST model, for any given integer $k \geq 2$, a (ρ, μ)-partition with $\rho = k - 1$ and $\mu = O(n^{1+1/k})$ can be computed in expectation in $O(k)$ rounds.*

This algorithm can also be used in asynchronous networks with synchronizer α (see Theorem 3.2).

Corollary 3.6. *In the asynchronous CONGEST model, for any given integer $k \geq 2$, a (ρ, μ)-partition with $\rho = k - 1$ and $\mu = O(n^{1+1/k})$ can be computed in expectation in time $O(k)$ using $O(km)$ messages.*

Intuitively, synchronizer γ works by applying synchronizer α between subsets and synchronizer β within subsets. Overall, the algorithm works as follows:

- Synchronization within a subset as in synchronizer β:
  - Each center of a subset generates a pulse for all nodes of the subset
  - Downcast of the pulse in the tree from the center to all nodes of the subset
  - Each node executes the current round upon receiving the pulse
  - A node is safe once it has received all acknowledgment messages
  - A subset is safe once all nodes of the subset are safe
  - The center learns through aggregated upcast in the tree when the subset is safe
- Synchronization between subsets as in synchronizer α:
  - Once a subset is safe, adjacent subsets are informed about this:
    - Downcast of the information in the tree from the center to all nodes of the subset
    - Nodes in their own subset inform adjacent subsets via selected edges to neighbors in those subsets
  - The center learns through aggregated upcast in the tree when information has been received via all selected edges to adjacent subsets that the corresponding subsets are safe
  - Once all adjacent subsets are safe, a new pulse is generated

Note: It is not sufficient for all nodes of a subset to be safe in order for the subset to be safe. The nodes may still have neighbors in other subsets that are not yet safe; this was not the case with synchronizer β.

Synchronizer γ provides the following guarantees.

Theorem 3.7. *Every synchronous algorithm requiring R rounds and sending M messages can be simulated in asynchronous networks in time $O(R\rho)$ using $O(M+R(\mu+n))$ messages, if a (ρ,μ)-partition has already been computed.*

The overhead of ρ in the time complexity arises because a constant number of upcasts and downcasts must be performed for the simulation of each round. The runtime for this is proportional to the maximum radius ρ of a tree in the subsets. The overhead of $\mu + n$ in the message complexity arises from two aspects. For each simulated round, a constant number of messages are sent over each selected edge between the subsets, i.e., a total of $O(\mu)$ messages. Additionally, for each simulated round, a constant number of messages are sent over each tree edge during the constant number of upcasts and downcasts; since the trees have no common nodes, they collectively have at most n edges.

Together with the asynchronous algorithm for computing a (ρ,μ)-partition (Corollary 3.6), the following guarantees result for synchronizer γ.

Corollary 3.8. *Every synchronous algorithm requiring R rounds and sending M messages can be simulated in asynchronous networks in time $O(Rk)$ using in expectation $O(M+Rn^{1+1/k}+mk)$ messages.*

In particular, for $k = \lceil \log n \rceil$, this yields a runtime of $O(R \log n)$ and a message complexity of $O(M + Rn + m \log n)$.

3.6 Summary

We have learned about three different synchronizer approaches, each providing different overheads for time and message complexity. They differ conceptually in that they operate more locally, globally, or hybrid. Table 1 summarizes the guarantees of the presented synchronizers.

	Time	**Messages**	**Initialization Overhead**
Synchronizer α	$O(R)$	$O(M + Rm)$	–
Synchronizer β	$O(RD)$	$O(M + Rn)$	Breadth-first search tree
Synchronizer γ	$O(R\rho)$	$O(M + R(\mu + n))$	(ρ, μ)-partition

Table 1: Overview of the presented synchronizers

This section is partially based on a lecture by Christoph Lenzen.

4 Computing a Maximal Independent Set

The problems considered so far (e.g. Broadcast) are *global* problems. Algorithms for solving this type of problems require $\Omega(n)$ rounds, at least when the diameter D is proportional to n. In contrast, *local* problems are those that can be solved in $o(n)$ rounds (independent of the value of D). A prominent example of such a local problem is the computation of a Maximal Independent Set.

4.1 Definitions

Definition 4.1 (Independent Set)**.** *In an undirected graph $G = (V, E)$, an **Independent Set** is a subset of nodes $U \subseteq V$ such that no two nodes are adjacent. We say that U is a **Maximal** Independent Set if no node can be added to U while keeping U an Independent Set. A **Maximum** Independent Set U has the largest possible cardinality among all Independent Sets.*

An example illustrating the distinction between these two definitions is given in Fig. 4.1.

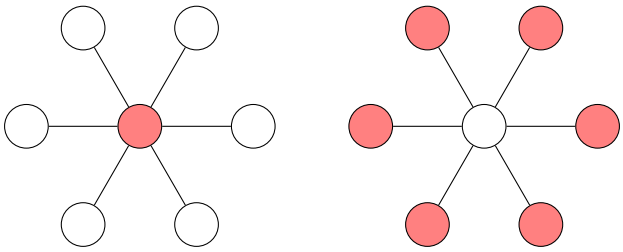


Figure 4.1: The colored nodes form a Maximal Independent Set in both the left and the right network. However, only the right Independent Set is a Maximum Independent Set, since it has the largest possible cardinality.

4.2 Sequential Greedy Algorithm

In the greedy algorithm for computing a Maximal Independent Set, all nodes are considered in arbitrary order. A considered node v is added to the Independent Set only if v has no neighbor in the current Independent Set.

```
1 U ← ∅
2 foreach v ∈ V do
3     if ∀u ∈ U : (u, v) ∉ E then
4         U ← U ∪ {v}
```

Algorithm 1: Sequential Computation of a Maximal Independent Set

The running time in the *RAM model* is $O(m+n)$ with a suitable implementation. In the *CONGEST model*, however, the worst case requires $\Omega(n)$ rounds.

4.3 Distributed Algorithm

The goal of a distributed algorithm is for each node to know whether it is in U or not. Algorithm [**Luby86**, **AlonBI86**, **MetivierRSZ11**] proceeds in phases, and in each phase every node v performs the following three steps:

1. **Choose random number:**
   Node v chooses a uniform real random number $r(v) \in [0, 1)$ and sends it to all neighboring nodes u

2. **Make decision:**
   After node v has received the messages $r(u)$ from all neighboring nodes, v decides to become part of U if $r(v) < r(u)$ for every neighbor u.
   If $v \in U$, all neighboring nodes are notified, and v terminates.

3. **Receive** messages:
   If node v receives a message about a neighbor being added to U, then v terminates.

```
1  while v has not terminated do
2      Choose uniform real random number r(v) ∈ [0, 1)
3      Send r(v) to all neighbors
4      Receive r(u) from each neighbor u
5      if r(v) < r(u) for every neighbor u then
6          Add v to the set U
7          Notify neighbors that v ∈ U
8          Terminate v
9      Receive messages about neighbors added to U
10     if at least one neighbor has been added to U then
11         Terminate v
```

Algorithm 2: Decentralized Computation of a Maximal Independent Set

4.3.1 Correctness

We want to show that this algorithm computes a Maximal Independent Set. For this, we first prove the *Independent Set* property, then *maximality*, and finally the termination of the algorithm.

a) Independent Set We show by induction over the individual phases that the nodes of U form an Independent Set and that the set of terminated nodes consists of U and the neighbors of U. The proof follows directly from the description of the algorithm. If the induction hypothesis holds, then it is true that every node added to U has no neighbors in U from the current phase and no neighbors in U from a previous phase. This means every node we add has no neighbors in U at the time of addition. Moreover, since we notify all its neighbors about the addition, it will not later receive any neighbors from U.

b) Maximality For every terminated node v:

- $v \in U$, or
- there exists $u \in U$ such that u is a neighbor of v

Thus, if all nodes have terminated, this holds for every node v. Therefore, there is no node v that could still be added to U without violating the Independent Set property.

c) Termination Due to the random component, the termination of this algorithm is guaranteed only with *probability* 1.

The probability that two randomly chosen real numbers are equal is 0. By applying the Union Bound, we can argue that if the algorithm requires n phases, the probability that in those n phases all random numbers are distinct is equal to 1. In each phase, at least the node with the globally smallest random number is added to the set U, so n phases suffice. Applying the Union Bound (Lemma B.1) gives us termination with probability 1.

4.3.2 Running Time

Observation: Whenever a node v is added to U, node v and all its neighbors terminate. This leads to a certain reduction in the number of edges in the subgraph of non-terminated nodes in each phase.

For the analysis, we first make the *idealizing assumption* that sending random numbers is possible in $O(1)$ rounds, and thus each phase takes $O(1)$ rounds.

Thus, if we can show that the number of edges is reduced by at least half in each phase (with high probability), then

- $O(\log n)$ phases are needed, and
- it follows from the above assumption a running time of $O(\log n)$.

Implementation detail: For the CONGEST model, it suffices to transmit only $O(\log n)$ bits of the random numbers (see below).

Analysis: Let us first consider the *expected value* of the number of removed edges in a phase. We employ the following analytical techniques:

- Each edge is considered as two directed edges.
- The number of removed edges is systematically underestimated.

Additionally, for the analysis we introduce a dominance relation among the nodes:

Definition 4.2 (Dominance Relation)**.** *u **dominates** neighbor v ($u \twoheadrightarrow v$) if:*

1. *$r(u) < r(u')$ for all neighbors u' of u, and*
2. *$r(u) < r(v)$ for all neighbors $v' \neq u$ of v*

For the probability that "u dominates v" we have:

$$\Pr[u \twoheadrightarrow v] \geq \frac{1}{\deg(u) + \deg(v)}$$

This can be justified as follows: The total number of nodes in the neighborhoods of u and v is at most $\deg(u) + \deg(v)$. This is an upper bound, since the neighborhoods may overlap. We consider all nodes in these two neighborhoods and their associated random numbers $r(\cdot)$. We can now sort these nodes in increasing order by their random numbers. This gives a uniform random permutation π of the nodes that reside in the two neighborhoods.

In a randomly chosen permutation, the probability that a fixed node u is first in this permutation is equal to $1/(\text{number of nodes in both neighborhoods})$, since this event has the same probability for every node.

We can now estimate the number of removed edges (u, v) using the dominance relations. The number of removed edges is at least the number of edges for which $u \twoheadrightarrow v$ holds. For this, we define:

- F: set of undirected edges at the beginning of the current phase
- X: random variable for the number of removed *directed* edges
- $X_{(u \twoheadrightarrow v)}$: number of edges incident to v that are removed due to $(u \twoheadrightarrow v)$

$$X_{(u \twoheadrightarrow v)} = \begin{cases} \deg(v) & \text{if } u \twoheadrightarrow v \\ 0 & \text{otherwise} \end{cases}$$

Note: The second property of the dominance relation ensures that v is removed due to u, and not rather due to a neighbor w of v. Thus, no removed edge is counted twice (since each counted edge is uniquely assigned to a $X_{(u \twoheadrightarrow v)}$).

Using linearity of expectation and the upper bound for $\Pr[u \twoheadrightarrow v]$ from above, we obtain the following estimate:

$$\begin{aligned} \operatorname{Ex}[X] &\geq \sum_{\{u,v\} \in F} \left(\operatorname{Ex}[X_{(u \twoheadrightarrow v)}] + \operatorname{Ex}[X_{(v \twoheadrightarrow u)}] \right) \\ &= \sum_{\{u,v\} \in F} \left(\Pr[u \twoheadrightarrow v] \cdot \deg(v) + \Pr[v \twoheadrightarrow u] \cdot \deg(u) \right) \\ &\geq \sum_{\{u,v\} \in F} \left(\frac{\deg(v)}{\deg(u) + \deg(v)} + \frac{\deg(u)}{\deg(v) + \deg(u)} \right) \\ &= \sum_{\{u,v\} \in F} 1 = |F| \end{aligned}$$

In expectation, the number of *directed* edges removed in a phase is therefore at least $|F|$. Since for each undirected edge there are two directed edges that can only be removed simultaneously, this corresponds to at least $\frac{|F|}{2}$ *undirected* edges being removed.

Lemma 4.3. *In each phase, in expectation, half of the edges in the subgraph induced by the non-terminated nodes are removed.*

The expected value alone only gives a relatively weak statement. We would prefer a stronger statement of the form that with high probability, a significant number of edges are removed in each phase. We obtain this through a combined application of Markov Bound and Chernoff Bound.

Lemma 4.4. *In each phase, with probability at least $\frac{1}{4}$, at least $\frac{1}{3}$ of the edges in the subgraph induced by the non-terminated nodes are removed.*

Proof. We will show that with probability at least $\frac{1}{4}$, at most $\frac{2}{3}$ of the edges are *not* removed from the subgraph induced by the non-terminated nodes. This statement implies the lemma: If at most $\frac{2}{3}$ of the edges are *not* removed, then at least $\frac{1}{3}$ of the edges must be removed.

Let F be the set of edges of the subgraph induced by the non-terminated nodes at the beginning of the current phase. Let Y be the random variable indicating the number of edges that are *not* removed from the subgraph induced by the non-terminated nodes in this phase. From Lemma 4.3, we know that $\mathrm{Ex}[Y] \leq \frac{|F|}{2}$. By applying the Markov inequality (see Theorem B.12) with $\alpha = \frac{4}{3}$, it follows that:

$$\Pr\left[Y \geq \frac{2}{3} \cdot |F|\right] \leq \Pr\left[Y \geq \frac{4}{3} \cdot \mathrm{Ex}[Y]\right] \leq \frac{3}{4}\,.$$

For the complementary event, we have:

$$\Pr\left[Y < \frac{2}{3} \cdot |F|\right] = 1 - \Pr\left[Y \geq \frac{2}{3} \cdot |F|\right] \geq 1 - \frac{3}{4} = \frac{1}{4}\,.$$

As argued above, this bound implies the lemma. □

Lemma 4.5. *With high probability, Algorithm 10 terminates after* $72\lceil c \log n \rceil$ *phases.*

Proof. The idea here is to divide phases into "good" and "bad" phases. For this, we introduce a binary random variable for each phase:

$$Z_i = \begin{cases} 1 & \text{if at least } \frac{1}{3} \text{ of the edges are removed in phase } i \\ 0 & \text{otherwise} \end{cases}$$

Phase i is considered good if $Z_i = 1$, and bad if $Z_i = 0$. Thus, $Z = \sum_{i=1}^{72\lceil c \log n \rceil} Z_i$ gives the number of good phases.

From linearity of expectation, we therefore have

$$\mathrm{Ex}[Z] = \tfrac{1}{4} \cdot 72\lceil c \ln n \rceil \geq 18c \ln n\,.$$

In general, the number of (originally $m \leq n^2$) edges is reduced after k good phases to at most $(\frac{2}{3})^k m \leq (\frac{2}{3})^k n^2$. As soon as the subgraph induced by the non-terminated nodes has no more edges, the algorithm terminates. This is the case after $k' = \log_{3/2}(n^2) + 1$ good phases, since $(\frac{2}{3})^{k'} n^2 < 1$.

The probability of the complement, namely that fewer than k' good phases occur, can now be estimated using a Chernoff Bound (see Theorem B.14). We set $p = \frac{1}{4}$, $\mu = p \cdot 72\lceil c \log n \rceil = 18\lceil c \log n \rceil$, and $\delta = \frac{1}{3}$. In Lemma 4.4, we have already shown that $\Pr[Z_i = 1] \geq \frac{1}{4} = p$ holds. Using the chain of inequalities

$$k' - 1 = \log_{3/2}(n^2) \leq \tfrac{2 \ln n}{\ln(3/2)} \leq 6 \ln n \leq (1 - \tfrac{1}{3}) 18c \ln n = (1 - \tfrac{1}{3})\mu$$

we can now apply the Chernoff Bound as follows:

$$\Pr[Z < k'] = \Pr[Z \leq k' - 1] \leq \Pr\left[Z \leq (1 - \tfrac{1}{3})\mu\right] \leq \frac{1}{e^{\frac{\delta^2}{2} \cdot \mu}} = \frac{1}{e^{\mu/18}} \leq \frac{1}{e^{c \ln n}} = \frac{1}{n^c}\,.$$

Thus, $\Pr[Z \geq k'] \geq 1 - \frac{1}{n^c}$. Therefore, the algorithm terminates with probability at least $1 - \frac{1}{n^c}$ within $72\lceil c \log n \rceil$ phases. □

4.3.3 Bandwidth Constraints

So far, we have assumed that we can generate arbitrary real random numbers from the interval $[0, 1)$ and send them in one round. In the CONGEST model, however, the message size is limited to $O(\log n)$ bits. We must therefore show that with this bandwidth constraint of $O(\log n)$, it is possible for each edge $\{u, v\}$ to decide in a constant number of rounds whether $r(u) < r(v)$ or $r(u) > r(v)$.

The idea behind the adaptation for the CONGEST model is to represent the real random numbers as infinite strings of random bits. To compare two such numbers, it would suffice for the two nodes to exchange bits only up to the first position where they differ. We want to show that with high probability, it is sufficient to exchange only the first $(c+3)\lceil \log n \rceil = O(\log n)$ bits.

The probability that the first $(c+3)\lceil \log n \rceil$ bits are identical is

$$\left(\frac{1}{2}\right)^{(c+3)\lceil \log n \rceil} \leq \left(\frac{1}{2}\right)^{(c+3)\log n} = \frac{1}{n^{c+3}}.$$

The base $\frac{1}{2}$ arises because for each pair of bits from the four possible combinations $\{00, 01, 10, 11\}$, there are two combinations in which the bits are equal. The probability that for any of at most n^2 pairs in at most n rounds of the algorithm, the first $(c+3)\lceil \log n \rceil$ bits are equal is, by the Union Bound (see Lemma B.1), at most $\leq n^3 \cdot \frac{1}{n^{c+3}} = \frac{1}{n^c}$. Thus, a bandwidth of $O(\log n)$ suffices with high probability.

4.3.4 Excursus: Robustness of High Probability

We recall that the term "high probability" is defined as follows.

Definition 4.6. *An event occurs* with high probability *if it occurs with probability at least* $1 - \frac{1}{n^c}$ *for an arbitrary given constant* $c \geq 1$.

In algorithms that are correct with high probability, the constant hidden in the O-notation typically depends on c and affects, for example, the running time or message complexity.

Suppose now that there are n events $A_1, \ldots A_n$, each occurring with high probability. How likely is it that all of them occur simultaneously, i.e., what is the probability of the event $\bigcap_{i=1}^{n} A_i$? We can argue as follows that with "almost" high probability, all these events occur simultaneously:

$$\Pr\left[\bigcap_{i=1}^{n} A_i\right] = 1 - \Pr\left[\bigcup_{i=1}^{n} \bar{A}_i\right] \geq 1 - \sum_{i=1}^{n} \Pr[\bar{A}_i] = 1 - \sum_{i=1}^{n} (1 - \Pr[A_i])$$
$$\geq 1 - \sum_{i=1}^{n} \frac{1}{n^c} = 1 - \frac{n}{n^c} = 1 - \frac{1}{n^{c-1}}$$

If the probability for each A_i could be increased to $1 - \frac{1}{n^{c+1}}$, then the event $\bigcap_{i=1}^{n} A_i$ would also occur with high probability (for the given constant c). In algorithmic contexts, such an increase in probability can often be achieved by running the algorithm with $c+1$ instead of c, which typically changes the time and/or message complexity only by constants. This section is partially based on lectures by Christoph Lenzen and Roger Wattenhofer.

5 Graph Spanners

5.1 Problem Setting

A fundamental challenge in the storage and processing of graphs is their size. A graph with n vertices can potentially have $\Omega(n^2)$ edges. Numerous methods have therefore been developed to compress graphs by other graphs that have the same vertices as the original but significantly fewer edges, ideally only $O(n)$. Typically, such compressed graphs only approximately satisfy the same properties as the originals, and there is often a trade-off between the size of the compressed graph and the quality of the approximation of the original.

In this section, we deal with compression methods that approximately preserve distances. The central definition for this is that of a spanner.

Definition 5.1. *A* t-spanner *(spanner with* stretch t*) of a graph* $G = (V, E)$ *is a subgraph* $H = (V, F)$ *such that*

$$\operatorname{dist}_H(u, v) \leq t \cdot \operatorname{dist}_G(u, v)$$

for all pairs of vertices $u, v \in V$*.*

The spanner H is a subgraph of G in the sense that the vertex set V of both graphs is the same and the edge set F of H is a subset of the edge set E of G (i.e., $F \subseteq E$).

We first discuss two basic properties of this definition. First, the subgraph H never overestimates the distances in G, since all paths available in H are also available in G, in particular every shortest path in H.

Lemma 5.2. *For every spanner* H *of* G*:* $\operatorname{dist}_H(u, v) \geq \operatorname{dist}_G(u, v)$ *for every pair of vertices* $u, v \in V$*.*

Furthermore, spanners can be characterized by the fact that the inequality from Definition 5.1 holds for all shared endpoints of edges.

Lemma 5.3. *A subgraph* $H = (V, F)$ *is a* t*-spanner of* $G = (V, E)$ *if and only if*

$$\operatorname{dist}_H(u, v) \leq t \cdot w_G(u, v)$$

for every edge $(u, v) \in E$*.*

The equivalence of the two formulations (all pairs of vertices vs. shared endpoints of edges) follows almost directly from considering all edges of a shortest path.

In the following, we treat spanners of undirected, unweighted graphs (in which every edge has weight 1).

5.2 Greedy Spanner

5.2.1 3-Spanner

We first analyze the following (sequential) greedy algorithm [**AlthoferDDJS93**] for computing a 3-spanner of a graph $G = (V, E)$.

```
1 F ← ∅
2 foreach (u, v) ∈ E do
3 |   Let H = (V, F)
4 |   if dist_H(u, v) > 3 then
5 |   |   F ← F ∪ {(u, v)}
```

The algorithm iterates over the edges of G and checks for each edge whether a path of length at most 3 between the endpoints of the edge already exists in the spanner computed so far. If not, the edge is added to the spanner. The algorithm thus follows a *greedy* scheme, because a decision is made individually for each edge whether it is necessary or not. The algorithm computes *by design* a spanner with stretch 3, which we now prove formally.

Lemma 5.4. *The spanner $H = (V, F)$ computed by the greedy algorithm is a 3-spanner of G.*

Proof. Let $(u, v) \in E$ be an arbitrary edge of G. If (u, v) was added to the edge set F of H, then obviously $\mathrm{dist}_H(u, v) = 1 \leq 3$. Otherwise ($(u, v) \notin F$), consider the iteration in which $\mathrm{dist}_H(u, v)$ is checked (and the decision is made not to add (u, v) to F), and let $H' = (V, F')$ be the state of H during this iteration. By the algorithm, $\mathrm{dist}_{H'}(u, v) \leq 3$. Since H' is a subgraph of H, we also have $\mathrm{dist}_H(u, v) \leq \mathrm{dist}_{H'}(u, v)$, and thus $\mathrm{dist}_H(u, v) \leq 3$. Since (u, v) was an arbitrary edge, by Lemma 5.3, H is a 3-spanner of G. □

The proof that the computed spanner H has only $O(n^{3/2})$ edges is somewhat more involved. The concept of the girth of a graph is useful here.

Definition 5.5. *The* girth *of a graph is the length of its shortest cycle.*

Lemma 5.6. *The girth of $H = (V, F)$ is greater than 4.*

Proof. Suppose H has a cycle K of length at most 4 (i.e., with at most 4 edges). Let (u, v) be the edge of cycle K that was added last to F by the algorithm. The edges in $K \setminus \{(u, v)\}$ form a path from u to v, which already exists in H in the iteration in which (u, v) is added to F. Before adding (u, v) to F, there is no path of length at most 3 from u to v. Therefore, the path $K \setminus \{(u, v)\}$ has length at least 4, and the cycle K thus has length at least 5. This contradicts the assumption that K has length at most 4. □

Lemma 5.7. *Every undirected graph with n vertices and girth greater than 4 has $O(n^{3/2})$ edges.*

For the proof of this lemma, we will need the following additional lemma, which we prove immediately afterwards.

Lemma 5.8. *Every undirected graph with n vertices and minimum degree $\geq n^{1/2} + 1$ has girth at most 4.*

Proof of Lemma 5.7. Let G be a graph with girth greater than 4 and at least $3n^{3/2}$ edges. Repeatedly remove vertices with degree less than $n^{1/2}+1$ (with all incident edges) from G, until every vertex has degree at least $n^{1/2}+1$. Let $G' = (V', E')$ be the resulting graph. In total, at most $n \cdot (n^{1/2}+1) \leq 2n^{3/2}$ edges are removed, so

$$|E'| \geq |E| - 2n^{3/2} \geq 3n^{3/2} - 2n^{3/2} = n^{3/2} > 0\,.$$

In particular, G' is therefore not empty and $|V| \geq 1$. Since $|E'| \leq |V'|^2$, we also have

$$|V'| \geq \frac{|E'|}{|V'|} \geq \frac{|E'|}{n} \geq n^{3/2}/n = n^{1/2}\,.$$

By construction, G' now has a minimum degree of at least $n^{1/2} + 1 \geq |V'|^{1/2} + 1$.

We can now apply Lemma 5.8 to G': G' has girth at least 4. Since G' is a subgraph of G, every cycle in G' also exists in G. It follows that G has girth at least 4, which contradicts our initial assumption. Thus, every graph with girth > 4 has fewer than $3n^{3/2} = O(n^{3/2})$ edges. □

Proof of Lemma 5.8. Suppose that G has girth at least 5. We consider an arbitrary vertex v and the breadth-first search tree of depth 2 starting from v.

Every edge of a vertex at distance at most 1 from v to a vertex at distance at most 2 from v must be an edge of that vertex to its parent or to a child in the breadth-first search tree, since otherwise G would contain a cycle of length at most 4 (see Fig. 5.1). Every vertex at distance less than 2 from v (i.e., the internal nodes in the breadth-first search tree of depth 2 rooted at v) therefore has at least $\sqrt{n}$ children in the breadth-first search tree, since at most one of its incident edges can go to a parent.

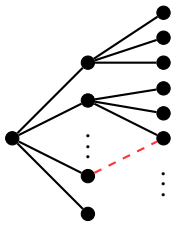


Figure 5.1: Breadth-first search tree of depth 2 from an arbitrary root vertex v. The red dashed edge cannot exist, since it would close a cycle of length 4.

The number of vertices at distance 2 from v is therefore at least $n^{1/2} \cdot n^{1/2} = (n^{1/2})^2 = n$. Together with v, we count at least $n + 1$ vertices for G. Due to this contradiction, our assumption was false, and we conclude that G has girth at most 4. □

With the greedy algorithm, we have thus shown the following statement.

Theorem 5.9. *Every graph G with n vertices has a 3-spanner with $O(n^{3/2})$ edges.*

5.2.2 Generalization

The statement of Theorem 5.9 can be generalized to the following trade-off between stretch and size of the spanner.

Theorem 5.10. *For every integer $k \geq 2$, every graph G with n vertices has a $(2k-1)$-spanner with $O(n^{1+1/k})$ edges.*

For $k = 2$, we obtain, as already shown by us, a 3-spanner of size $O(n^{3/2})$. For $k = \lceil \log n \rceil$, we obtain

$$n^{1/k} \leq n^{1/\log n} = \left(2^{\log n}\right)^{1/\log n} = 2^{(1/\log n)\cdot \log n} = 2$$

and thus $O(n^{1+1/k}) = O(n)$. This corresponds, up to constant factors, to the minimally necessary number of $\Omega(n)$ edges for connected graphs.

A $(2k-1)$-spanner with $O(n^{1+1/k})$ edges can also be computed with the greedy algorithm; we only need to generalize the tolerated path length in the spanner from 3 to $2k-1$.

```
1 F ← ∅
2 foreach (u, v) ∈ E do
3     Let H = (V, F)
4     if dist_H(u, v) > 2k − 1 then
5         F ← F ∪ {(u, v)}
```

We now sketch how the previous arguments must be adjusted to prove Theorem 5.10.

To prove that the stretch is $2k-1$, the same proof as in Lemma 5.4 can be used; the number 3 simply needs to be replaced by $2k-1$. For the proof that H has the desired size, we proceed similarly.

Lemma 5.11. *The girth of $H=(V,F)$ is greater than $2k$.*

Proof. Suppose H has a cycle K of length at most $2k$ (i.e., with at most 4 edges). Let (u,v) be the edge of cycle K that was added last to F by the algorithm. The edges in $K \setminus \{(u,v)\}$ form a path from u to v, which already exists in H in the iteration in which (u,v) is added to F. Before adding (u,v) to F, there is no path of length at most $2k-1$ from u to v. Therefore, the path $K \setminus \{(u,v)\}$ has length at least $2k$, and the cycle K thus has length at least $2k+1$. This contradicts the assumption that K has length at most $2k$. □

Lemma 5.12. *Every undirected graph with n vertices and girth greater than $2k$ has $O(n^{1+1/k})$ edges.*

To prove this lemma, the proof of Lemma 5.7 also only needs to be carried out with different numbers. The proof of the auxiliary lemma also only needs to be slightly adapted.

Lemma 5.13. *Every undirected graph with n vertices and minimum degree $\geq n^{1/k}+1$ has girth at most $2k$.*

Proof. Suppose that G has girth at least $2k+1$. We consider an arbitrary vertex v and the breadth-first search tree of depth k starting from v.

Every edge of a vertex at distance at most $k-1$ from v to a vertex at distance at most k from v must be an edge of that vertex to its parent or to a child in the breadth-first search tree, since otherwise G would contain a cycle of length at most $2k$. Every vertex at distance less than k from v (i.e., the internal nodes in the breadth-first search tree of depth k rooted at v) therefore has at least $n^{1/k}$ children in the breadth-first search tree, since at most one of its incident edges can go to a parent.

The number of vertices at distance 2 from v is therefore at least $(n^{1/k})^k = n$. Together with v, we count at least $n+1$ vertices for G. Due to this contradiction, our assumption was false, and we conclude that G has girth at most $2k$. □

5.2.3 Optimality Conjecture

The trade-off between stretch and size of the spanner computed by the greedy algorithm goes back to graph-theoretic considerations between the size of graphs and their girth. The *girth conjecture* of Paul Erdős [**Erdos63**] states that for every $k \geq 2$ and every sufficiently large n, there exists a graph with n vertices and $\Omega(n^{1+1/k})$ edges that contains no cycle of length at most $2k$. This conjecture has only been proven for small values of k, and it is an open problem in combinatorics whether the generalization holds for arbitrary k. If the girth conjecture is true, then the $2k-1/O(n^{1+1/k})$ trade-off is optimal – up to constants in the size.

Lemma 5.14. *If the girth conjecture holds, then for all sufficiently large n, there exists a graph G with n vertices such that every $(2k-1)$-spanner of G has at least $\Omega(n^{1+1/k})$ edges.*

Proof. Let G be a graph as in the girth conjecture, i.e., G has $\Omega(n^{1+1/k})$ edges and girth greater than $2k$. Suppose there exists a non-trivial $(2k-1)$-spanner H of G. Non-trivial means there is at least

one edge (u, v) in G that does not appear in H. Since H is a $(2k-1)$-spanner of G, there exists a path P of length at most $2k-1$ from u to v in H. But now $P \cup \{(u, v)\}$ is a cycle of length at most $2k$ in G, which contradicts the assumption that the girth of G is greater than $2k$. Thus, G is the only $(2k-1)$-spanner of G. It is therefore correct to say that every $(2k-1)$-spanner of G has as many edges as G, namely $\Omega(n^{1+1/k})$. □

Finding a better trade-off between stretch and asymptotic size of spanners would amount to disproving the girth conjecture, which by itself would be a breakthrough in combinatorics.

5.3 Spanner Construction via Clustering

As we have seen, the simple greedy algorithm already computes a useful spanner. However, this greedy algorithm is highly sequential, since the edges of the graph cannot be checked independently of each other. For parallel and distributed models of computation (such as the CONGEST model), other algorithms have therefore been developed that compute spanners with the presumably optimal trade-off or come as close as possible to this trade-off.

We now discuss in more detail how spanners can be computed efficiently in the CONGEST model. A well-known algorithm by Baswana and Sen [**BaswanaS07**] has the following guarantees.

Theorem 5.15 ([**BaswanaS07**])**.** *For every $k \geq 2$, a $(2k-1)$-spanner of an unweighted graph with $O(n^{1+1/k} + kn)$ edges can be computed in expectation in $O(k^2)$ rounds in the CONGEST model.*

This algorithm can be extended to weighted graphs. In the following, we present the algorithm for the special case $k = 2$; a 3-spanner with $O(n^{3/2})$ edges will thus be computed. The goal is that at the end of the computation, every vertex knows which of its incident edges belong to the spanner.

The algorithm of Baswana and Sen is based on the computation of clusters, which can be formally defined as follows.

Definition 5.16. *A cluster is a set of connected vertices. A clustering is a set of non-overlapping clusters.*

The algorithm for computing the 3-spanner essentially consists of three parts. First, a subset Z of vertices is randomly chosen as the centers. Then, star clusters are formed around the centers as their centers, with the respective edges to the center of the star being added to the spanner. Finally, additional edges are added to the spanner: clustered vertices add *one* of the edges to the adjacent cluster to the spanner, and unclustered vertices add the edges to all their neighbors to the spanner. We now provide the corresponding pseudocode.

$H \leftarrow (V, \emptyset)$
$Z \leftarrow \emptyset$
foreach *vertex* $v \in V$ **do**
    Add v to Z with probability $p = \frac{1}{\sqrt{n}}$ and create a cluster for v
foreach *vertex* $v \in V \setminus Z$ **do**
    if v *has (at least) one neighbor from* Z **then**
        Add v to the cluster of a neighbor from Z
        Add edge to this neighbor to H
foreach *vertex* $v \in V$ **do**
    if v *is part of a cluster* **then**
        Add one edge to H for each cluster adjacent to v
    else
        Add edges to all neighbors of v to H

An example of a possible result of the algorithm is given in Fig. 5.2.

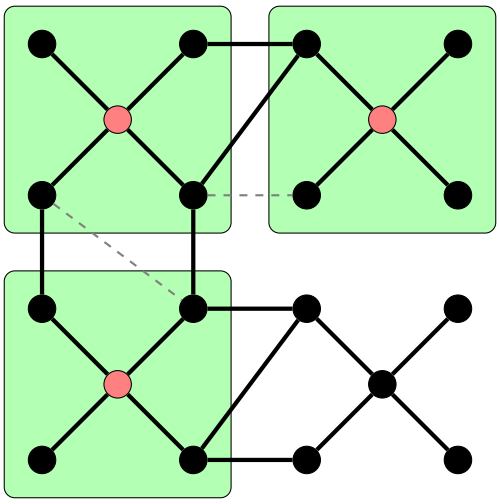


Figure 5.2: Example of a result of the algorithm. Edges not in the spanner are shown as gray dashed lines. The randomly chosen centers are marked in red and the clusters with the centers as their centers are marked in green.

In the algorithm, clusters can be distinguished by the IDs of their centers. These IDs can be communicated to all vertices in the cluster as well as to all adjacent vertices of a cluster in constant time. Therefore, the algorithm can obviously be implemented in a constant number of rounds in the CONGEST model. We now analyze the stretch and size of the computed spanner.

Lemma 5.17. *The spanner H computed by the algorithm has stretch* 3.

Proof. Let (u, v) be an arbitrary edge from G. If either of the endpoints u or v is not clustered, H contains the edge (u, v). If both u and v are clustered, we distinguish two cases: either u and v are in the same cluster or in different clusters. If u and v are in the same cluster, there is a path of length 2 from u to v through the center of the cluster. If u and v are in different clusters, then the cluster of v is adjacent to u, since there is at least one edge (namely (u, v)) from u into the cluster of

v. Therefore, H contains some edge (u, w) such that w is a vertex in the cluster of v. Thus, there is a path of length at most 3 from u to v through w and the center of the shared cluster of w and v. □

Lemma 5.18. *The spanner H computed by the algorithm has $O(n^{3/2})$ edges in expectation.*

Proof. There are essentially three types of edges in H:

1. Edges to the cluster center for clustered vertices
2. Edges to adjacent clusters for clustered vertices
3. Edges to neighbors for unclustered vertices

We now show that there are $O(n^{3/2})$ edges in expectation of each of these three types, from which it follows (due to the linearity of expectation) that H has $O(n^{3/2})$ edges.

Every clustered vertex has one edge of type 1, and since there are at most n clustered vertices, there are at most $O(n) = O(n^{3/2})$ edges of type 1. Every clustered vertex has at most as many edges of type 2 as there are clusters. The number of clusters is $|Z|$. Since the number of centers is binomially distributed with parameters p and n, we have $\mathrm{Ex}[|Z|] = pn = \sqrt{n}$. Since there are at most n clustered vertices, there are in expectation at most $O(n^{3/2})$ edges of type 2. Every unclustered vertex has as many edges of type 3 as it has neighbors. If one iterates over the neighbors of a vertex in any order, the time of occurrence of the first center is geometrically distributed with parameter p; thus, in expectation, the first center among the neighbors appears after $\frac{1}{p} = \sqrt{n}$ vertices. Since unclustered vertices do not have centers as neighbors, they have at most $O(\sqrt{n})$ neighbors in expectation. Since there are at most n unclustered vertices, there are in expectation at most $O(n^{3/2})$ edges of type 3. □

Finally, it is worth noting that the result of the algorithm can also be interpreted as a $(1, n^{3/2})$-partition in the sense of Definition 3.4 for use in the synchronizer γ. This section is partially based on a lecture by Virginia Vassilevska Williams.

6 Computing Shortest Paths

6.1 Problem Statement

There are various algorithms for computing shortest paths in distributed models such as the CONGEST model. We define two problems:

Single-Source Shortest Paths (SSSP): A source node s is given, and the goal is to compute the distance from this node to every node v. Each node initially knows whether it is the source node or not. In the end, each node v should know the distance $\mathrm{dist}(s, v)$ from the source node s to v. One could additionally require that a shortest-paths tree must be computed. In a distributed model, this would mean that each node must know the predecessor node on the shortest path from s; this corresponds to the parent node in the tree.

All-Pairs Shortest Paths (APSP): The goal is to compute the distance $\mathrm{dist}(u, v)$ for all pairs of nodes u and v. In the end, each node v knows the distance $\mathrm{dist}(u, v)$ for every other node u. Thus, each node must compute $n - 1$ distinct distance values.

6.2 Unweighted Graphs

6.2.1 Review: Breadth-First Search

In "unweighted" graphs, all edges have the same weight, and we normalize this weight to 1. We can solve the SSSP problem in such networks using breadth-first search: The source node s has the initial distance 0 to itself ($\mathrm{dist}(s, s) = 0$), and each node that first receives distance information sends a message with a distance incremented by 1 to all its neighbors.

An example of the breadth-first search algorithm, which we have already learned, can be found in Fig. 6.1: Initially, the source node has distance 0 to itself and sends the distance information 1 to all its neighbors in the first round. The source node becomes the parent of its neighbors in the BFS tree, and the corresponding tree edge is the edge over which the distance information was received. The neighbors adopt this distance value and in the second round send the distance information 2 to all their neighbors, except their parent node in the tree. This propagation of distance values repeats until all nodes have been reached.

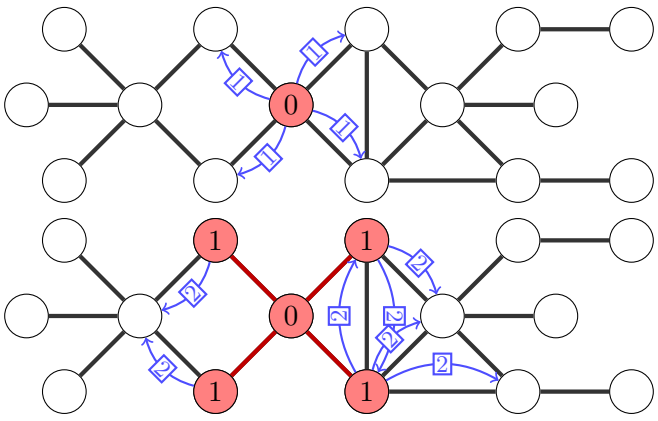


Figure 6.1: Example of the first round (top) and second round (bottom) of breadth-first search in the given network

From the guarantees we have already learned about the breadth-first search algorithm, the following theorem for the SSSP problem follows.

Theorem 6.1. *In the CONGEST model, the SSSP problem for unweighted graphs can be solved in $O(D)$ rounds.*

6.2.2 All-Pairs Shortest Paths

Idea: We run breadth-first search for each node in unweighted graphs. This gives each node the distance to every other node.

Naive Implementation: If a node participates in multiple breadth-first searches, it is possible that this node simultaneously receives distance information from multiple BFS runs and would have to forward all this information in the same round. However, because of the limited message size in the CONGEST model, this cannot simply be implemented by packing all this information into one message. The naive approach would be to execute the breadth-first searches sequentially, which would require $O(nD)$ rounds. However, in the case of breadth-first search, there is indeed a more efficient solution: parallelization via randomization.

6.2.3 Parallelization via Random Delay

We use the Random Delay technique [**LeightonMR94**, **Nanongkai14**] to run multiple instances of breadth-first search in parallel. The goal is to run breadth-first search for $k = |U|$ different source nodes.

Idea: We exploit the fact that in the breadth-first search algorithm, each node sends distance information to its neighbors at most once.

Random Delay Algorithm:

1 Each source node s_i uniformly randomly chooses $\delta_i \in \{0, \dots, k-1\}$
2 Node s_i starts instance of the algorithm with delay δ_i

We will see that this type of randomization ensures that the distance information to be sent for each node is distributed reasonably well so that only few distance information from different BFS runs need to be sent simultaneously. We will therefore prove the following lemma.

Lemma 6.2. *The Random Delay Algorithm computes a BFS tree for each node in U in $O(k+D)$ rounds and requires message size $O(\log^2 n)$ with high probability.*

It is clear that with a delay of at most $k-1$, the last breadth-first search is completed after $O(k+D)$ rounds. Our analysis will therefore focus on bounding the message size. If message size $O(\log^2 n)$ were available, then the Random Delay Algorithm would require $O(k+D)$ rounds. Since we only have message size $O(\log n)$ available in the CONGEST model, we simulate the algorithm from Lemma 6.2 as follows: Each round of the original algorithm from the lemma is divided into $\log n$ simulated rounds, in which $O(\log n)$ bits of the message to be transmitted are sent each time. This implies that (with this simulation) the Random Delay Algorithm requires $O((k+D)\log n)$ rounds in the CONGEST model.

Theorem 6.3. *In the CONGEST model, there is a Monte-Carlo algorithm that solves the APSP problem for unweighted graphs in $O(n \log n)$ rounds with high probability.*

The theorem follows from the above discussion, since $D \leq n-1$ and $k = n-1$ holds. This is a Monte-Carlo algorithm that is correct when the message size of $O(\log n)$ can be maintained. If this is not the case, it is unclear what to do, and we allow the algorithm to abort, which results in an incorrect or incomplete result.

6.2.4 Analysis

We now wish to prove the following lemma.

Lemma 6.4. *In each round of the Random Delay Algorithm, with high probability, each node sends messages from only $O(\log n)$ instances simultaneously.*

We use the following strategy to prove this lemma: We show that with high probability, it never occurs that for any node, messages need to be sent in more than $O(\log n)$ instances of breadth-first search simultaneously. This implies the desired bandwidth bound for Lemma 6.2, because each message from the $O(\log n)$ instances has size $O(\log n)$. Thus, an overall message size of $O(\log^2 n)$ is sufficient.

Proof of Lemma 6.4. We know that in each instance of the breadth-first search algorithm, each node sends a message only once. For each node v and each instance i, we denote by $M_{v,i}$ that unique message that v must send in the i-th instance of breadth-first search at some point in time. Furthermore, we denote by $r_{v,i}$ that round in which this message $M_{v,i}$ would be sent if the i-th instance of breadth-first search were executed completely without delays (and without the other instances).

First, we fix a round r, a node v, and an instance i. The probability that message $M_{v,i}$ is sent by node v in round r is at most $\frac{1}{k}$, i.e.:

$$\Pr[M_{v,i} \text{ sent by } v \text{ in round } r] \leq \frac{1}{k}$$

Why is this the case? We know that due to the delay δ_i of the i-th instance, the message $M_{v,i}$ is sent in round $\delta_i + r_{v,i}$. If we now assume that the message is sent in round r, then $r = \delta_i + r_{v,i}$ must hold, or equivalently: $\delta_i = r - r_{v,i}$. For this equation to be satisfied, $r - r_{v,i}$ must take a value between 0 and $k-1$, and δ_i must correspond to this value. There are k different possibilities for choosing δ_i, all equally likely, but at most one choice satisfies the requirement $\delta_i = r - r_{v,i}$. Therefore, the probability of satisfying this requirement is at most $\frac{1}{k}$.

Now we fix a round r, a node v, and a set of messages $\mathcal{M} \subseteq \bigcup_{i=1}^{k} \{M_{v,i}\}$. The set $\mathcal{M}$ is a subset of the individual messages that v sends in the k instances. The probability of a message $M_{v,i}$ being sent by v in round r depends only on the choice of the delay δ_i. Since the delays were chosen independently, the number of messages from $\mathcal{M}$ that are sent by v in round r is binomially distributed – the process can therefore be modeled by a Bernoulli experiment. As analyzed above, the probability that a single message from $\mathcal{M}$ is sent exactly in round r by node v is at most $\frac{1}{k}$. For the probability that all messages from $\mathcal{M}$ are sent exactly in round r by node v, we thus have:

$$\Pr[\text{"all messages from } \mathcal{M} \text{ sent by } v \text{ in round } r\text{"}] \leq \left(\frac{1}{k}\right)^{|\mathcal{M}|} \tag{6.1}$$

We now wish to argue that for a set of messages $\mathcal{M}$ of size $|\mathcal{M}| = \Omega(\log n)$, it is very unlikely that all messages from $\mathcal{M}$ are to be sent simultaneously in some round r by some node v. In general,

there are at most $\binom{k}{\ell}$ message sets $\mathcal{M} \subseteq \bigcup_{i=1}^{k}\{M_{v,i}\}$ of size $|\mathcal{M}| = \ell$. This is because there are essentially $\binom{k}{\ell}$ different ways to choose a subset of size ℓ from a set of size k.

Now let $\ell' = \max\{(c+4)\log n, 2e\}$, where e is Euler's number, and let $S_{r,v}$ be the following event: there exists a subset $\mathcal{M} \subseteq \bigcup_{i=1}^{k}\{M_{v,i}\}$ of size $|\mathcal{M}| \geq \ell'$ such that in the Random Delay Algorithm, all messages from $\mathcal{M}$ are sent simultaneously by node v in round r. Now we apply (6.1) with a union bound over all possible subsets of sizes ℓ' to k and obtain:

$$\Pr[S_{r,v}] \leq \sum_{\ell' \leq \ell \leq k} \binom{k}{\ell} \cdot \left(\frac{1}{k}\right)^{\ell}.$$

By applying the general inequality $\binom{a}{b} \leq \left(\frac{ea}{b}\right)^{b}$ and the inequalities resulting from the definition of ℓ', namely $\ell' \geq (c+4)\log n$ and $\ell' \geq 2e$, we can upper bound the probability $\Pr[S_{r,v}]$ as follows:

$$\Pr[S_{r,v}] \leq \sum_{\ell' \leq \ell \leq k} \left(\frac{ek}{\ell}\right)^{\ell} \left(\frac{1}{k}\right)^{\ell} = \sum_{\ell' \leq \ell \leq k} \left(\frac{e}{\ell}\right)^{\ell} \leq k\left(\frac{1}{2}\right)^{(c+4)\log n} = k \cdot \frac{1}{n^{c+4}} \leq \frac{1}{n^{c+3}}.$$

Applying the union bound again, we obtain

$$\Pr\left[\bigcup_{1 \leq r \leq 2n, v \in V} S_{r,v}\right] \leq \sum_{1 \leq r \leq 2n, v \in V} \Pr[S_{r,v}] \leq 2n^2 \cdot \frac{1}{n^{c+3}} \leq \frac{1}{n^{c}}.$$

The complement event of $\bigcup_{1 \leq r \leq 2n, v \in V} S_{r,v}$ is that for every round r and every node v, the set of messages sent by node v in round r has size at most $\ell' = O(\log n)$, and this holds with probability at least $1 - \frac{1}{n^c}$ according to the above bound. □

6.2.5 Random Delay Summary

With a slight modification of the proof, the Random Delay technique can also be applied to algorithms in which each node sends more than once. This generalization provides the following guarantee.

Lemma 6.5. *Let $\mathcal{A}$ be an algorithm in the CONGEST model that requires $R(n) = n^{O(1)}$ rounds and in which each node sends messages for a total of $M(n) = n^{O(1)}$ rounds. Then $k = n^{O(1)}$ instances of $\mathcal{A}$ can be executed with high probability in $O((R(n) + kM(n))\log n)$ rounds.*

In this lemma, $n^{O(1)}$ stands for an arbitrary polynomial in n.

6.3 Weighted Graphs

6.3.1 Model

For weighted graphs, there are some modeling aspects that should be considered first. Each edge $e = (u, v)$ is assigned a weight $w(u, v)$, which is initially known to both node u and node v. In the following, the edge weights are always positive integers from 1 to W. For simplicity, we assume that W is polynomial in n, i.e., $W = n^{O(1)}$. This allows, for example, the sum of up to n edge weights to be sent in a message of size $O(\log n)$. We assume that the edge weights represent costs and not delays; direct communication with neighbors still takes only one round. We continue to denote the diameter of the *unweighted* network by D. The following definition is also often useful for distance computations in weighted graphs.

Definition 6.6. *For each pair of nodes u and v and each $h \geq 0$, the* h-distance $\mathrm{dist}^h(u,v)$ *denotes the length of the shortest path from u to v with at most h edges.*

6.3.2 Bellman-Ford Algorithm

In the following, we use a variant of the Bellman-Ford algorithm for SSSP computation. This algorithm works as follows:

Initialization in round 1:

$$\delta_1(v) = \begin{cases} 0 & \text{if } v = s \\ \infty & \text{otherwise} \end{cases}$$

Update in round $r \geq 2$:

$$\delta_r(v) = \min_{(u,v)\in E} (\delta_{r-1}(u) + w(u,v))$$

The correctness and runtime of this algorithm follow from the following invariant.

Lemma 6.7 (Invariant)**.** *After round r, $\delta_r(v)$ corresponds to the length of the shortest path from s to v with at most $r-1$ edges for each node v, i.e., $\delta_r(v) = \mathrm{dist}^{r-1}(s,v)$.*

We omit the proof of this invariant, as it requires only standard arguments that we have already used in several places. The correctness and runtime of the algorithm follow from the invariant, since the overall shortest path always has at most $n-1$ edges. Thus, we obtain the following result.

Theorem 6.8. *In the CONGEST model, the SSSP problem for weighted graphs can be solved in $O(n)$ rounds.*

We consider an example of executing the Bellman-Ford algorithm: In the network of Fig. 6.2, the source node knows the distance 0 to itself at the beginning.

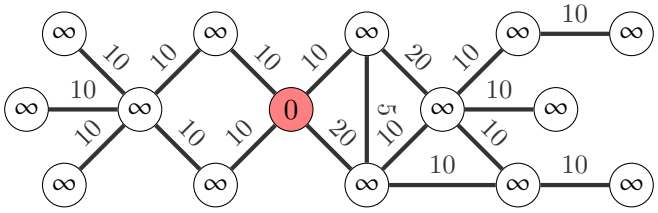


Figure 6.2: SSSP problem for weighted graphs

In the first round, the source node sends its distance to all neighboring nodes. This distance is received by the neighbor nodes in round 2, and the updated value of each neighbor corresponds to the weight of its edge to the source node, see Fig. 6.3.

When these update steps are repeated until no further changes occur, we obtain the result shown in Fig. 6.4. Each node then knows its distance to s.

6.3.3 All-Pairs Shortest Paths

We wish to solve the APSP problem in weighted graphs using the same idea as in unweighted graphs, namely by running an instance of an SSSP algorithm for each node of the network using the

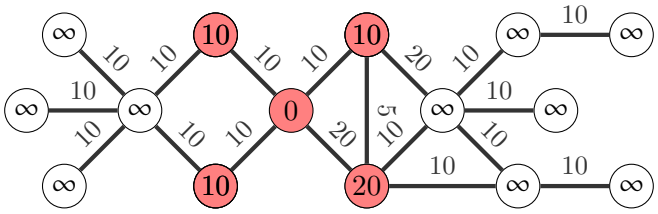


Figure 6.3: SSSP problem for weighted graphs

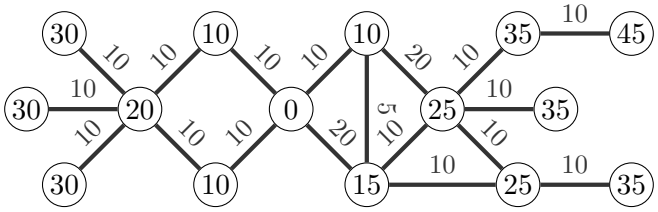


Figure 6.4: SSSP problem for weighted graphs

Random Delay technique. However, in the Bellman-Ford algorithm, every node sends a message in every round. Thus, with the Random Delay technique, one would only obtain a runtime bound of $O(n^2 \log n)$. This is of course worse than $O(n^2)$, the bound one obtains by sequential execution of the n instances of Bellman-Ford. The goal now is to design an SSSP algorithm with a lower number of messages per node. This can be achieved with a kind of "weighted" breadth-first search. Here, we simulate the weighted graph as an unweighted graph by conceptually replacing each edge with a path of corresponding length. This idea can be implemented with the following algorithm:

---

1 s has distance *$ 0 $* to itself , sends in round $w(s,v)$ the distance information $w(s,v)$ to each neighbor node v to v (received by v in round $w(u,v)+1$)

2

3 As soon as distance information δ is received for node $u \neq s$ for the first time:
4 Let r be the current round
5 For each neighbor node v: Send distance information $\delta + w(u,v)$ to v in round $r + w(u,v) - 1$ (received by v in round $r + w(u,v)$)

Analogous to the already known breadth-first search, the following invariant can be established for the "weighted" breadth-first search, whose proof we again omit.

Lemma 6.9 (Invariant). *For each node v: The first distance message is received in round* $\operatorname{dist}(s,v)+1$*; it has the content* $\operatorname{dist}(s,v)$.

The offset +1 arises because round counting starts at 1, but the smallest distance, namely $\operatorname{dist}(s,s)$, has the value 0.

Thus, by simulating breadth-first search, we obtain an SSSP algorithm in which each node sends messages only once, and whose runtime depends on the desired depth of the BFS tree.

Lemma 6.10. *For each $T \geq 0$, a value $\delta(v)$ can be computed for each node v in $T+1$ rounds, such that $\delta(v) = \operatorname{dist}(s,v)$ if $\operatorname{dist}(s,v) \leq T$ and $\delta(v) = \infty$ otherwise, where each node sends messages to its neighbors in at most one round.*

Thus, $O(nW)$ rounds are sufficient to compute the distance $\operatorname{dist}(s,v)$ for each node v, because each edge has a weight of at most W and each shortest path consists of at most $n-1$ edges. This

yields a "pseudopolynomial" SSSP algorithm, whose runtime depends on the value of W (and not on $\log W$, as required by "polynomial" algorithms.)

Lemma 6.11. *In the CONGEST model, the SSSP problem for weighted graphs can be solved in $O(nW)$ rounds, where each node sends messages at most once.*

We now use the Random Delay technique, and applying Lemma 6.5 gives the following theorem for solving the APSP problem.

Theorem 6.12. *In the CONGEST model, there is a Monte-Carlo algorithm that solves the APSP problem for weighted graphs in $O(nW \log n)$ rounds with high probability.*

6.4 Approximation Algorithm for Single-Source Shortest Paths

In the following, we will examine the Single-Source Shortest Path problem for weighted graphs more closely. We have already learned a simple algorithm in Section 6.3.2 that solves the problem in $O(n)$ rounds. The fastest known algorithm for this problem [**ChechikM20**] requires $O((\sqrt{n}D^{1/4} + D) \cdot \log^{O(1)} n)$ rounds with high probability. It can also be shown that algorithms for the SSSP problem cannot be arbitrarily good. In general, $\Omega(\sqrt{n}/\log n + D)$ rounds are required [**PelegR00**]. Finding an approximately "tight" upper or lower bound for this problem is a major open problem.

An "tight" upper/lower bound is only known so far for approximation algorithms. We say that an algorithm computes an (multiplicative) α-approximation for the SSSP problem if, at the end, each node v has a distance estimate $\delta(s, v)$ such that:

$$\operatorname{dist}(s, v) \leq \delta(s, v) \leq \alpha \operatorname{dist}(s, v) .$$

On the one hand, in general, $\Omega(\sqrt{n/(\alpha \log n)}+D)$ rounds are required to compute an α-approximation for the SSSP problem [**Elkin06**]. On the other hand, a $(1+\epsilon)$-approximation for the SSSP problem can be computed with high probability in $O((\sqrt{n}+D) \cdot \log^{O(1)}(n)/\epsilon^{O(1)})$ rounds [**BeckerFKL21**].

We will now develop a slightly less efficient approximation algorithm that requires $O(n^{2/3} \log^2(n)/\epsilon + D)$ rounds and computes a $(1+\epsilon)$-approximation with high probability, where $0 < \epsilon \leq 1$ is a parameter for the desired approximation quality. For this approximation algorithm, we use two helpful lemmas.

Lemma 6.13. *Let h be a parameter, and let Z (centers) be a set to which each node was added independently with probability $p = ((c+2)\ln n)/h$. Then, with probability at least $1 - \frac{1}{n^c}$: For every pair of nodes u and v, there is a shortest path from u to v that has fewer than h edges or contains a center among the first h nodes different from the source node.*

Lemma 6.14. *Let Z be a subset of nodes and h be a parameter. In $O((|Z| + h)\log(nW)\log(n)/\epsilon)$ rounds, an approximate distance $\widetilde{\operatorname{dist}}(x, v)$ can be computed for each $x \in Z$ and each $v \in V$ (which v knows at the end), such that with high probability:*

$$\operatorname{dist}(x, v) \leq \widetilde{\operatorname{dist}}(x, v) \leq (1+\epsilon)\operatorname{dist}^h(x, v) .$$

We will prove these lemmas later and focus in the following on developing an approximation algorithm with the help of the two lemmas. The basic idea of the algorithm is to reduce the SSSP computation to a relatively small overlay network H with the centers as nodes, i.e., a network of the form $H = (Z, Z \times Z)$ with weights $w_H(x, y) = \widetilde{\operatorname{dist}}(x, y)$ for all $x, y \in Z$, where $\widetilde{\operatorname{dist}}(x, y)$ denotes a

distance approximation to be determined further. Note that the overlay network is created only conceptually; the edges between the centers are *not* inserted into the communication network.

We first formulate our SSSP approximation algorithm for an arbitrarily choosable parameter h and later determine a good choice for this parameter. The algorithm consists of five steps:

1. Internally for each node v: Add v to Z with probability $p = ((c+2)\ln n)/h$, always add s to Z
2. Compute, for all pairs $x \in Z, v \in V$, approximate distances $\widetilde{\operatorname{dist}}(x,v)$, such that: $\operatorname{dist}(x,v) \leq \widetilde{\operatorname{dist}}(x,v) \leq (1+\epsilon)\operatorname{dist}^h(x,v)$
3. Make $\widetilde{\operatorname{dist}}(x,y)$ for all pairs $x, y \in Z$ known throughout the network via up- and downcasts
4. Internally for each node v: Construct graph $H_v = (Z \cup \{v\}, (Z \cup \{v\})^2)$ with weight $w_{H_v}(x,y) = \widetilde{\operatorname{dist}}(x,y)$ for each edge (x,y)
5. Internally for each node v: Compute $\delta(s,v) := \operatorname{dist}_{H_v}(s,v)$ as the result

6.4.1 Runtime

For the runtime analysis, we first observe that the nodes of the network do not need to communicate at all for steps 1, 2, and 5. Step 2 requires $O((|Z|+h)\log(nW)\log(n)/\epsilon)$ rounds according to Lemma 6.14, and step 3 requires $O(|Z|^2 + D)$ rounds through a combination of multiple upcast (see Section 2.6.1) and multiple downcast (see Section 2.5.1).

The number of centers $|Z|$ is in expectation $O(\frac{n}{h}\log n)$, and by applying the Chernoff bound (Theorem B.13), one can argue that $|Z|$ deviates from this expectation by only a constant factor with high probability (provided $h \leq \frac{n}{c}$ for a sufficiently large constant c). Therefore, our algorithm requires with high probability $O((h + \frac{n}{h}\log n)\cdot\log(nW)\log(n)/\epsilon + \frac{n^2}{h^2}(\log n)^2 + D)$ rounds. For our algorithm we set $h = n^{2/3}$ (so that $h \leq \frac{n}{c}$ holds for sufficiently large n), and we thus obtain a round complexity of $O((n^{2/3} + n^{1/3}\log n)\cdot\log(nW)\log(n)/\epsilon + n^{2/3}(\log n)^2 + D)$. This can be simplified to $O(n^{2/3}\log(nW)\log(n)/\epsilon + D)$.

6.4.2 Correctness

We now show that the result $\delta(s,v) := \operatorname{dist}_{H_v}(s,v)$ computed by the algorithm (for each node $v \in V$) is correct, i.e., achieves the desired approximation quality.

Lemma 6.15. *With high probability:* $\operatorname{dist}_G(s,v) \leq \operatorname{dist}_{H_v}(s,v) \leq (1+\epsilon)\operatorname{dist}_G(s,v)$ *for each node* $v \in V$.

Proof. Let $v \in V$ be arbitrary. Since every node of H_v also appears in G and for every edge weight of an edge (x,y) in H_v the inequality $w_{H_v}(x,y) := \widetilde{\operatorname{dist}}(x,y) \geq \operatorname{dist}_G(x,y)$ holds (i.e., the edge weight does not underestimate the distance in G), the inequality $\operatorname{dist}_{H_v}(s,v) \geq \operatorname{dist}_G(s,v)$ clearly holds for all $v \in V$. It remains to show that $\operatorname{dist}_{H_v}(s,v) \leq (1+\epsilon)\operatorname{dist}_G(s,v)$ for all $v \in V$.

Let π be the shortest path from s to v in G. By repeated application of Lemma 6.13, π can be chosen such that with high probability the path contains a center every at most h edges (see Fig. 6.5). Let $x_1, x_2, \ldots, x_k$ be the sequence of centers on the internal nodes of π, and let $x_0 := s$ and $x_{k+1} := v$. Since $x_0, \ldots, x_k \in Z$, H_v contains edge (x_i, x_{i+1}) for all $0 \leq i \leq k-1$. Also, H_v contains edge (x_k, x_{k+1}). Therefore, there is a path $\pi' = (x_0, x_1, \ldots, x_k, x_{k+1})$ in H_v from $s = x_0$ to $v = x_{k+1}$.

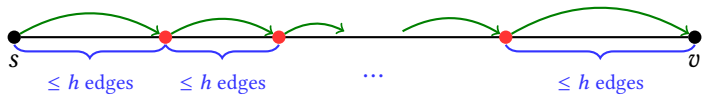


Figure 6.5: Partitioning of the shortest path from s to v such that a center (red) is reached every at most h edges.

We can upper bound the weights of the edges on π' as follows. For all $0 \leq i \leq k$, the subpath of π from x_i to x_{i+1} has at most h edges. Since every subpath of a shortest path is also a shortest path, we have

$$\operatorname{dist}^h_G(x_i, x_{i+1}) = \operatorname{dist}_G(x_i, x_{i+1}) \tag{6.2}$$

for all $0 \leq i \leq k$. According to the algorithm (step 4), the corresponding edge (x_i, x_{i+1}) in H_v has weight $w_{H_v}(x_i, x_{i+1}) = \widetilde{\operatorname{dist}}(x_i, x_{i+1})$. According to Lemma 6.14, $\widetilde{\operatorname{dist}}(x, v) \leq (1+\epsilon)\operatorname{dist}^h(x, v)$. Due to (**??**), we therefore have overall

$$w_{H_v}(x_i, x_{i+1}) \leq (1+\epsilon)\operatorname{dist}(x, v)\,. \tag{6.3}$$

Now $\operatorname{dist}_{H_v}(s, v)$ can be upper bounded as follows: First, the shortest path from s to v in H_v is never longer than the path π' we are considering, due to its property as the *shortest* path, i.e., $\operatorname{dist}_{H_v}(s, v) \leq w_{H_v}(\pi') := \sum_{i=0}^{k} w_{H_v}(x_i, x_{i+1})$. Due to 6.3, we have $\sum_{i=0}^{k} w_{H_v}(x_i, x_{i+1}) \leq (1+\epsilon)\sum_{i=0}^{k}\operatorname{dist}_G(x, x_{i+1})$. By the definition of $x_0, x_1, \ldots, x_{k+1}$ as nodes on a shortest path from $s = x_0$ to $v = x_{k+1}$ in G, we have $\operatorname{dist}_G(s, v) = \sum_{i=0}^{k}\operatorname{dist}_G(x, x_{i+1})$ (the shortest path is in total as long as the sum of its subpaths). Overall, we thus have $\operatorname{dist}_{H_v}(s, v) \leq (1+\epsilon)\operatorname{dist}_G(s, v)$. □

6.4.3 Proof of Lemma 6.13

In our proof, for each pair of nodes $u, v \in V$, we fix *one* of the shortest paths from u to v. Let Z be the set of randomly chosen centers, and let $\pi_1, \pi_2, \ldots \pi_\ell$ (with $\ell \leq n^2$) be those pairwise shortest paths with at least h edges. We wish to show that with high probability, each of these paths contains a center (apart from its respective starting node).

For each such path π_i, the number of centers within its first h nodes different from the starting node is binomially distributed with parameters h and $p = ((c+2)\ln n)/h$. Let A_i be the event that occurs if π_i contains a center within its first h nodes different from the starting node. The complement event $\bar{A}_i$ has probability

$$\Pr[\bar{A}_i] = (1-p)^h = (1-p)^{((c+2)\ln n)/p} = \left((1-p)^{\frac{1}{p}}\right)^{\ln n^{c+2}}\,.$$

Using the inequality (A.18) ($\left(1-\frac{1}{x}\right)^x \leq \frac{1}{e}$), we obtain

$$\Pr[\bar{A}_j] \leq \left(\frac{1}{e}\right)^{\ln n^{c+2}} = \frac{1}{n^{c+2}}\,.$$

We can now estimate the probability that each of the paths $\pi_1, \pi_2, \ldots \pi_\ell$ contains a center within its first h nodes different from the starting node using the complement event:

$$\Pr\left[\bigcap_{i=1}^{\ell} A_i\right] = 1 - \Pr\left[\bigcup_{i=1}^{\ell}\bar{A}_i\right] \geq 1 - \sum_{i=1}^{\ell}\Pr\left[\bar{A}_i\right] \geq 1 - \ell\cdot\frac{1}{n^{c+2}} \geq 1 - \frac{1}{n^c}\,.$$

6.4.4 Proof of Lemma 6.14

We will focus on the following lemma.

Lemma 6.16. *A distance approximation* $\widetilde{\mathrm{dist}}(s,\cdot)$*, for which*

$$\mathrm{dist}(s,v) \le \widetilde{\mathrm{dist}}(s,v) \le (1+\epsilon)\,\mathrm{dist}^h(s,v)$$

holds for each node v*, can be computed in* $O(h\log(hW)/\epsilon)$ *rounds. Each node sends messages in* $O(\log(hW))$ *rounds.*

Lemma 6.14 then follows from Lemma 6.16 by applying the Random Delay technique from Lemma 6.5.

For our proof of Lemma 6.16, we now repeatedly use a subroutine with the following guarantees.

Lemma 6.17 ([**ChechikM20**])**.** *Let* $M > 0$, $T \ge 0$*, and let* G *be a graph with edge weights* $\ge M$*. Then, from a source node* s*, all nodes* v *with* $\mathrm{dist}_G(s,v) \le T$ *and their distances* $\mathrm{dist}_G(s,v)$ *can be computed in* $O(T/M)$ *rounds, where each node sends messages in at most one round.*

The proof of this lemma is an interesting exercise.

In the following, we analyze the result of this subroutine for all $i \ge 0$ with parameters $M_i := \frac{\epsilon 2^i}{h}$ and $T_i := (1+\epsilon)\cdot 2^{i+1}$ in the graph $G_i^{\uparrow}$, in which all smaller weights from G are rounded up to value M_i, i.e.,

$$w_{G_i^{\uparrow}}(u,v) := \begin{cases} M_i & \text{if } w_G(u,v) < M_i \\ w(u,v) & \text{otherwise} \end{cases}$$

for all edges (u,v) from G.

Lemma 6.18. *For all pairs of nodes* $x, y \in V$*:*

1. $\mathrm{dist}_{G_i^{\uparrow}}(x,y) \ge \mathrm{dist}_G(x,y)$
2. *If* $\mathrm{dist}^h_G(x,y) \ge 2^i$*:* $\mathrm{dist}_{G_i^{\uparrow}}(x,y) \le (1+\epsilon)\,\mathrm{dist}^h_G(x,y)$
3. *If* $2^i \le \mathrm{dist}^h_G(x,y) \le 2^{i+1}$*:* $\mathrm{dist}_{G_i^{\uparrow}}(x,y) \le T_i$

Proof. The first property obviously holds because the weights in $G_i^{\uparrow}$ are at least as large as in G. The last property follows from the second property and from the definition of T_i: Since $\mathrm{dist}^h_G(x,y) \ge 2^i$, we have $\mathrm{dist}_{G_i^{\uparrow}}(x,y) \le (1+\epsilon)\,\mathrm{dist}^h_G(x,y)$, and since $\mathrm{dist}^h_G(x,y) \le 2^{i+1}$, we have $(1+\epsilon)\,\mathrm{dist}^h_G(x,y) \le (1+\epsilon)2^i = T_i$, so overall: $\mathrm{dist}_{G_i^{\uparrow}}(x,y) \le T_i$.

For the proof of the second property, we consider a shortest path π from x to y in G (i.e., with total weight $w_G(\pi) = \mathrm{dist}_G(x,y)$). Every edge in $G_i^{\uparrow}$ has a value at most M_i higher due to rounding up than in G. Therefore, the path π, which consists of at most h edges, has a value in $G_i^{\uparrow}$ at most $h\cdot M_i$ higher than in G. Since the path π also exists in $G_i^{\uparrow}$, the shortest path in $G_i^{\uparrow}$ has weight at most that of π in $G_i^{\uparrow}$, i.e., $\mathrm{dist}_{G_i^{\uparrow}}(x,y) \le w_{G_i^{\uparrow}}(\pi)$. Together with the definition of M_i and the inequality $2^i \le \mathrm{dist}^h_G(x,y)$, this yields

$$\begin{aligned}\mathrm{dist}_{G_i^{\uparrow}}(x,y) \le w_{G_i^{\uparrow}}(\pi) \le w_G(\pi) + h\cdot M_i = \mathrm{dist}^h_G(x,y) + \epsilon 2^i &\le \mathrm{dist}^h_G(x,y) + \epsilon\,\mathrm{dist}^h_G(x,y) \\ &= (1+\epsilon)\,\mathrm{dist}^h_G(x,y)\,.\end{aligned}$$

□

We now use the following algorithm to compute the h-distances from a given source node s:

1. For each $0 \leq i < \lfloor \log(hW) \rfloor$, compute the following value $\widetilde{\mathrm{dist}}_i(s,v)$ for each node $v \in V$:

$$\widetilde{\mathrm{dist}}_i(s,v) := \begin{cases} \mathrm{dist}_{G_i^\uparrow}(s,v) & \text{if } \mathrm{dist}_{G_i^\uparrow}(s,v) \leq T_i \\ \infty & \text{otherwise.} \end{cases}$$

2. Internally, for each node v: Compute $\widetilde{\mathrm{dist}}(s,v) := \min_{0 \leq i < \lfloor \log(hW) \rfloor} \widetilde{\mathrm{dist}}_i(s,v)$

The computation of the values $\widetilde{\mathrm{dist}}_i(s,v)$ for each $0 \leq i < \lfloor \log(hW) \rfloor$ is performed by the algorithm from Lemma 6.17 with parameters M_i and T_i in $O(T_i/M_i) = O(h/\epsilon)$ rounds. This results in an overall runtime of $O(h \log(hW)/\epsilon)$ rounds.

The correctness of this algorithm can be argued as follows: Let $v \in V$. For each $0 \leq i < \lfloor \log(hW) \rfloor$, we can distinguish two cases: If $\mathrm{dist}_{G_i^\uparrow}(s,v) \leq T_i$, then $\widetilde{\mathrm{dist}}_i(s,v) = \mathrm{dist}_{G_i^\uparrow}(s,v)$; since due to the second property from Lemma 6.18 we have $\mathrm{dist}_{G_i^\uparrow}(s,v) \geq \mathrm{dist}_G(s,v)$, we obtain $\widetilde{\mathrm{dist}}_i(s,v) \geq \mathrm{dist}_G(s,v)$. Otherwise, $\widetilde{\mathrm{dist}}_i(s,v) = \infty$ and $\widetilde{\mathrm{dist}}_i(s,v) \geq \mathrm{dist}_G(s,v)$ holds trivially. Overall, we thus always have $\widetilde{\mathrm{dist}}_i(s,v) \geq \mathrm{dist}_G(s,v)$. If this inequality holds for each of the values $\widetilde{\mathrm{dist}}_i(s,v)$, then it in particular also holds for the minimum of all these values, i.e., $\widetilde{\mathrm{dist}}(s,v) = \min_{0 \leq i < \log(hW)} \widetilde{\mathrm{dist}}_i(s,v) \geq \mathrm{dist}_G(s,v)$.

Now choose j between 0 and $\lfloor \log(hW) \rfloor$ such that $2^j \leq \mathrm{dist}^h_G(s,v) < 2^{j+1}$ holds; since $\mathrm{dist}^h_G(s,v) \leq hW$ holds, there is always a unique j that satisfies this condition. Due to the third property from Lemma 6.18, we have $\mathrm{dist}_{G_j^\uparrow}(x,v) \leq T_j$ and thus $\widetilde{\mathrm{dist}}_j(s,v) = \mathrm{dist}_{G_j^\uparrow}(s,v)$. Due to the second property from Lemma 6.18, we have $\mathrm{dist}_{G_j^\uparrow}(s,v) \leq (1+\epsilon)\,\mathrm{dist}^h_G(x,v)$ and thus $\widetilde{\mathrm{dist}}_j(s,v) \leq (1+\epsilon)\,\mathrm{dist}^h_G(x,v)$. Overall, we obtain $\widetilde{\mathrm{dist}}(s,v) = \min_{0 \leq i < \lfloor \log(hW) \rfloor} \widetilde{\mathrm{dist}}_i(s,v) \leq \widetilde{\mathrm{dist}}_j(s,v) \leq (1+\epsilon)\,\mathrm{dist}^h_G(x,v)$.

7 Epidemic Information Dissemination

7.1 Introduction

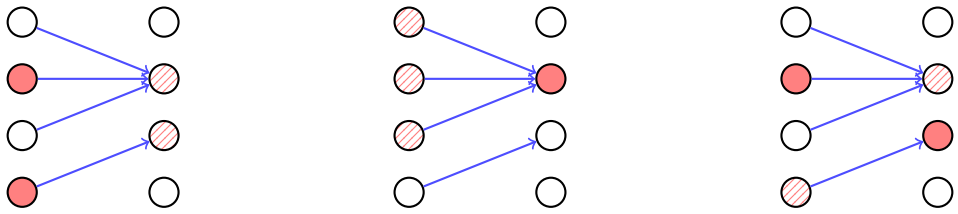
In this section we focus on *Point-to-Point* Networks, in which each node can establish a connection with exactly one other node per round. This particularly models end-to-end connections across lower network layers.

A central question is how to efficiently disseminate information to all nodes of the network.

A potential application scenario for this are replicated databases in Peer-to-Peer networks. Here, the goal is to keep the data identical at all nodes. Thus, as soon as a node receives new information, it should be distributed to all other nodes.

We abstract this setting using the so-called *Phone-Call* model. In this model, we represent the network as a complete graph with n nodes. Communication between nodes takes place in synchronous rounds, and in each round each node can call another node. We distinguish between three communication modes (see Fig. **??**):

1. *Push*: The caller informs the callee
2. *Pull*: The callee informs the caller
3. *Push & Pull*: Combination of push and pull



(a) Push (b) Pull (c) Push & Pull

Figure 7.1: The three communication modes in the Phone-Call Model

An obvious idea for performing a broadcast in the push model is that the i-th informed node calls the i-th uninformed node. To implement this idea, a globally consistent ordering of the nodes must exist. Fig. **??** visualizes a round of the corresponding algorithm.

In this algorithm, each informed node calls an uninformed node, and each uninformed node is called by at most one informed node. Therefore, the number of informed nodes doubles every round, and thus $O(\log n)$ rounds are needed. It is clear that in the push model, doubling the number of informed nodes every round is the best possible case. The message complexity is also optimal at $O(n)$, since each node is called exactly once.

A disadvantage of this algorithm is its lack of fault tolerance: if an information dissemination attempt fails, this can result in very many nodes not being informed. Another disadvantage is that every node must explicitly know all other nodes of the network. Conceptually, it is also unclear how this scheme could be adapted to the pull model.

As a replacement for this type of deterministic broadcast algorithm, the concept of *epicast* has been proposed in the literature [**DemersGHILSSST88**]. The idea is that information propagates epidemically through the network through random selection, like a virus or a rumor. The advantage

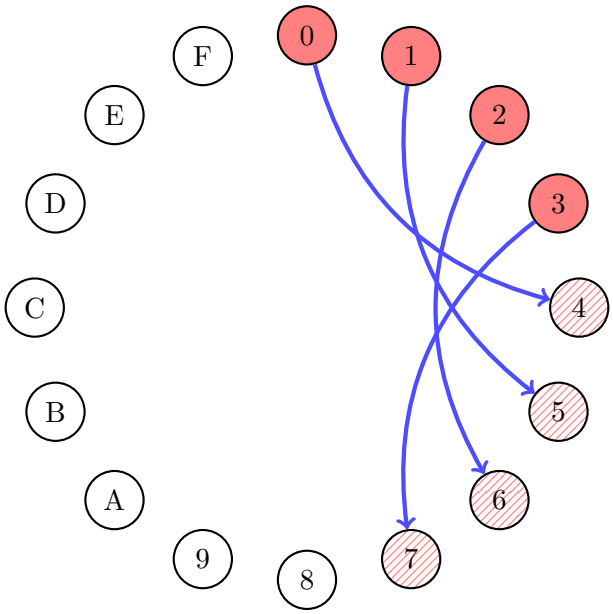


Figure 7.2: Example of a round of a deterministic broadcast algorithm for the push model. Nodes 0, 1, 2, and 3 are already informed, and nodes 4, 5, 6, and 7 receive the information in the current round.

of this approach is that it leads to simple, robust, and fast algorithms. The disadvantage is a certain amount of message overhead.

For the analysis of epicasts, we introduce the *Random-Phone-Call* model, which modifies the Phone-Call model such that in each round each node calls a *uniformly randomly* chosen node (instead of an arbitrarily chosen node). For simplicity, we allow the case that a node calls itself. Thus, in each round the probability that node u calls node v is $\frac{1}{n}$. Fig. shows an example of a round in the random-push model. The random choice of the called node can in principle be implemented without explicit knowledge of the other nodes. Random addressing is supported, for example, in many Peer-to-Peer networks.

As we will see below, epicasts in the Random-Phone-Call model require $O(\log n)$ rounds with high probability. Since a concrete upper bound on the duration of an epicast can be specified, in an application an additional timestamp could be used to check with high probability whether all nodes are already informed and information dissemination can be stopped. Since it never matters in the analysis *which* nodes are already informed, but only *how many*, this automatically gives a certain robustness of the process against transmission errors: In a pessimistic estimate, each erroneous round increases the total number of required rounds by one. Regardless of concrete application scenarios, the Random-Phone-Call model models an interesting mathematical process that can also serve as a simple model for the spread of infections, for example. It should be noted that the analysis extends very easily to a model with transmission probabilities and that theoretically other network structures besides the complete graph can be considered.

7.2 Analysis of the Random-Push Model

Theorem 7.1 ([**FriezeG85**])**.** *Starting from one informed node, in the Random-Push model all n nodes are informed with high probability after $O(\log n)$ rounds.*

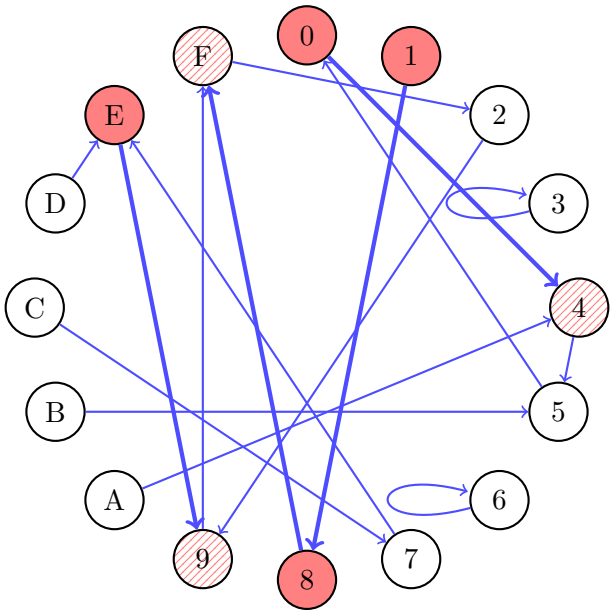


Figure 7.3: Example of a round in the random-push model. Nodes 0, 1, 8, and E are already informed, and nodes 4, 9, and F receive the information in the current round.

We devote the rest of this subsection to the proof of the theorem.

We use the following random variables for our analysis:

- $I(t)$: number of informed nodes at the beginning of round t
- $U(t)$: number of uninformed nodes at the beginning of round t
- $i(t) = \frac{|I(t)|}{n}$: relative proportion of informed nodes at the beginning of round t
- $u(t) = \frac{|u(t)|}{n}$: relative proportion of uninformed nodes at the beginning of round t

Thus we always have $I(t) + U(t) = n$, $i(t) + u(t) = 1$.

In the following analysis, we first fix an arbitrary round t. We first attempt to make general statements about how much the number of informed nodes grows in a round. Every node that becomes informed for the first time in this round is attributed to its smallest informed caller. For this purpose, we introduce the following terminology: A node j performs a **first information dissemination** in round t if

- j is informed at the beginning of round t,
- the node called by j in round t is uninformed at the beginning of round t, and
- the node called by j in round t is not called in round t by any informed node smaller than j

We first provide a bound on the probability that an informed node performs a first information dissemination. Below we denote $\mathcal{I}(t)$ as the set of informed nodes at the beginning of round t and,

for all $j \in \mathcal{I}(t)$, $c_j(t)$ as the node called by node j in round t. We now define the following random variable for each node $1 \leq j \leq I(t)$:

$$X_j(t) := \begin{cases} 1 & \text{if } j \text{ performs a first information dissemination} \\ 0 & \text{otherwise.} \end{cases}$$

Equivalently, the random variable can be written as:

$$X_j(t) = \begin{cases} 1 & \text{if } c_j(t) \notin \mathcal{I}(t) \cup \{c_1(t), \ldots, c_{j-1}(t)\} \\ 0 & \text{otherwise.} \end{cases}$$

In the push model, node j uniformly randomly calls one of the n nodes of the network. The probability that this called node lies in the set $\mathcal{I}(t) \cup \{c_1(t), \ldots, c_{j-1}(t)\}$ is

$$\frac{|\mathcal{I}(t) \cup \{c_1(t), \ldots, c_{j-1}(t)\}|}{n}$$

We observe that the set $\mathcal{I}(t) \cup \{c_1(t), \ldots, c_{j-1}(t)\}$ can have at most size $I(t) + j - 1 \leq 2I(t)$. This yields the following estimate:

$$\begin{aligned} \Pr[X_j(t) = 1] &= 1 - \Pr[X_j(t) = 0] \\ &= 1 - \Pr[c_j(t) \in \mathcal{I}(t) \cup \{c_1(t), \ldots, c_{j-1}(t) \\ &= 1 - \frac{|\mathcal{I}(t) \cup \{c_1(t), \ldots, c_{j-1}(t)\}|}{n} \\ &\geq 1 - \frac{2I(t)}{n} = 2i(t)\,. \end{aligned}$$

We now bound the expected value of $i(t+1)$ in terms of $i(t)$. Let $F(t) := I(t+1) - I(t)$ be the random variable for the number of nodes informed for the first time in round t. By linearity of expectation and the straightforward determination of the expected value of a binary random variable, we obtain:

$$\begin{aligned} \mathrm{Ex}[F(t)] &= \mathrm{Ex}\left[\sum_{j=1}^{I(t)} X_j(t)\right] \\ &= \sum_{j=1}^{I(t)} \mathrm{Ex}[X_j(t)] \\ &= \sum_{j=1}^{I(t)} \Pr[X_j(t) = 1] \\ &\geq \sum_{j=1}^{I(t)} (1 - 2i(t)) \\ &= I(t) - 2i(t)I(t) \end{aligned}$$

The expected value of the relative proportion of informed nodes after round t can now be determined as follows:

$$\begin{aligned}\mathrm{Ex}[i(t+1)] &= \mathrm{Ex}\left[i(t) + \frac{F(t)}{n}\right] \\ &= i(t) + \frac{\mathrm{Ex}[F(t)]}{n} \\ &\geq i(t) + \frac{I(t) - 2i(t)I(t)}{n} \\ &= i(t) + i(t) - 2i(t)^2 \\ &= 2i(t) - 2i(t)^2\end{aligned}$$

This bound on the expectation provides us with the following intuition for the further analysis: As long as relatively few nodes are informed, the term $2i(t)^2$ is negligible and the term $2i(t)$ dominates; the number of informed nodes then approximately grows by a factor of 2 each round.

For the remainder of the proof, we divide the process into two phases:

1. The *growth phase* lasts as long as $I(t) \leq \frac{1}{3}n$ holds.
2. The *shrinking phase* begins as soon as $I(t) > \frac{1}{3}n$ holds.

We show that both phases last $O(\log n)$ rounds with high probability, which implies that the entire process lasts $O(\log n)$ rounds with high probability.

7.2.1 Growth Phase

We first show that in each round of the growth phase, $I(t)$ grows by at least a factor of $\frac{7}{6}$ with constant probability. This occurs when $F(t) > \frac{1}{6}I(t)$. Since in this phase $I(t) \leq \frac{1}{3}n$ – or equivalently $i(t) \leq \frac{1}{3}$ – we obtain

$$\Pr[X_j(t) = 1] \geq 1 - 2i(t) \geq 1 - 2 \cdot \tfrac{1}{3} = \tfrac{1}{3}\,.$$

We would now like to apply the Chernoff bound from Theorem B.14 with $p = \frac{1}{3}$, $\mu = \frac{1}{3}I(t)$ and $\delta = \frac{1}{2}$, to obtain the following estimate:

$$\begin{aligned}\Pr[F(t) \leq \tfrac{1}{6}I(t)] &= \Pr\left[\sum_{j=1}^{I(t)} X_j(t) \leq \frac{1}{6}I(t)\right] \\ &= \Pr\left[\sum_{j=1}^{I(t)} X_j(t) \leq \left(1 - \frac{1}{2}\right) \cdot \frac{1}{3}I(t)\right] \\ &\leq \frac{1}{e^{\frac{(\frac{1}{2})^2}{2} \cdot \frac{1}{3}I(t)}} \\ &= \frac{1}{e^{\frac{1}{24}I(t)}} \leq \frac{1}{e^{\frac{1}{24}}}\,,\end{aligned}$$

where the last inequality holds since $I(t) \geq 1$. We thus obtain

$$\Pr[I(t+1) > \tfrac{7}{6}I(t)] = \Pr[F(t) > \tfrac{1}{6}I(t)] = 1 - \Pr[F(t) \leq \tfrac{1}{6}I(t)] \geq 1 - \frac{1}{e^{\frac{1}{24}}} \geq \tfrac{4}{100}\,.$$

Analogous to the runtime analysis in the distributed MIS algorithm (see Lemma 4.5), we now divide the rounds of the growth phase into *good* and *bad* rounds. Round t is considered good if $I(t+1) > \frac{7}{6}I(t)$, and otherwise bad. After at most $\log_{7/6} \frac{n}{3}$ good rounds, $I(t) > \frac{1}{3}n$ holds (at which point the growth phase is complete). By additionally applying the Chernoff bound, it can be shown that with high probability $O(\log n)$ rounds are needed for $\log_{7/6} \frac{n}{3}$ good rounds to occur. Thus the growth phase requires $O(\log n)$ rounds with high probability.

Our analysis has thus far overlooked a crucial detail: The Chernoff bound applies only to independent variables, but the variables $X_j(t)$ are not independent. For example, if $X_j(t) = 1$ for a node j, this rules out the possibility that other nodes inform the same node for the first time. However, this ultimately reduces the probability that $X_j(t) = 1$ for node $j' \neq j$. However, we will see that it suffices to restrict the degree of independence of our random variables to the following statement:

$$\Pr[X_j(t) = 1 \mid X_1(t) = x_1, \ldots, X_{j-1}(t) = x_{j-1}] \geq 1 - 2i(t) \tag{7.1}$$

for all potential assignments $x_1, \ldots, x_{j-1} \in \{0, 1\}$.

Lemma 7.2 ([**Doerr20**]). *Let $X_1, \ldots, X_k$ be* arbitrary *binary random variables and let $Y_1, \ldots, Y_k$ be* independent *binary random variables. If for all j and all $x_1, \ldots, x_{j-1} \in \{0, 1\}$ it holds that*

$$\Pr[X_j = 1 \mid X_1 = x_1, \ldots, X_{j-1} = x_{j-1}] \geq \Pr[Y_j = 1]$$

then for all $T \geq 0$

$$\Pr\left[\sum_{j=1}^{k} X_j \leq T\right] \leq \Pr\left[\sum_{j=1}^{k} Y_j \leq T\right]$$

In general, one wishes to apply this lemma and then bound $\Pr\left[\sum_{j=1}^{k} Y_j \leq T\right]$ using a Chernoff bound.

In our analysis of the push model, we can define for each $1 \leq j \leq I(t)$ a binary random variable Y_j such that $Y_j = 1$ with probability $1 - 2i(t)$ and $Y_j = 0$ otherwise. With (**??**) this yields

$$\Pr[X_j(t) = 1 \mid X_1(t) = x_1, \ldots, X_{j-1}(t) = x_{j-1}] \geq 1 - 2i(t) = \Pr[Y_j = 1]$$

and with the help of the above lemma, our above analysis changes only in that we apply the Chernoff bound to the sum of the $Y_j(t)$'s instead of the sum of the $X_j(t)$'s – all other arguments remain exactly the same.

Thus, it remains to show that (**??**) holds. It can be easily seen that our previous analysis continues to hold without issues: For each assignment $x_1, \ldots, x_{j-1} \in \{0, 1\}$ we have

$$\begin{aligned}
&\Pr[X_j(t) = 0 \mid X_1(t) = x_1, \ldots, X_{j-1}(t) = x_{j-1}] \\
&\quad = \Pr[c_j(t) \in I(t) \cup \{c_1(t), \ldots, c_{j-1}(t)\} \mid X_1(t) = x_1, \ldots, X_{j-1}(t) = x_{j-1}] \\
&\quad \leq \frac{I(t) + j - 1}{n} \leq \frac{2I(t)}{n} = 2i(t)\,.
\end{aligned}$$

7.2.2 Shrinking Phase

For the analysis of the shrinking phase, we take a different perspective: Instead of analyzing the growth of the set of informed nodes, we explicitly analyze the shrinking of the set of uninformed nodes.

Consider the probability that an arbitrary uninformed node is not informed in round t. Each already informed node calls this uninformed node with probability $1 - \frac{1}{n}$, and there are a total of $I(t)$ such informed nodes. Thus, the node is not informed with probability $(1 - \frac{1}{n})^{I(t)}$. Since in the shrinking phase $I(t) \geq \frac{1}{3}n$ holds, we can derive the following upper bound for this probability:

$$\left(1 - \frac{1}{n}\right)^{I(t)} \leq \left(1 - \frac{1}{n}\right)^{\frac{1}{3}n} = \left(\left(1 - \frac{1}{n}\right)^{n}\right)^{\frac{1}{3}} \leq \left(\frac{1}{e}\right)^{\frac{1}{3}} = \frac{1}{e^{\frac{1}{3}}}.$$

Here, we have used the useful inequality $(1 - \frac{1}{x})^x \leq \frac{1}{e}$ (see (**??**)).

Now consider the probability that a (fixed) uninformed node is not informed over $3(c+1)\ln n$ consecutive rounds. If we consider information dissemination as a Bernoulli trial, it follows that this is equal to the probability that the node is not informed in *one* round ($\leq \frac{1}{e^{1/3}}$), raised to the number of rounds, i.e.:

$$\left(\left(1 - \frac{1}{n}\right)^{I(t)}\right)^{3(c+1)\ln n} \leq \left(\frac{1}{e^{\frac{1}{3}}}\right)^{3(c+1)\ln n} = \frac{1}{e^{\frac{1}{3}\cdot 3(c+1)\ln n}} = \frac{1}{n^{c+1}}. \tag{7.2}$$

We now want to analyze the probability that at least one of the uninformed nodes in $3(c+1)\ln n$ consecutive rounds is not informed. For this, we use the Union Bound (Lemma B.1) and therefore multiply the probability from (**??**) by $U(t) \leq n$ to obtain an upper bound of $U(t) \cdot \frac{1}{n^{c+1}} \leq \frac{1}{n^c}$. The probability of the complement event, namely that after $3(c+1)\ln n$ rounds of the shrinking phase all nodes have been informed, is therefore at least $1 - \frac{1}{n^c}$. With high probability, the shrinking phase therefore lasts $O(\log n)$ rounds.

7.3 Analysis of the Random-Pull Model

We will not explicitly analyze the pull model in this course. In the pull model as well, with high probability $O(\log n)$ rounds are needed until all nodes are informed. The analysis requires similar proof techniques as those we have seen for the push model.

7.4 Analysis of the Random-Push&Pull Model

Both for the Random-Push and the Random-Pull model, with high probability a round complexity of $O(\log n)$ results and thus trivially a message complexity of $O(n \log n)$, since in each round each of the n nodes sends a message. Furthermore, it is clear that $\Omega(n)$ messages are always needed, since each node must receive at least one message. Interestingly, it turns out that the message complexity reduces to $O(n \log\log n)$ when push and pull are combined.

Theorem 7.3 ([KarpSSV00]). *Starting from one informed node, in the Random-Push&Pull model, after $O(\log n)$ rounds with a total of $O(n \log\log n)$ sent messages, all n nodes are informed with high probability.*

The first part of the theorem follows directly from Theorem 7.1, since for the combination of push and pull, no more rounds are ever needed than for push alone. We devote the rest of this subsection to the proof of the message complexity of $O(n \log\log n)$. We use the same notation and

terminology as in the analysis of the Random-Push model (Section **??**). Since we only perform an asymptotic estimate of the message complexity, we always assume that n is large enough, i.e., greater than a constant n_0, which arises implicitly from our analysis.

For the remainder of the proof, we divide the process into three phases:

1. The *growth phase* lasts as long as $I(t) \leq \frac{n}{\ln n}$ holds.
2. The *shrinking phase* lasts as long as $n - \frac{n}{\ln n} > U(t) \geq \sqrt{3cn(\ln n)^5}$ holds.
3. The *closing phase* begins as soon as $U(t) < \sqrt{3cn(\ln n)^5}$ holds.

We show that in each of the three phases, $O(n \log\log n)$ messages are sent with high probability, which implies that a total of $O(n \log\log n)$ messages are sent with high probability. For the growth phase, we directly show that $O(n)$ messages are sent. For the shrinking phase, we show that it lasts $O(\log\log n)$ rounds, which trivially implies that $O(n \log\log n)$ messages are sent. For the closing phase, we show a round complexity of $O(1)$, which implies a message complexity of $O(n)$.

7.4.1 Growth Phase

From the analysis of the Random-Push model, it follows that the growth phase lasts $t^* = O(\log n)$ rounds with high probability. Let $M(t)$ be the random variable for the number of messages sent in round t. We now define the following three random variables, based on the three fundamental types of interactions between two nodes:

- $A(t)$: number of informed nodes that call uninformed nodes (push)
- $B(t)$: number of uninformed nodes that call informed nodes (pull)
- $C(t)$: number of informed nodes that call informed nodes (push&pull)

We observe that $M(t) = A(t) + B(t) + 2C(t)$ holds. (The factor 2 is because when an informed node is called by another informed node, both nodes each send a message.) Trivially, $A(t) \leq I(t)$ and $C(t) \leq I(t)$. This gives us $M(t) \leq 3I(t) + B(t)$. Since each node can be informed only once (i.e., can change its state from uninformed to informed), $\sum_{t=1}^{t^*} B(t) \leq n$. For the number of messages sent during the growth phase, using $I(t) \leq \frac{n}{\ln n}$ and $t^* = O(\log n)$, we now obtain the following estimate with high probability:

$$\begin{aligned}\sum_{t=1}^{t^*} M(t) \leq \sum_{t=1}^{t^*} (3I(t) + B(t)) &= \sum_{t=1}^{t^*} 3I(t) + \sum_{t=1}^{t^*} B(t) \\ &\leq \frac{3n}{\ln n} \cdot t^* + n = O(n)\end{aligned}$$

7.4.2 Shrinking Phase

We first show that in this phase the expected number of uninformed nodes shrinks "quadratically" each round. In general, the probability that an uninformed node calls an uninformed node is $u(t)$. This yields

$$\mathrm{Ex}[U(t+1)] = u(t) \cdot U(t)$$

or equivalently:

$$\mathrm{Ex}[u(t+1)] = \frac{u(t)\cdot U(t)}{n} = (u(t))^2\,.$$

Iterating this equation gives

$$\mathrm{Ex}[u(t+k)] = (u(t))^{(2^k)}\,.$$

We now want to determine how many times this equation must be applied, or how large k must be, for $U(t) < 1$ (or equivalently: $u(t) < \frac{1}{n}$) to hold in expectation, i.e., no nodes remain uninformed. In the shrinking phase $U(t) < n - \frac{n}{\ln n}$ holds (so $u(t) < 1 - \frac{1}{\ln n}$). We observe that for $k \geq 2\log_2(\ln n)$ we obtain the following.

$$(u(t))^{(2^k)} < \left(1 - \frac{1}{\ln n}\right)^{(\ln n)^2} \leq \frac{1}{e^{\ln n}} = \frac{1}{n}\,. \tag{7.3}$$

The idea is now to use a Chernoff bound to show that with high probability, approximately quadratic shrinking occurs in *every* round of the shrinking phase.

First, for each uninformed node j in round t, let the binary random variable $X_j(t)$ be defined as

$$X_j(t) := \begin{cases} 1 & \text{if in round } t \text{ uninformed node } j \text{ calls an uninformed node} \\ 0 & \text{otherwise.} \end{cases}$$

Then $U(t+1) = \sum_{j=1}^{U(t)} X_j(t)$. Furthermore, we have $\Pr[X_j(t) = 1] = u(t) \leq 1 - \frac{1}{\ln n}$, since in the shrinking phase $U(t) \leq n - \frac{n}{\ln n}$ holds. For applying the Chernoff bound (Theorem B.13), we now set $p := 1 - \frac{1}{\ln n}$ (and thus $\mu = \left(1 - \frac{1}{\ln n}\right)\cdot U(t)$) and $\delta := \frac{1}{(\ln n)^2}$. Then, together with the bound $U(t) \geq \sqrt{3cn(\ln n)^5}$ for the shrinking phase, we obtain:

$$\begin{aligned}\Pr\left[U(t+1) \geq (1+\delta)\cdot u(t)\cdot U(t)\right] = \Pr\left[\sum_{j=1}^{U(t)} X_j(t) \geq (1+\delta)\cdot\mu\right] &\leq \frac{1}{e^{\frac{\delta^2}{3}\cdot\mu}} = \frac{1}{e^{\frac{\delta^2}{3}\cdot\frac{(U(t))^2}{n}}} \\ &\leq \frac{1}{e^{\frac{\delta^2}{3}\cdot\frac{3cn(\ln n)^5}{n}}} = \frac{1}{e^{c\ln n}} = \frac{1}{n^c}\,.\end{aligned}$$

For the complement event we now have

$$\Pr\left[U(t+1) < (1+\delta)\cdot u(t)\cdot U(t)\right] = 1 - \Pr\left[U(t+1) \geq (1+\delta)\cdot u(t)\cdot U(t)\right] \geq 1 - \frac{1}{n^c}\,.$$

Thus, $U(t+1) \leq (1+\delta)\cdot u(t)\cdot U(t)$ holds with high probability in each round of the shrinking phase, or equivalently:

$$u(t+1) \leq (1+\delta)\cdot(u(t))^2\,.$$

By iterating this inequality, with high probability, after k rounds of the shrinking phase:

$$u(t+k) \leq (1+\delta)^{(2^k)-1}\cdot(u(t))^{(2^k)} \leq (1+\delta)^{(2^k)}\cdot(u(t))^{(2^k)}\,.$$

With $k := \lceil 2\log_2(\ln n)\rceil$ and $U(t) < n - \frac{n}{\ln n}$ (so $u(t) < 1 - \frac{1}{\ln n}$), we obtain from (**??**):

$$(1+\delta)^{(2^k)}(u(t))^{(2^k)} \leq (1+\delta)^{(2^k)}\cdot\frac{1}{n}$$

Now, due to our choice $\delta = \frac{1}{(\ln n)^2}$, we have $k = \lceil \log_2 \frac{1}{\delta} \rceil$, and we obtain with (??) (the inequality $(1 + \frac{1}{x})^x \leq e$) and $\delta \leq 1$:

$$(1+\delta)^{(2^k)} \leq (1+\delta)^{(2^{1+\log_2 \frac{1}{\delta}})} = (1+\delta)^{2 \cdot \frac{1}{\delta}} \leq e^2 .$$

Since n is "large enough", we may assume that $\sqrt{3cn(\ln n)^5} \geq e^2$ holds. Thus, for $k := \lceil 2 \log_2(\ln n) \rceil$ we obtain the inequality

$$u(t+k) \leq \frac{\sqrt{3cn(\ln n)^5}}{n} .$$

Thus the shrinking phase takes at most $k = O(\log \log n)$ rounds.

7.4.3 Closing Phase

To show that the closing phase consists of at most $2c + 3$ rounds with probability at least $1 - \frac{1}{n^c}$, we analyze the complement event. Since in the closing phase there are at most $\sqrt{3cn(\ln n)^5}$ uninformed nodes, in each round of the closing phase the probability of an uninformed node calling an uninformed node is at most

$$q := \frac{\sqrt{3cn(\ln n)^5}}{n}$$

The probability of an uninformed node in $2c+3$ rounds of the closing phase calling only uninformed nodes (and thus remaining uninformed) is therefore at most q^{2c+3}. The closing phase lasts longer than $2c + 3$ rounds if at least one of the at most $\sqrt{3cn(\ln n)^5}$ uninformed nodes in the $2c + 3$ rounds of the closing phase calls only uninformed nodes. The probability of this event can now be bounded using the union bound as at most

$$\sqrt{3cn(\ln n)^5} \cdot p^{2c+3} = \frac{\left(\sqrt{3cn(\ln n)^5}\right)^{2c+4}}{n^{2c+3}} = \frac{(3c(\ln n)^5)^{c+2} \cdot n^{c+2}}{n^{2c+3}} = \frac{(3c(\ln n)^5)^{c+2}}{n^{c+1}} \leq \frac{1}{n^c} .$$

The last inequality holds because we assume that n is "large enough", specifically so large that $n \geq (3c(\ln n)^5)^{c+2}$ holds. This section is partially based on lectures by Robert Elsässer and Christian Schindelhauer.

A Mathematical Background

A.1 Exponents

The *exponential function* with a real number $b \in \mathbb{R}^{>0} \setminus \{1\}$ as *base* is the function $\exp_b : \mathbb{R} \to \mathbb{R}^{>0}$, defined by the mapping rule $x \mapsto b^x$. For $k \in \mathbb{Z}^{>0}$, b^k is obtained by multiplying b by itself $(k-1)$ times, formally $b^k = \prod_{i=1}^{k} b$, and $b^{1/k}$ is the k-th root of b, i.e., $b^{1/k} = \sqrt[k]{b}$.

The *Euler's number* e can be defined in various equivalent ways, for instance as the limit of the sequence whose n-th term is given by $(1 + \frac{1}{n})^n$, i.e., $e = \lim_{n\to\infty}(1 + 1/n)^n$. If the base in the exponential function is not explicitly specified, the exponential function with base e is typically meant, i.e., $\exp(x) = e^x$.

We need the following rules of calculation for exponents in the lecture (which hold for all $x, y \in \mathbb{R}^{>0}$ and all bases $b \in \mathbb{R}^{>0} \setminus \{1\}$):

$$b^0 = 1 \tag{A.1}$$
$$b^x \cdot b^y = b^{x+y} \tag{A.2}$$
$$\frac{b^x}{b^y} = b^{x-y} \tag{A.3}$$
$$(b^x)^y = b^{x \cdot y} \tag{A.4}$$
$$b^{-x} = \frac{1}{b^x} \tag{A.5}$$

A.2 Logarithms

The inverse function of the exponential function with base b is the *logarithm function* $\log_b : \mathbb{R}^{>0} \to \mathbb{R}$ with base b. The *logarithm* $\log_b(x)$ of x to base b is therefore the unique number y such that $b^y = x$. The logarithm with base e is denoted by $\ln$, i.e., $\ln(x) = \log_e(x)$. If the base is not explicitly specified in the logarithm, in computer science the logarithm with base 2 is typically meant.

We need the following rules of calculation for logarithms in the lecture (which hold for all $x, y \in \mathbb{R}^{>0}$ and all bases $b, b' \in \mathbb{R}^{>0} \setminus \{1\}$):

$$b^{\log_b(x)} = x \tag{A.6}$$
$$\log_b(1) = 0 \tag{A.7}$$
$$\log_b(x \cdot y) = \log_b(x) + \log_b(y) \tag{A.8}$$
$$\log_b\left(\frac{x}{y}\right) = \log_b(x) - \log_b(y) \tag{A.9}$$
$$\log_b\left(\frac{1}{x}\right) = -\log_b(x) \tag{A.10}$$
$$\log_b(x^y) = y \cdot \log_b(x) \tag{A.11}$$
$$\log_b(x) = \frac{\log_{b'}(x)}{\log_{b'}(b)} \tag{A.12}$$

If b is a constant, then $\frac{1}{\log_{b'}(b)}$ is also a constant, and by rule (A.12) for base conversion we have $\log_b(x) = O(\log_{b'}(x))$. For this reason, the base of the logarithm is usually omitted in O notation, and in particular $\ln(x) = O(\log(x))$ holds.

In the lecture we often aim to repeat an algorithm with error probability p (e.g., $p \leq \frac{1}{2}$) sufficiently many times so that the event that all executions of the algorithm were erroneous has a probability of at most q (e.g., $q = \frac{1}{n^c}$). With $k = \lceil \log_p(q) \rceil$ repetitions of the algorithm, the probability that all executions were erroneous equals

$$p^k = p^{\lceil \log_p(q) \rceil} \leq p^{\log_p(q)} = q\,.$$

For $p \leq \frac{1}{2}$ and $q = \frac{1}{n^c}$, therefore $k = \lceil \log_{1/2}(\frac{1}{n^c}) \rceil = \lceil \log_2(n^c) \rceil = O(c \log n)$ repetitions would be necessary so that $p^k \leq \frac{1}{n^c}$ holds.

A.3 Miscellaneous

A.3.1 Rounding operator

The *ceil* operator $\lceil \cdot \rceil$ rounds a real number up to the nearest integer, i.e., $\lceil x \rceil = \min\{z \in \mathbb{Z} \mid z \geq x\}$. For this type of rounding, the inequality $x \leq \lceil x \rceil \leq x + 1$ holds.

A.3.2 Geometric series

The n-th partial sum of a geometric sequence with quotient q is defined as $\sum_{k=0}^{n} q^k$. For $q \neq 1$, we have

$$\sum_{k=0}^{n} q^k = \frac{1 - q^{n+1}}{1 - q} = \frac{q^{n+1} - 1}{q - 1} \tag{A.13}$$

and for $q = 2$, in particular,

$$\sum_{k=0}^{n} 2^k = \frac{2^{n+1} - 1}{1} \leq 2^{n+1} = O(2^n)\,. \tag{A.14}$$

For $|q| < 1$, the limit of the partial sums converges and we have

$$\sum_{k=0}^{\infty} q^k = \frac{1}{1 - q}\,. \tag{A.15}$$

A.3.3 Inequalities

Bernoulli's inequality For every real number $x \geq -1$ and every integer $n \geq 0$, we have

$$(1 + x)^n \geq 1 + nx\,. \tag{A.16}$$

Euler's number From the limit definition of Euler's number, it follows that for every real number $x > 0$

$$\left(1 + \frac{1}{x}\right)^x \leq e \tag{A.17}$$

holds, and for every real number $x \geq 1$

$$\left(1 - \frac{1}{x}\right)^x \leq \frac{1}{e} \tag{A.18}$$

holds.

B Foundations of Probability Theory

B.1 Discrete Probability Spaces

An *discrete probability space* (Ω, p) consists of at most countable[5] *sample space* Ω (whose elements are called outcomes) and a *probability function* $p : \Omega \to [0, 1]$, which maps outcomes to real

[5]Recall: A set is called countable if it has the same cardinality as the set of natural numbers.

numbers between 0 and 1, such that $\sum_{\omega\in\Omega} p(\omega) = 1$. In the case where $\Omega = \{\omega_1, \omega_2, \ldots\}$ is countably infinite, we must have $\sum_{i=1}^{\infty} p(\omega_i) = 1$.[6] An *event* is a set of outcomes, i.e., a subset $A \subseteq \Omega$, where a singleton subset $\{\omega\}$ consisting of a single outcome ω is called an *elementary event.* The *probability distribution* $\Pr : 2^{\Omega} \to [0,1]$ on (Ω, p), which maps events to real numbers between 0 and 1, is defined by $\Pr[A] := \sum_{\omega\in A} p(\omega)$ for all $A \subseteq \Omega$. The *complementary event* of an event A is the event $\bar{A} := \Omega \setminus A$, and from this definition it follows that $\Pr[\bar{A}] = 1 - \Pr[A]$.

Example. Rolling a die can be modeled with a sample space $\Omega = \{1,2,3,4,5,6\}$ with six different outcomes and the probability function p given by $p(1) = p(2) = p(3) = p(4) = p(5) = p(6) = \frac{1}{6}$. Let $G = \{2,4,6\}$ be the event that represents rolling an even number. Then $\Pr[G] = p(2) + p(4) + p(6) = 3 \cdot \frac{1}{6} = \frac{3}{6} = \frac{1}{2}$. Let $U = \{1,3,5\}$ be the event that represents rolling an odd number. Since U is the complementary event of G, we have $\Pr[U] = 1 - \Pr[G] = 1 - \frac{1}{2} = \frac{1}{2}$.

Lemma B.1 (Boole's Inequality, also known as the Union Bound)**.** *Let A and B be two events and $C = A \cup B$ the event that occurs when event A occurs or when event B occurs. Then $\Pr[C] \leq \Pr[A] + \Pr[B]$. More generally, for any finite, nonempty set $\mathcal{A} = \{A_1, \ldots, A_n\}$ of events, we have $\Pr[\bigcup_{i=1}^{n} A_i] \leq \sum_{i=1}^{n} \Pr[A_i]$.*

Example. When rolling a die, let $G = \{2,4,6\}$ be the event that represents rolling an even number and $Q = \{1,4\}$ be the event that represents rolling a perfect square. Then by Boole's inequality, $\Pr[G \cup Q] \leq \Pr[G] + \Pr[Q] \leq \frac{3}{6} + \frac{2}{6} = \frac{5}{6}$. In fact, $G \cup Q = \{1,2,4,6\}$, so $\Pr[G \cup Q] = \frac{4}{6} = \frac{2}{3}$.

B.2 Random Variables and Expected Values

A (real-valued) *random variable* $X : \Omega \to \mathbb{R}$ on a discrete probability space (Ω, p) is a function that assigns to each outcome $\omega \in \Omega$ a real number $X(\omega)$. For each $x \in \mathbb{R}$, we denote the event $X = x$ by $\{\omega \in \Omega \mid X(\omega) = x\}$. The *probability distribution* $\Pr_X : \mathbb{R} \to [0,1]$ of a random variable X is given by $\Pr_X[x] := \Pr[X = x] = \Pr[\{\omega \in \Omega \mid X(\omega) = x\}]$. The *expected value* of X is defined as

$$\mathrm{Ex}[X] := \sum_{\omega\in\Omega} X(\omega) \cdot p(\omega) = \sum_{x\in\mathbb{R}} x \cdot \Pr[X = x]\,.$$

where in the latter sum, terms for values x that are not in the range $X(\Omega)$ are automatically 0.

Example. We again model rolling a die with the probability space (Ω, p) given by $\Omega = \{1, \ldots, 6\}$ and $p(k) = \frac{1}{6}$ for all $k \in \Omega$. Let X be the random variable defined by

$$X(x) = \begin{cases} x & \text{if } k \in \{1,2,3,4,5,6\} \\ 0 & \text{otherwise} \end{cases}\,.$$

Then X corresponds to the number of pips rolled. The probability distribution of X is given by

$$\Pr[X = x] = \begin{cases} \frac{1}{6} & \text{if } x \in \{1, \ldots, 6\} \\ 0 & \text{otherwise} \end{cases}$$

For the expected value of X, we have

$$\mathrm{Ex}[X] = 1 \cdot \Pr[X = 1] + \cdots + 6 \cdot \Pr[X = 6] = 1 \cdot \tfrac{1}{6} + \ldots 6 \cdot \tfrac{1}{6} = (1 + \cdots + 6)\tfrac{1}{6} = \tfrac{21}{6} = \tfrac{7}{2} = 3.5$$

[6] Recall: $\sum_{i=1}^{\infty} p(\omega_i)$ denotes the limit of the partial sums of the series, i.e., $\sum_{i=1}^{\infty} p(\omega_i) = \lim_{n\to\infty} \sum_{i=1}^{n} p(\omega_i)$.

Theorem B.2 (Linearity of Expected Value)**.** *Let X and Y be any two random variables and $Z := X+Y$ be the sum of these random variables. Then* $\mathrm{Ex}[Z] = \mathrm{Ex}[X] + \mathrm{Ex}[Y]$*. More generally, for finitely many random variables $X_1, \ldots, X_n$, we have* $\mathrm{Ex}[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} \mathrm{Ex}[X_i]$.

At this point, it should be noted that for the linearity of expected value, *no* independence of the random variables is required.

Example. We consider rolling two dice, where the random variable X denotes the number of pips on the first roll and Y denotes the number of pips on the second roll. By the linearity of expected value, the expected sum of the pips of both rolls is $\mathrm{Ex}[X + Y] = \mathrm{Ex}[X] + \mathrm{Ex}[Y] = 3.5 + 3.5 = 7$.

B.3 Statistical Independence and Conditional Probability

Two events A and B are called *(stochastically) independent* if $\Pr[A \cap B] = \Pr[A] \cdot \Pr[B]$. More generally, a finite, nonempty set $\mathcal{A} = \{A_1, \ldots, A_n\}$ of events is *(stochastically) independent* if for every nonempty subset $I \subseteq \{1, \ldots, n\}$, we have $\Pr[\bigcap_{i \in I} A_i] = \prod_{i \in I} \Pr[A_i]$.[7]

Example. When rolling a die, let $G = \{2, 4, 6\}$ be the event that represents rolling an even number and $Q = \{1, 4\}$ be the event that represents rolling a perfect square. Then $\Pr[G] = \frac{3}{6} = \frac{1}{2}$ and $\Pr[Q] = \frac{2}{6} = \frac{1}{3}$, so $\Pr[G] \cdot \Pr[Q] = \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6}$. Since $G \cap Q = \{4\}$, we have $\Pr[G \cap Q] = \frac{1}{6} = \Pr[G] \cdot \Pr[Q]$, and thus the events G and Q are independent.

These definitions extend directly to random variables: Two random variables X and Y are called *(stochastically) independent* if the events $X = x$ and $Y = y$ are independent for all $x, y \in \mathbb{R}$, i.e., $\Pr[X = x \cap Y = y] = \Pr[X = x] \cdot \Pr[Y = y]$ for all $x, y \in \mathbb{R}$. More generally, a finite, nonempty set $\mathcal{X} = \{X_1, \ldots, X_n\}$ of random variables is *(stochastically) independent* if the events $X_1 = x_1, \ldots, X_n = x_n$ are independent for all $x_1, \ldots, x_n \in \mathbb{R}$, i.e., $\Pr[\bigcap_{i \in I} X_i = x_i] = \prod_{i \in I} \Pr[X_i = x_i]$ for every nonempty subset $I \subseteq \{1, \ldots, n\}$.

For arbitrary events A and B with $\Pr[B] > 0$, the *probability of A given B* is defined as $\Pr[A|B] := \frac{\Pr[A \cap B]}{\Pr[B]}$. Thus, two events A and B with $\Pr[B] > 0$ are independent if and only if $\Pr[A \mid B] = \Pr[A]$.

Example. When rolling a die, let $G = \{2, 4, 6\}$ be the event that represents rolling an even number, $Q = \{1, 4\}$ be the event that represents rolling a perfect square, and $P = \{2, 3, 5\}$ be the event that represents rolling a prime number. Then $\Pr[Q \mid G] = \frac{\Pr[Q \cap G]}{\Pr[G]} = \frac{1/6}{1/2} = \frac{1}{3} = \Pr[Q]$ and $\Pr[P \mid G] = \frac{\Pr[P \cap G]}{\Pr[G]} = \frac{1/6}{1/2} = \frac{1}{3} \neq \frac{1}{2} = \Pr[P]$.

Lemma B.3 (Law of Total Probability)**.** *Let A and B be arbitrary events and let $\bar{B}$ denote the complementary event of B. Then* $\Pr[A] = \Pr[A \mid B] \cdot \Pr[B] + \Pr[A \mid \bar{B}] \cdot \Pr[\bar{B}]$.

B.4 Probability Distributions

Definition B.4 (Uniform Distribution)**.** *A random variable X is said to be uniformly distributed (i.e., have a uniform distribution) with parameters $a \in \mathbb{N}$ and $b \in \mathbb{N}$, where $a \leq b$, ($X \sim U(a, b)$) if*

$$\Pr[X = k] = \begin{cases} \frac{1}{n} & \text{if } k \in \{a, a+1, \ldots, b\} \\ 0 & \text{otherwise} \end{cases}$$

with $n := b - a + 1$.

[7] The weaker condition that $\mathcal{A} = \{A_1, \ldots, A_n\}$ is *pairwise (stochastically) independent* is stated as $\Pr[A_i \cap A_j] = \Pr[A_i] \cdot \Pr[A_j]$ for all $i, j \in I$ such that $i \neq j$.

A uniformly distributed random variable X describes a random process with n possible outcomes, all of which are equally likely.

Example. Let X be the number of pips rolled on a die. Then X is uniformly distributed with $a = 1$ and $b = 6$ (so $n = 6$) (see Fig. B.1).

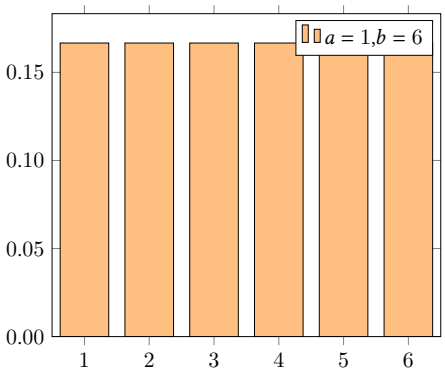


Figure B.1: Uniform distribution with $a = 1$ and $b = 6$ (so $n = 6$)

Definition B.5 (Bernoulli Distribution)**.** *A random variable X is said to follow a Bernoulli distribution with parameter $p \in [0, 1]$ if*

$$\Pr[X = k] = \begin{cases} p & \text{if } k = 1 \\ 1 - p & \text{if } k = 0 \\ 0 & \text{otherwise} \end{cases}$$

Here, p is often referred to as the probability of success.

A Bernoulli-distributed random variable X describes a random process, often called a Bernoulli trial or Bernoulli experiment, in which there are only two possible outcomes, with 1 interpreted as *success* and 0 as *failure*.

Example. Let X be the random variable that is 1 if a six is rolled and 0 otherwise. Then X follows a Bernoulli distribution with $p = \frac{1}{6}$ (see Fig. B.2).

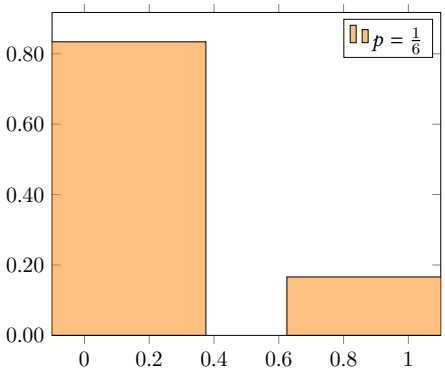


Figure B.2: Bernoulli distribution with $p = \frac{1}{6}$

Lemma B.6. *Let X follow a Bernoulli distribution with probability of* success *p. Then* $\mathrm{Ex}[X] = p$.

Proof. $\mathrm{Ex}[X] = 0 \cdot \Pr[X = 0] + 1 \cdot \Pr[X = 1] = \Pr[X = 1] = p$ □

Definition B.7 (Binomial Distribution)**.** *Let $\{X_1, \ldots, X_n\}$ be a finite set of independent random variables, each Bernoulli-distributed with parameter p, and let $X = \sum_{i=1}^{n} X_i$. Then X is said to follow a* binomial distribution *with parameters $n \in \mathbb{N}$ and $p \in [0, 1]$ ($X \sim B(n, p)$) and we have*

$$\Pr[X = k] = \begin{cases} \binom{n}{k} p^k (1-p)^{n-k} & \text{if } k \in \{0, 1, \ldots, n\} \\ 0 & \text{otherwise} \end{cases}.$$

Here, n is typically the number of trials *and p is the probability of success.*

A binomially distributed random variable X describes the number of successes in a Bernoulli process with n *trials.*

Example. Let X be the number of sixes rolled when a die is rolled three times. Then X follows a binomial distribution with $n = 3$ and $p = \frac{1}{6}$ (see Fig. B.3).

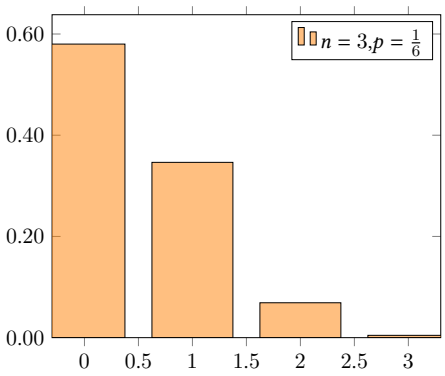


Figure B.3: Binomial distribution with $n = 3$ and $p = \frac{1}{6}$

Lemma B.8. *Let X follow a binomial distribution with parameters n and p. Then $\mathrm{Ex}[X] = pn$.*

Proof. We have $X = \sum_{i=1}^{n} X_i$ for random variables $X_1, \ldots, X_n$, each Bernoulli-distributed with parameter p. By Lemma B.6, $\mathrm{Ex}[X_i] = p$ for all $1 \leq i \leq n$. From the linearity of expected value (Lemma B.2), it follows that $\mathrm{Ex}[X] = \mathrm{Ex}[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} \mathrm{Ex}[X_i] = \sum_{i=1}^{n} p = pn$. □

Definition B.9 (Geometric Distribution)**.** *Let $\{X_1, X_2, \ldots\}$ be a countably infinite set of independent random variables, each Bernoulli-distributed with parameter p, and let $X = \min\{i \geq 1 \mid X_i = 1\}$. Then X is said to follow a* geometric distribution *with parameter $p \in [0, 1]$ ($X \sim G(p)$), and we have*

$$\Pr[X = k] = \begin{cases} p(1-p)^{k-1} & \text{if } k \in \{1, 2, \ldots\} \\ 0 & \text{otherwise} \end{cases}.$$

Here, p is typically referred to as the probability of success.

A geometrically distributed random variable X describes the number of Bernoulli trials needed to achieve a success.

Example. Let X be the number of rolls needed to roll a six. Then X follows a geometric distribution with $p = \frac{1}{6}$ (see Fig. B.4).

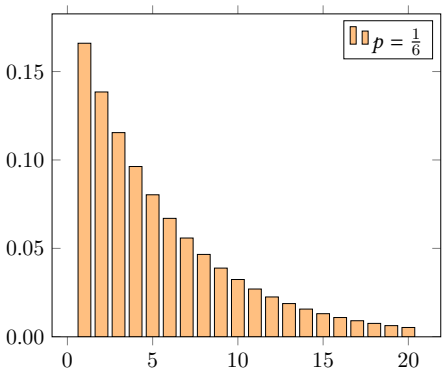


Figure B.4: Geometric distribution with $p = \frac{1}{6}$ (shown for $k \leq 20$)

Theorem B.10. *Let X follow a geometric distribution with parameter p. Then* $\mathrm{Ex}[X] = \frac{1}{p}$.

Proof. The probability of succeeding exactly on the k-th repetition is $\Pr[X = k] = (1-p)^{k-1}p$, i.e., the probability of $k-1$ consecutive *failures* followed by a *success*. By the definition of expected value, $\mathrm{Ex}[X] = \sum_{k=1}^{\infty} k \cdot \Pr[x = k]$ (sum over all realizations of the random variable multiplied by the corresponding probability).

We now perform the following transformations:

$$\begin{aligned}
\mathrm{Ex}[X] &= \sum_{k=1}^{\infty} k \cdot \Pr[X = k] \overset{①}{=} \sum_{k=1}^{\infty} k(1-p)^{k-1}p \overset{②}{=} \sum_{k=0}^{\infty} (k+1)(1-p)^{k}p \\
&\overset{③}{=} \sum_{k=0}^{\infty} k(1-p)^{k}p + \sum_{k=0}^{\infty} (1-p)^{k}p \\
&\overset{④}{=} \sum_{k=1}^{\infty} k(1-p)^{k}p + \sum_{k=1}^{\infty} (1-p)^{k-1}p \\
&\overset{⑤}{=} (1-p) \sum_{k=1}^{\infty} k(1-p)^{k-1}p + \sum_{k=1}^{\infty} \Pr[X = k]
\end{aligned}$$

① Substitution

② Index shift

③ Expanding $(k+1) \cdot \ldots$ and splitting into two sums

④ In the first sum: for $k = 0$, the summand is 0, so we can start the sum from 1 instead
In the second sum: index shift in the other direction

⑤ In the first sum: factoring out $1-p$
In the second sum: substituting the definition of $\Pr[X = k]$

Now the left sum in the last line corresponds to the definition of expected value, and the right sum, as argued in the proof of Theorem 1.10, equals 1. For reference: In the right sum, we consider the probabilities of all possible realizations of k, i.e., the probabilities that the first hit occurs in round k, for all rounds k. These events are disjoint (the first hit cannot occur in two different rounds) and they span the entire event space (one of the events *must* occur, since the waiting time must be a natural number greater than or equal to 1).

Summary: $E[X] = (1-q)E[X] + 1$, or equivalently $E[X] = \frac{1}{q}$. □

Theorem B.11 (Memorylessness of the Geometric Distribution). *For any geometrically distributed random variable X and all $m, n \geq 0$, we have*

$$\Pr[X > m + n \mid X > m] = \Pr[X > n] .$$

Example. Let X be the number of rolls needed to roll a six. Then $\Pr[X > 3 \mid X > 2] = \Pr[X > 2 + 1 \mid X > 2] = \Pr[X > 1] = 1 - p = \frac{5}{6}$. So if no six was rolled after two rolls, the probability of rolling a six on the next roll is $\frac{1}{6}$.

B.5 Concentration Inequalities

For some algorithm analyses, the expected value alone is not informative enough. One often asks how likely it is to deviate from the expected value. It is often interesting to bound the deviation by a *multiplicative factor*. A simple way to make such bounds is given by Markov's inequality.

Theorem B.12 (Markov's Inequality). *) Let X be a non-negative*[8] *random variable and let $\alpha > 0$ be arbitrary. Then*

$$\Pr[X \geq \alpha \cdot \mathrm{Ex}[X]] \leq \frac{1}{\alpha} .$$

Example. Let X be the number of pips rolled on a die. If we want to estimate how likely it is to roll at least four pips, Markov's inequality – with the knowledge that $\mathrm{Ex}[X] = 3.5$ – gives $\Pr[X \geq 4] = \Pr[X \geq \frac{8}{7} \cdot \mathrm{Ex}[X]] \leq \frac{7}{8}$. The correct result, however, is $\Pr[X \geq 4] = \frac{1}{2}$.

Another important tool for such bounds is the so-called Chernoff bound. In addition to a general formulation of the Chernoff bound, the literature provides several specializations that allow relatively strong statements. In the following, we state a Chernoff bound that has proven particularly useful for the analysis of randomized algorithms and that can be applied especially to binomially distributed random variables. Fig. B.5 visually illustrates how a binomially distributed random variable concentrates around its expected value as the number of trials n increases.

For the Chernoff bound, we consider a random variable $X = \sum_{i=1}^{n} X_i$ that is the sum of n *independent* Bernoulli-distributed (i.e., binary) random variables X_i, all of which have their probabilities of success bounded above (or below) by p. The random variable X thus corresponds to the number of successes achieved in the n trials. The Chernoff bound then allows us to state how close X is to $\mu := pn$. If the probabilities of success of the X_i's are all *equal* to p, then X follows a binomial distribution (with parameters n and p), and we have $\mu = \mathrm{Ex}[X]$.

Theorem B.13 (Chernoff Bound for the Upper Tail). *Let $X_1, \ldots, X_n \in \{0, 1\}$ be* independent *binary random variables with $\Pr[X_i = 1] = p$ and let $\mu := pn$. Then for every $\delta > 0$:*

$$\Pr\left[\sum_{i=1}^{n} X_i \geq (1 + \delta) \cdot \mu\right] \leq \frac{1}{e^{\frac{\min\{\delta, \delta^2\}}{3} \cdot \mu}}$$

Theorem B.14 (Chernoff Bound for the Lower Tail). *Let $X_1, \ldots, X_n \in \{0, 1\}$ be* independent *binary random variables with $\Pr[X_i = 1] = p$ and let $\mu := pn$. Then for every $\delta \in [0, 1]$:*

$$\Pr\left[\sum_{i=1}^{n} X_i \leq (1 - \delta) \cdot \mu\right] \leq \frac{1}{e^{\frac{\delta^2}{2} \cdot \mu}}$$

[8] The random variable X is non-negative if for all $x < 0$, we have $\Pr[X = x] = 0$.

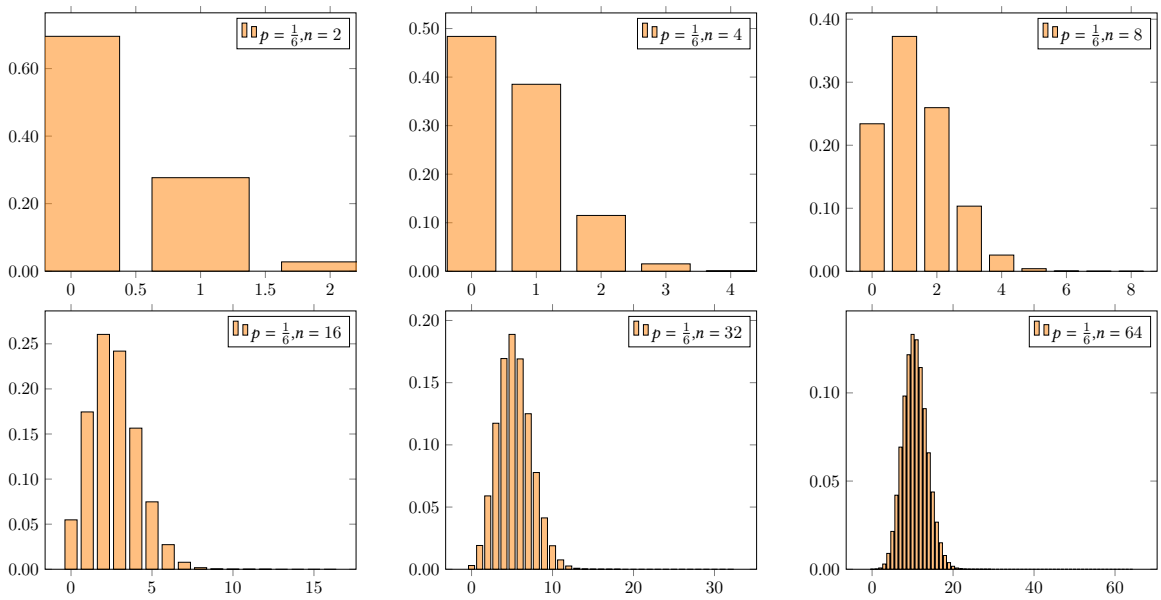


Figure B.5: As the number of trials increases, the realizations of a binomially distributed random variable (here with probability of success $p = \frac{1}{6}$) concentrate around its expected value