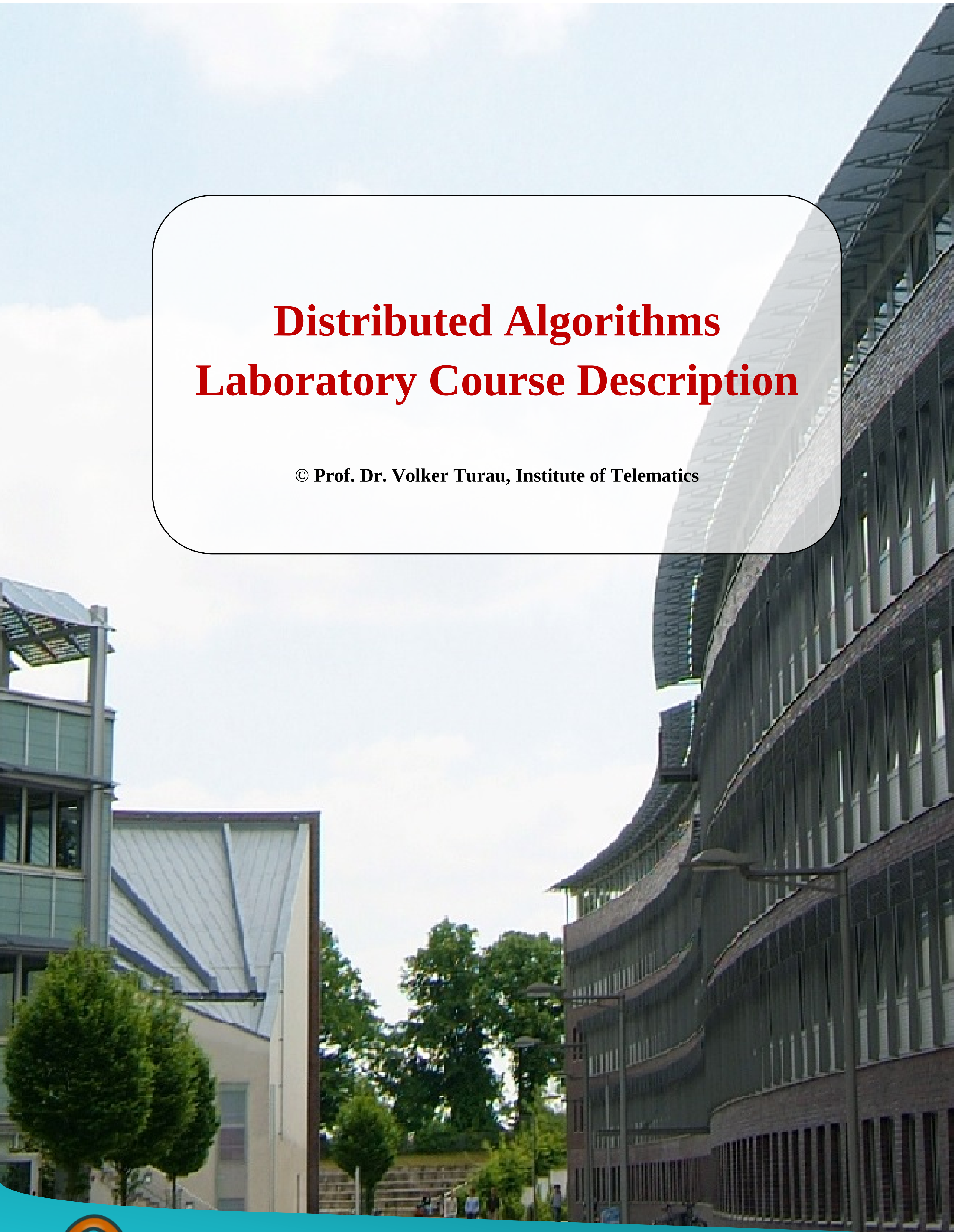


Distributed Algorithms Laboratory Course Description

Institute of Telematics
Hamburg University of Technology

TUHH
Hamburg University of Technology

This document describes the lab of the master lecture Distributed Algorithms from Technical University Hamburg. The ubiquitous availability of access points to the internet and the ever growing data transmission rates of computer networks have lifted distributed computing to a central subject of computer science with high relevance for modern society. Distributed algorithms are a special type of parallel algorithm. They are executed concurrently, with separate parts of the algorithm being run simultaneously on independent networked computers that have limited information about what the other computers are doing. The goal of the lab is to convey techniques and methods taught in this lecture. The covered topics range from basic algorithms such as leader election and tree algorithms to more advanced subjects such as randomized algorithms and synchronizers. The lecture covers besides fundamental problems such as minimal spanning trees, colorings, and matchings also practical algorithms, e.g., for mutual exclusion and blockchains.

Special thanks go to Christoph Weyer for many discussions and for designing all graphics.

Hamburg University of Technology

turau@tuhh.de

Contents

Task 1.1 : Graphs

The network on which a distributed algorithm is executed is represented as a graph. Often special graphs such as a ring, a path or a complete graph are used. Determine for each of these three graphs the number m of edges, Δ, $\mathtt{Diam}$, and χ depending on the number n of nodes!

Task 1.2 : Hypercube Graph Q_l

The *hypercube graph* Q_l is a regular graph with 2^l nodes. The set of nodes V is the set of all binary vectors of length l. Two nodes are neighbors if their binary vectors differ exactly in one position.

1. Draw the hypercube graphs Q_1 to Q_4!
2. Determine $\mathtt{deg}(v)$ for a node v! How many edges does Q_l have?
3. For $l \geq 1$ determine $\Delta(Q_l)$, $\mathtt{Diam}(Q_l)$, and $\chi(Q_l)$!

Task 1.3 : Algorithm $\mathcal{A}_{\mathsf{MAX}}$

Algorithm $\mathcal{A}_{\mathsf{MAX}}$ is designed to run in the asynchronous model. Consider the first two executions demonstrated on the lecture's slides (without and with delay). Determine the number of messages send in each asynchronous round!

Task 1.4 : Algorithm $\mathcal{A}_{\mathsf{MAX}}$

Determine the minimal and the maximal number of messages send for an execution of algorithm $\mathcal{A}_{\mathsf{MAX}}$ for a ring with five nodes. Consider all cases (delays, ordering of values of $\mathtt{x}$)!

Task 1.5 : Algorithm $\mathcal{A}_{\mathsf{MAX}}$

As demonstrated in the lecture algorithm $\mathcal{A}_{\mathsf{MAX}}$ does not work correctly if only started by a single node. Modify this algorithm such that it works correctly even when started by a single node only! What is the price to pay for this feature?

Task 1.6 : Algorithm $\mathcal{A}_{\mathsf{SMAX}}$

Consider the synchronous model. Explain why algorithm $\mathcal{A}_{\mathsf{SMAX}}$ may require less messages compared to executing algorithm $\mathcal{A}_{\mathsf{MAX}}$ in this model.

Task 2.1: Hirschberg-Sinclair's Algorithm

Is the algorithm of Hirschberg-Sinclair a deterministic or a non-deterministic algorithm when executed in the asynchronous model? Is it determinate or non-determinate?

Consider a ring with seven nodes with identifiers $v_0, v_2, v_6, v_5, v_4, v_1, v_3$ (in that order). Describe the sequence of messages for this ring when the algorithm of Hirschberg-Sinclair is executed. What is the total number of messages sent? Does the total number of messages depend on the selection of nodes that start the algorithm?

Task 2.2: Tree Broadcast

The task of the *broadcast operation* is to deliver a message from a dedicated node (the root) to all other nodes of a graph G. A common strategy for a broadcast is to use a spanning tree T of the graph. The root node starts the broadcast by forwarding the message to all its children in T. Each internal node receiving the message does the same. This called *Tree broadcast*.

Let $G = (V, E)$ be an undirected graph and T be a spanning tree of G with root r. Assume that each node of T knows its parent and the set of its children. Note that root has no parent.

1. Give a formal description of an algorithm $\mathcal{A}_{\mathsf{TB}}$ for the tree broadcast!
2. Determine time and message complexity of $\mathcal{A}_{\mathsf{TB}}$ in the asynchronous model in terms of T!
3. What is the influence of the choice of spanning tree T of G on the time complexity of $\mathcal{A}_{\mathsf{TB}}$? Consider a complete graph!
4. What would be an optimal choice for T?
5. Propose an extension of Algorithm $\mathcal{A}_{\mathsf{TB}}$ such that the root knows when the algorithm has terminated! Compute message and time complexity of the new algorithm.
6. Propose a distributed algorithm to count the number n of nodes in the network using T. Upon termination each node should know n.

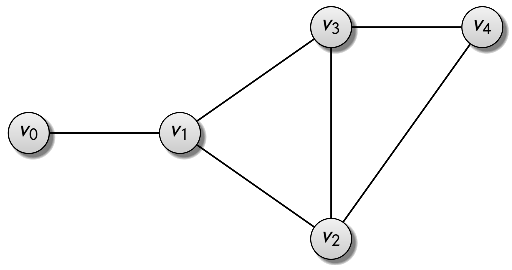
Task 2.3: Flooding

Another solution to the broadcast problem is flooding. Algorithm $\mathcal{A}_{\mathsf{FLOOD}}$ simply forwards the message over all links. A dedicated node called *source* starts the flooding algorithm.

1. Give a formal description of an algorithm $\mathcal{A}_{\mathsf{FLOOD}}$ for flooding! Try to keep the number of messages as low as possible!
2. Determine time and message complexity of $\mathcal{A}_{\mathsf{FLOOD}}$ in the asynchronous model in terms of G!
3. For each node v let $v.\mathtt{parent}$ be the node from which v received the message for the first time. Use this relation to define a directed tree T_{FLOOD} with source as the root. Prove that it describes a tree.
4. Show that in the synchronous case T_{FLOOD} is a breadth-first tree with respect to source.
5. Show that in the asynchronous case T_{FLOOD} may have height $n - 1$.
6. Propose an extension of Algorithm $\mathcal{A}_{\mathsf{FLOOD}}$ such that the source knows when the algorithm has terminated! Compute message and time complexity of the new algorithm.

Task 3.1: Algorithm $\mathcal{A}_{\mathsf{DFS}}$

Apply algorithm $\mathcal{A}_{\mathsf{DFS}}$ to the following graph (v_0 is the root).



Task 3.2: Algorithm $\mathcal{A}_{\mathsf{BFS}_{\mathsf{BF}}}$

Provide a graph for which the Bellman-Ford algorithm $\mathcal{A}_{\mathsf{BFS}_{\mathsf{BF}}}$ requires at least n^3 messages!

Task 3.3: Algorithm $\mathcal{A}_{\mathsf{DFS}_{\mathsf{C}}}$

The DFS algorithm $\mathcal{A}_{\mathsf{DFS}}$ presented in the lecture requires $2m$ rounds and $2m$ messages. Isreal Cidon proposed a new algorithm $\mathcal{A}_{\mathsf{DFS}_{\mathsf{C}}}$ that improves these numbers considerably. Algorithm $\mathcal{A}_{\mathsf{DFS}_{\mathsf{C}}}$ avoids blocking. Each node informs its neighbors with message `VISIT` when it is visited for the first time, before sending further messages. All neighbors can be concurrently informed in one round. Note that the `DFS` message may be sent to an already visited node (whose `VISIT` has not yet been received). In such a case, two things happen:

- A node receiving the `DFS` message via a link over that it already sent a message `VISIT` (`status = unused`) discards the message
- The arrival of the message `VISIT` over a link marked as `child` indicates to the sender that it has sent the `DFS` message to an already explored node and that it was rejected. The node generates the `DFS` message anew (i.e. calls `search()`) and proceeds to explore the other neighbors.

Each node has a Boolean variable `visited` (initially **`false`**) and the array `status`. The latter indicates the status of each outgoing link, the status of a link is one of `used`, `unused`, `parent` or `child` (initial value is `unused`).

Even though the `DFS` message may be sent unnecessarily to some nodes, both message and time complexity are improved. The code of $\mathcal{A}_{\mathsf{DFS}_{\mathsf{C}}}$ is shown below.

```
function search() {
   neigh_unvis := {u ∈ N(v) | status[u] = unused};
   if (neigh_unvis ≠ ∅) {
      let u ∈ neigh_unvis;
      send(u, DFS);
      status[u] := child;
   } else if (v = root)
      halt;
   else { // backtracking
      let u ∈ N(v) s.t. status[u] = parent;
```

```
        send(u, DFS);
    }
}
```

The code initially executed:

```
init:
    visited := false;
    foreach u ∈ N(v) do
        status[u] := unused;

start: // only called at root node
    visited := true;
    search();
    foreach u ∈ N(v) s.t. status[u] = unused do
        send(u, VISIT);
```

Upon reception of message DFS:

```
receive(w, DFS) :
    if (visited = false) {
        visited := true;
        status[w] := parent;
        search();
        foreach u ∈ N(v) s.t. status[u] ∈ {used, unused} do
            send(u, VISIT);
    } else {
        if (status[w] = unused)
            status[w] := used;
        if (status[w] = child)
            search();
    }
```

Upon reception of message VISIT:

```
receive(w, VISIT) :
    if (status[w] = unused)
        status[w] := used;
    if (status[w] = child) { // DFS was sent along link, receiver discarded it
        status[w] := used;
        search();
    }
```

1. Apply algorithm $\mathcal{A}_{DFS_C}$ to the graph from Task 3.1. Consider different delays of messages!
2. Compute the time complexity and give an upper bound for the message complexity of algorithm $\mathcal{A}_{DFS_C}$!

Task 3.4 : Mutual Exclusion Algorithms

Discuss the message complexity for mutual exclusion for a single execution of a CS for the

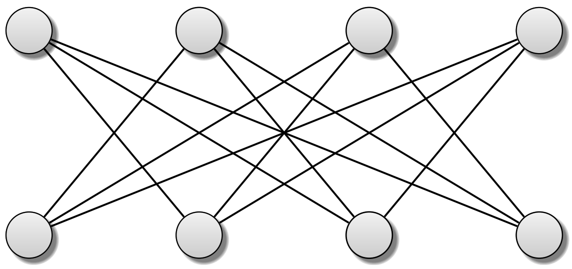
- Centralized Approach,
- Token-Based Algorithm,
- Raymond's Algorithm,
- Ricart-Agrawala Algorithm.

Task 3.5 : Algorithm $\mathcal{A}_{\mathsf{RA}}$

What happens if during the execution of the Ricart-Agrawala algorithm a message is lost? Is it possible that the mutex condition is breached?

Task 4.1: Algorithm $\mathcal{A}_{\text{Red}}$

Assign identifiers to the nodes of the following graph such that algorithm $\mathcal{A}_{\text{Red}}$ requires $\Delta(G)+1$ colors! How many colors are sufficient? Is there an assignment of identifiers such that $\mathcal{A}_{\text{Red}}$ requires only two colors? Can you generalize this example?



Task 4.2: Algorithm $\mathcal{A}_{\text{Red}}$

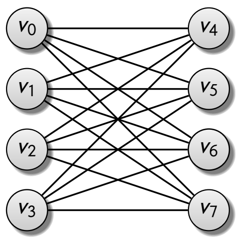
- Determine the message complexity of algorithm $\mathcal{A}_{\text{Red}}$!
- Try to reduce the number of messages sent!
- What changes are necessary to make the algorithm work in the asynchronous model? Does the optimization from the last step still work?

Task 4.3: Coloring a Ring

Adopt algorithm $\mathcal{A}_{3\text{col}}$ so that it works on rings!

Task 4.4: Forest Decompositions

Use Algorithm $\mathcal{A}_{\text{FD}}$ to construct a forest decomposition for the complete bipartite graph $K_{4,4}$. Why does this algorithm compute for this graph a decomposition with four forests?



There exist several forest decompositions for $K_{4,4}$ with three forests, find one! Prove that there cannot be a forest decomposition with two forests.

Task 4.5: Algorithm $\mathcal{A}_{\text{MM}}$

Apply Algorithm $\mathcal{A}_{\text{MM}}$ to the graph $K_{4,4}$ from Task 4.4.

Task 4.6 : Algorithm $\mathcal{A}_{MM}$

In the slides of the lecture algorithm $\mathcal{A}_{MM}$ was applied to a graph resulting in a matching with 3 edges, this is the best result possible. The number of edges in the constructed matching is not fixed. Which steps of the algorithm influence this number? Give an execution of this algorithm that results in a matching with two edges!

Task 5.1: Complexity of Algorithm $\mathcal{A}_{\text{RANK}}$

Give an example for a graph G with $\text{Diam}(G) = 2$ for which algorithm $\mathcal{A}_{\text{RANK}}$ requires n rounds to terminate.

Task 5.2: Coloring based on MIS

Let $\mathcal{A}_{\text{MIS}}$ be any distributed algorithm to compute a maximal independent set (MIS). Consider the following distributed algorithm in the synchronous model to compute a $\Delta + 1$ coloring. Initially all nodes are uncolored.

```
c := 0;
color := ⊥;
while (color = ⊥) {
    Find a MIS in the subgraph induced by the uncolored nodes using $\mathcal{A}_{\text{MIS}}$;
    if (v ∈ MIS)
        color := c;
    c := c + 1;
}
```

Note that the subgraph induced by the uncolored nodes is not necessarily connected. We assume that $\mathcal{A}_{\text{MIS}}$ can be applied in this case. Show that this algorithm computes a $\Delta + 1$ coloring within $\Delta + 1$ iterations.

Task 5.3: Algorithm $\mathcal{A}_{\text{IR}}$ with shorter messages

Consider the following variant of algorithm $\mathcal{A}_{\text{IR}}$ that uses only two parameters in message IR. Is this version correct?

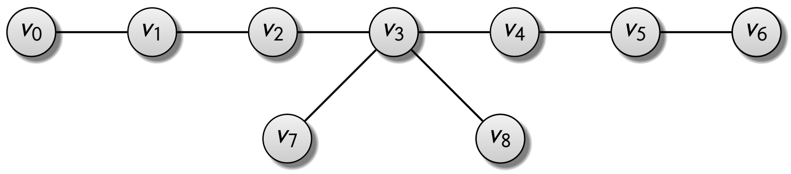
```
init:
    state := active;
    id := random value from {1, ..., k};

start:
    send(left, IR⟨id, 1⟩);

receive(w, IR⟨$id_w$, hop⟩) :
    if (state = passive)
        send(left, IR⟨$id_w$, hop + 1⟩);
    else if (state = active)
        if (hop = n) // message traveled complete ring
            state := leader;
        else
            if (id = $id_w$)
                id := random value from {1, ..., k};
                send(left, IR⟨id, 1⟩);
            else if (id < $id_w$)
                state := passive;
                send(left, IR⟨$id_w$, hop + 1⟩);
```

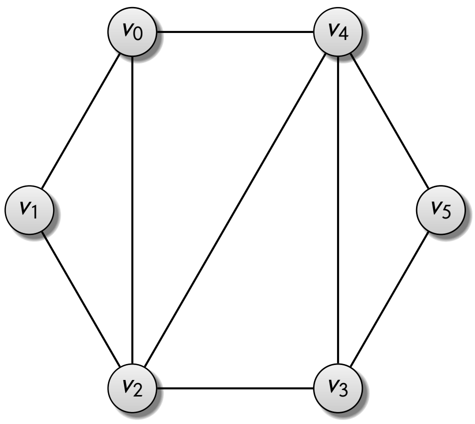
Task 5.4 : Algorithm $\mathcal{A}_{\text{MIS–Rand}}$

Apply algorithm $\mathcal{A}_{\text{MIS–Rand}}$ to the graph following graph! Consider different random choices. What is the most likely result?



Task 5.5 : Algorithm $\mathcal{A}_{\text{Col\_Rand}}$

Apply algorithm $\mathcal{A}_{\text{Col\_Rand}}$ to the graph following graph! Consider different random choices.

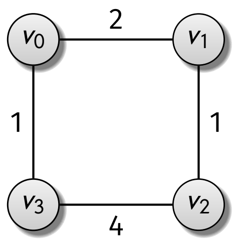


Task 6.1 : Prim's Algorithm

Prove that in Prim's algorithm each node sends at most one message `ISOE` to each neighbor.

Task 6.2 : Prim's Algorithm

Apply algorithm $\mathcal{A}_{\text{PRIM}}$ to the following graph. Start the algorithm at node v_0 and show all messages sent! Also depict the flow of messages in each of the phases. How many messages are sent in each phase?



Suppose there is an additional edge (v_3, v_1) with weight α. Depending on α how would this change the execution of algorithm $\mathcal{A}_{\text{PRIM}}$? What is the total number of messages sent?

Task 6.3 : MST General

In a graph with unique edge weights each MWOE of each MST fragment belongs to the unique MST. Show how to find at least $n/2$ edges of the MST without any communication! Give an example where this leads to a MST, i.e., $n-1$ edges.

Task 6.4 : Leader Election

The distributed implementation of Kruskal's can be used for leader election in general graphs (What are the weights in this case?) Why can algorithm $\mathcal{A}_{\text{PRIM}}$ not be used for this purpose?