Karlsruhe Institute of Technology

Experimental Evaluation of Distributed Node Coloring Algorithms in the SINR Model

Bachelor Thesis of

Markus Schlegel

At the Department of Informatics
Institute of Theoretical Computer Science

Reviewers:	Prof. Dr. Dorothea Wagner
	Prof. Dr. Peter Sanders
Advisors:	Dipl.-Inform. Fabian Fuchs

Time Period: 1st January 2015 – 17th April 2015

Statement of Authorship

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Karlsruhe, 17th April 2015

Abstract

Nodes in wireless ad hoc or sensor networks can only communicate efficiently when interference is kept low. One approach to minimizing interference is to limit the number of concurrently transmitting nodes in every neighborhood by using a periodic schedule. We periodically divide time into slots and assign one time slot to each sender. A sender is only allowed to transmit messages in its time slot. Such a schedule is called a periodic time-division multiple access schedule (TDMA). We can reduce interference in every neighborhood by scheduling nodes according to a TDMA schedule which ensures that nodes, which are neighbors in the communication graph, are transmitting in different time slots. Establishing such a schedule is closely related to the node coloring problem. In a properly colored graph, two neighboring nodes are never assigned the same color. In consequence, if we map each color to a slot in the periodic TDMA schedule, neighboring nodes are never allowed to transmit in the same time slot.

The Signal to Interference and Noise Ratio model (SINR) determines the success of each message transmission by taking the transmission power and the distance of all other concurrent message transmissions as well as a global noise value into account. Therefore, the SINR model describes real-world scenarios with great accuracy. For this reason, the SINR model is often called the physical model.

In this thesis, we experimentally evaluate the runtime performance of three distributed node coloring algorithms in the SINR model with the help of the network simulator framework *sinalgo*. In order to achieve an optimal runtime of the algorithms, some parameters must be experimentally determined, as the algorithms are originally designed to match asymptotic runtime bounds. Additionally we look at some practical improvements and evaluate their impact on the runtime.

Deutsche Zusammenfassung

Effiziente Kommunikation ist in drahtlosen Ad-Hoc- oder Sensor-Aktor-Netzen nur möglich, wenn wenig Interferenz zwischen den Nachrichtenübertragungen auftritt. Eine Möglichkeit, um die Interferenz zu minimieren, ist, verteilt ein periodisches TDMA-Schema aufzubauen, bei dem benachbarte Knoten auf unterschiedliche Zeitschlitze verteilt werden. Der Aufbau eines derartigen TDMA-Schemas lässt sich als Graphfärbbarkeitsproblem betrachten. In einem korrekt gefärbten Graphen haben zwei Nachbarn nie dieselbe Farbe. Wenn wir jede Farbe mit einem Zeitschlitz assoziieren, senden benachbarte Knoten also stets in unterschiedlichen Zeitschlitzen.

Im Signal-to-Interference-and-Noise-Modell (SINR) wird für eine erfolgreiche Nachrichtenübertragung von einem Sender zu einem Empfänger die gleichzeitige Sendeleistung aller anderen Sender, deren Abstand zum Empfänger, sowie ein globaler Rauschwert in Betracht gezogen. Damit ist dieses Modell sehr nah an der Realität und wird deshalb oft auch als Physikalisches Modell bezeichnet.

Wir untersuchen in dieser Arbeit experimentell die Laufzeit dreier verteilter Graphfärbealgorithmen im SINR-Modell mithilfe des Netzwerk-Simulationsframeworks sinalgo. Da diese Algorithmen im Hinblick auf deren asymptotische Laufzeit beschrieben sind, bestimmen wir zunächst jeweils die optimalen Parameter für jeden Algorithmus und betrachten einige Verbesserungen und deren Auswirkungen auf die Laufzeit.

Contents

1. Introduction

Ad hoc or sensor networks lack any initial structure that is shared among every node in the communication graph. Establishing such a structure is crucial to efficient communication mechanisms, since interference makes uncoordinated reliable communication a time consuming task. One approach for coordinated communication that avoids interference is to employ a *Medium Access Protocol* (MAC) such as *Time Division Multiple Access* (TDMA). With TDMA, time is periodically divided into a series of time slots and each node is assigned to one slot per period. A node is then only allowed to transmit in its assigned time slot. In ad hoc or sensor networks, one approach to keep local interference at a minimum is to set up a TDMA schedule in such a way that two neigboring nodes are never transmitting in the same time slot. Establishing such a schedule can be reduced to node coloring of the communication graph. Coloring a graph means assigning each node a color. In a properly colored graph there are no two neighboring nodes that are colored with the same color. If we associate each color with a time slot in the periodic TDMA schedule, no two neigboring nodes are assigned the same time slot. In a communication graph where Δ is the maximum degree the ultimate goal is to find a node coloring using $\Delta + 1$ colors since this is the best coloring that is always possible to find. A node coloring using few colors leads to a TDMA schedule with fewer time slots which means more efficient utilization of the medium's bandwidth.

Distributed node coloring algorithms have been studied extensively in the past. In [FPS04] Finocchi et al. provide an overview of different distributed coloring algorithms in a message-passing model. In message-passing models, messages are transmitted along graph edges. The success of one specific message transmission along an edge e is independent from any other message transmission along a graph edge that shares neither start nor end node with e. In this work we concentrate on node coloring algorithms in the SINR model as described in [GK00]. This model is closer to reality in that it accounts for global interference of all senders instead of only considering local interference by neighboring senders. Any communication scheme that wants to stand the harsh constraints of the SINR model has to drastically restrict the number of simultaneously transmitting nodes. All of the algorithms we study in this thesis achieve this restriction by having any node transmit only with a certain low probability in every time slot. Dropped messages may still occur of course, so nodes have to counteract by transmitting a message for a longer period of time instead of just for one time slot.

The harsh properties of the SINR model also involve that a TDMA schedule obtained from a proper node coloring does not necessarily imply that there will not be any dropped

messages due to interference. It is still possible that a message has to be dropped because the interference caused by senders that are further away than one hop may accumulate at the receiver. The probability of these events, however, is reduced considerably.

1.1. Related Work

The three algorithms we evaluate are described in [FP15] (Simple Coloring), [DT10] (MW Coloring) and [YWHL14] (Yu Coloring). Simple Coloring is based on a well-known coloring algorithm that is covered for example in [BE13]. This algorithm is designed for *message-passing models* such as the $\mathcal{LOCAL}$ model that is described in [Pel00]. Fuchs and Prutikin transferred this algorithm to the SINR model, proving an asymptotical runtime bound of $\mathcal{O}(\Delta \log n)$. Simple coloring produces a node coloring of $\Delta + 1$ colors.

MW Coloring is based on [MW08] which introduces a coloring algorithm that works in the *unstructured radio network model* [KMW04]. Derbel and Talbi transferred the algorithm to the SINR model, proving an asymptotical runtime bound of $\mathcal{O}(\Delta \log n)$. MW coloring produces a node coloring of $\mathcal{O}(\Delta)$ colors. The same work also introduces a way to schedule an interference free TDMA-like MAC protocol in the SINR model. They prove that it is possible to simulate message-passing algorithms of runtime τ in $\mathcal{O}(\Delta \log n + \Delta\tau)$ time slots in the SINR model. A similar result has been obtained by Fuchs and Wagner [FW13].

Node coloring algorithms that work in message-passing models have been studied extensively in the past. The current state of the art for arbitrary graphs is an $\mathcal{O}(\Delta) + \frac{1}{2}\log^* n$ algorithm [BE09]. For growth-bounded graphs there is an algorithm described in [SW08] that reaches the optimal runtime of $\mathcal{O}(\log^* n)$. Finocchi et al. ([FPS04]) evaluate the performance of different node coloring algorithms working in a message-passing model. They also evaluate the effect of varying the graph density or wake-up probability on the runtime of these algorithm. Halldórsson and Mitra ([HM11]) study a more general scheduling problem in the SINR model, transferring a centralized algorithm by [KV10] to a distributed version. In [ALPP09] Avin, Lotker, Pasquale and Pignolet examine edge colorings in the SINR model with uniform transmission power.

1.2. Our Contributions

In this thesis we consider three different distributed node coloring algorithms in the SINR model and evaluate their performance in six different simulated environments. We measure the runtime with respect to a low failure rate and the number of colors used. Since all of these algorithms have so far only been described with regard to their asymptotical runtime, we first determine the constant parameters that lead to optimal runtime in our testing environments. For each algorithm we consider a few practical improvements that do not improve the asymptotical runtime bounds but lead to better performance in our simulated testing scenarios.

In Chapter 2 we define the testing scenarios and metrics. In Chapter 3 we describe and evaluate the Simple Coloring algorithm. In Chapter 4 we describe and evaluate Moscibroda-Wattenhofer-based coloring. In Chapter 5 we describe and evaluate a coloring algorithm designed by Yu et al. Finally, in Chapter 7 we analyze the results and compare the performance of the algorithms.

2. Preliminaries

2.1. Terminology

In this section we explain the most fundamental terms that are used throughout the thesis. In later chapters where we look at the three discussed algorithms in detail, we introduce a few more terms that are specific to each algorithm.

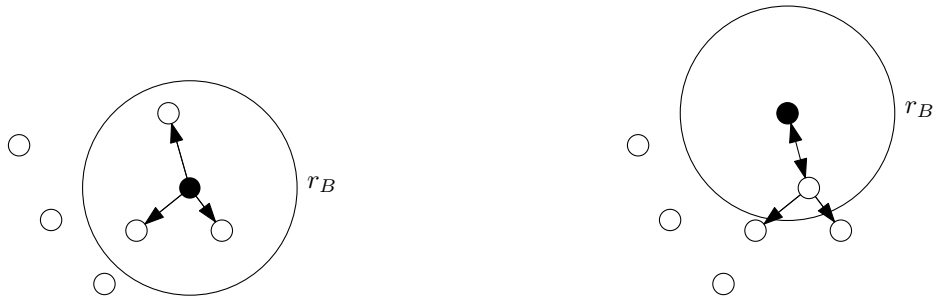
An *undirected graph* $G = (V, E)$ is a set of *nodes* (or *vertices*) V and a set of *edges* $E \subseteq \{\{u, v\}; u, v \in V\}$. In this thesis we may also speak of nodes as *senders* and of edges as *links*. We say that two nodes u and v are *connected* or *adjacent* if and only if $\{u, v\} \in E$. We say that u is a *neighbor* of v if and only if u and v are connected. The *neighborhood* of v is then the set of all neighbors of v. The *degree* of a node is the cardinality of its neighborhood. The maximum degree in a graph is denoted Δ.

We say that a *color* i is a positive integer including zero and $[i] = \{0, 1, 2, \ldots, i\}$ is a set of colors up to i inclusively. Let k be a positive integer. We then say that a graph $G = (V, E)$ is *k-colored* if each node $v \in V$ is associated with a color out of the set $[k]$ via a mapping $f : V \mapsto [k]$. We say that G is *properly k-colored* if G is k-colored and $\forall\{u, v\} \in E : f(u) \neq f(v)$.

In this thesis we look at graphs that are defined by the positions of nodes on an Euclidian plane. $\text{dist}(u, v)$ denotes the Euclidian distance between two nodes u and v. The edge set of a *unit disk graph* (UDG) $G_{r_B} = (V, E)$ is then defined as follows. There is an edge $\{u, v\} \in E$ if and only if $\text{dist}(u, v) < r_B$ where r_B is a parameter called the *broadcasting range*. Therefore the neighborhood of a node v in a UDG consists of all nodes that lie within the broadcasting range r_B around v. An illustration of the process of setting up a UDG is given in Figure 2.1.

In an unit disk graph, an *independent set in terms of R* is a set of nodes $S \subseteq V$ such that $\forall u, v \in S : dist(u, v) > R$. A *maximal independent set (MIS) in terms of R* is an independent set in terms of R such that there is no node $v \in V, v \notin S$ such that $\{v\} \cup S$ is still an independent set in terms of R. If $R = r_B$, we may only speak of a maximal independent set in general. In such a MIS there are no two nodes that are neighbors in the UDG.

The Signal-to-Interference-plus-Noise-Ratio Model (SINR) describes the success of message transmissions in shared medium scenarios with multiple senders by accounting for the accumulative interference and a global noise value. Let some nodes $v \in I \subseteq V$ in a network



(a) Setting up outgoing edges of a node. All nodes inside the radius r_B are connected.

(b) Edges are directed, but for every edge there is an opposite edge, effectively making the UDG an undirected graph.

Figure 2.1.: The communication graph is the unit disk graph with broadcasting range r_B

transmit simultaneously with uniform power P and let N be a global noise value. A message transmission from u to w is successful if and only if

$$\frac{\frac{P}{\text{dist}(u,w)^\alpha}}{\sum_{v \in I} \frac{P}{\text{dist}(v,w)^\alpha} + N} \geq \beta \tag{2.1}$$

where α is the geometric decay of a signal and β is a threshold value depending on the receiver's hardware. One assumes $\alpha \in [2, 6]$ and $\beta \geq 1$, however, advanced interference cancellation techniques have been proposed that make it possible to have $\beta \leq 1$ ([MA04]).

All algorithms we consider work on the *communication graph* of a network of senders. The communication graph is a unit disk graph defined by the broadcasting range r_B. A node will only consider messages that are transmitted by neighboring nodes and discard all other messages. A node is able to decide whether a message is originating at one of its neighbors by looking at the signal strength of the message.

We say that a node is *transmitting with probability p for d time slots* if in each of the d time slots the node transmits with probability p and does not transmit with probability $1 - p$.

Throughout this thesis we report average (arithmetic mean) values a along with the corresponding standard deviation s in the form $a \pm s$.

2.2. Sinalgo

We use Sinalgo ([G+08]) to simulate the coloring algorithms and measure their runtime. Sinalgo is an abbreviation for Simulator for Network Algorithms. The software provides a framework for simulating graph-based networking scenarios. Sinalgo is open-source and it is written in Java.

The Sinalgo framework provides a way to place any number of nodes on a plane according to a *distribution model*. For our purposes we use the preexisting *random distribution model*, which distributes nodes uniformly on the plane. Nodes are able to communicate on the foundation of a communication graph that is set up according to a *connectivity model*. For our purposes we use the preexisting *UDG connectivity model*. Message transmissions are subject to interference according to an *interference model*. For our purposes we use the preexisting *SINR interference model*. Messages take a certain amount of time to be

delivered. This duration is determined by the *message transmission model*, which we set to the preexisting *ConstantTime transmission model* with a constant transmission time of one time slot. Lossy networks can be simulated with the help of a *reliability model*. For our purposes we use the preexisting *ReliableDelivery reliability model* since we want to simulate a network where messages may be dropped solely due to interference and no other reasons. Nodes may change their position on the plane according to a *mobility model*. We disable the mobility option since we are interested in stationary nodes only. Furthermore it is possible to choose between synchronous or asynchronous execution of node implementations. For our purposes we choose synchronous execution.

We generate a subclass of the generic *Node* class in order to implement the three algorithms. Node subclasses override the *Node.handleMessages()* method that is called by the framework in every time slot. The Node subclass may then access its *inbox* where all messages that have arrived in that time slot are placed by the framework. Nodes are now able to perform any computation and transmit messages by using the method *Node.broadcastMessage()*. Nodes may also override the *Node.preStep()* and *Node.postStep()* methods that are called before and after each time slot correspondingly.

There is one object of type *CustomGlobal* per run. With the help of this object it is possible to perform computations before running the algorithm and the object decides when the algorithm has finished. We also use *CustomGlobal* in order to log the overall runtime to a log file.

The simulation of each time slot proceeds according to the following steps. First, *CustomGlobal.preRound()* is called, making it possible to perform preliminary computations. The framework then calls each node to update its set of outgoing connections according to the connectivity model. The framework calls interference tests for all messages that are currently on the shared medium. These interference tests are performed according to the interference model. The framework now calls Node.preStep(), Node.handleMessages() and Node.postStep() on every node in that order. If a node calls Node.broadcastMessage() inside these methods, the corresponding message is placed on the shared medium, making it subject to interference tests in the next time slot. The framework then calls CustomGlobal.hasTerminated() in order to determine if the algorithm should stop or if execution should continue.

2.3. Testing Environment

We test the performance of the three distributed node coloring algorithms in the SINR model. The goal is for each node to decide on a color in such a way that the resulting node coloring is proper.

Nodes are only able to communicate via a shared medium, there are no other means of communication. Message transmissions are therefore inherently unreliable. Efficient communication schemes have to restrict the number of nodes that are transmitting simultaneously at every point in time. Since the nodes start with no shared data structures they only transmit with certain probabilities that can vary throughout the execution of the algorithm.

In our environment, all nodes operate in a synchronous manner, i.e. there are time slots in which nodes are transmitting and receiving. All messages that are transmitted in a certain time slot will be received in the following time slot, if the SINR constraint holds and if sender and receiver are neighbors in the unit disk graph defined by the broadcasting range r_B. The distance between communicating nodes has no influence on the transmission time. The time slots are equal in duration and long enough for every node to finish its calculations.

Every node in the graph holds a globally unique identifier, which we call its ID. Before startup, every node is given the global maximum degree of the communication graph Δ and the total number of participating nodes n. This information is required by the evaluated algorithms.

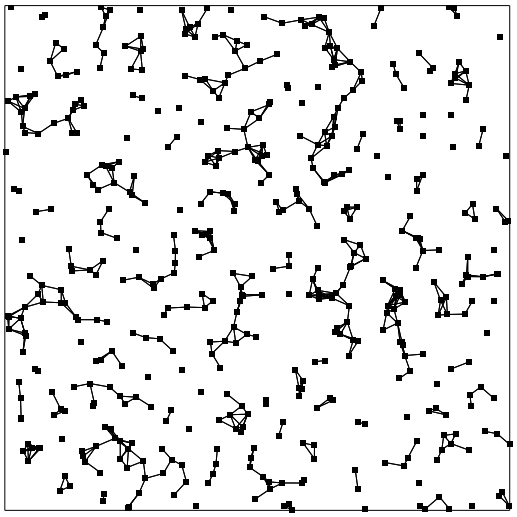


Figure 2.2.: A unit disk graph with 500 nodes and a broadcasting range of $r_B = 40$ on an area of $A = (1000, 1000)$

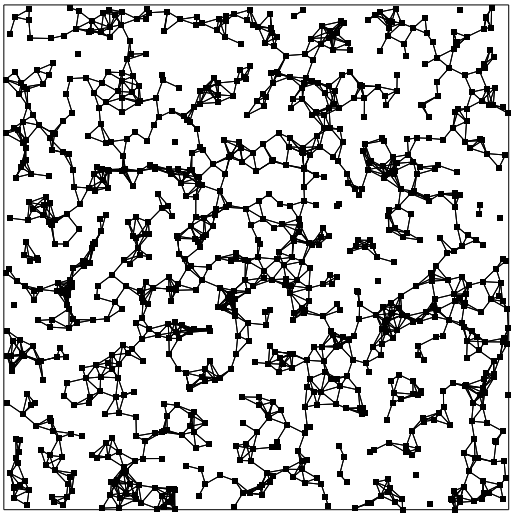


Figure 2.3.: A unit disk graph with 1000 nodes and a broadcasting range of $r_B = 40$ on an area of $A = (1000, 1000)$

A testing scenario is sufficiently defined by the number of participating nodes n, the plane $A(width, length)$ on which the nodes are placed, the broadcasting range r_B and the SINR parameters α, β and N. We define six different configurations of these parameters in which we test the three algorithms. All six configurations use a square plane of $1000m$ by $1000m$ and all of them use $r_B = 40m$. For SINR parameters we use $\beta = 0.7$, $N = 0$ and $\alpha \in \{2, 6\}$. The SINR constraint (2.1) leads to an upper bound for the broadcasting range. We call this bound the *transmission range* $r_T \leq (\frac{P}{\beta N})^{\frac{1}{\alpha}}$. This bound is only defined for $N > 0$, however. In a scenario without any global noise, there is no upper bound for r_B.

For the number of nodes we choose $n \in \{500, 1000, 1500\}$. The nodes are randomly placed on the surface area in an uniform way. On the given plane $A(1000m, 1000m)$ with the broadcasting range $r_B = 40m$ that means that we have one scenario ($n = 500$) where the

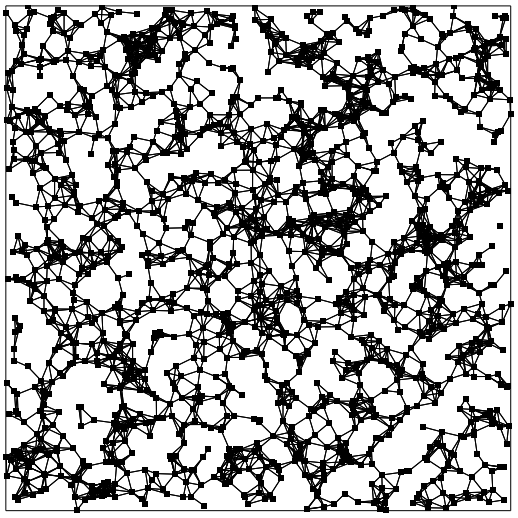


Figure 2.4.: A unit disk graph with 1500 nodes and a broadcasting range of $r_B = 40$ on an area of $A = (1000, 1000)$

communication graph is disjointed and consists of many disconnected smaller graphs, we have one scenario ($n = 1500$) where usually the communication graph is fully connected and we have one scenario ($n = 1000$) that is a compromise between these two extremes. Figure 2.2, Figure 2.3 and Figure 2.4 show communication graphs of size 500, 1000 and 1500 on the surface area of $A(1000m, 1000m)$ with a broadcasting range of $r_B = 40m$. Please note that if we place more nodes on the same surface area the graph becomes denser and therefore the maximum degree Δ increases. All tested algorithms are asymptotically bounded as a function of both n and Δ. In our specific scenarios if we increase n, Δ increases as well. The average maximum degree for $n = 500$ is 7.9 ± 1.0, for $n = 1000$ it is 13.0 ± 1.3 and for $n = 1500$ it is 17.3 ± 1.4.

For our evaluation we vary α because one of the three evaluated algorithms varies the broadcasting range throughout execution. Since higher α values means stronger signal decay this algorithm might benefit from $\alpha = 6$ more than the other two. We do not vary β since we determined in a preliminary evaluation that no algorithm would particularly benefit from lower or higher β values. For practical purposes we choose β to be comparatively low since higher values would lead to impractically high runtimes. The same argument applies to the noise value N. With a global noise value of $N = 0$, a scenario where every node is transmitting with the same power P is equivalent to a scenario where every node is transmitting with the same power λP for any scalar λ. We can therefore freely choose the transmission power $P \leftarrow 1$.

The six testing scenarios are summarized in Table 2.1.

Table 2.1.: Summary of testing scenarios

Scenario	1	2	3	4	5	6
n	500	500	1000	1000	1500	1500
α	2	6	2	6	2	6
β	0.7	0.7	0.7	0.7	0.7	0.7
N	0	0	0	0	0	0
A	$(1000m)^2$	$(1000m)^2$	$(1000m)^2$	$(1000m)^2$	$(1000m)^2$	$(1000m)^2$
r_B	40	40	40	40	40	40
Average Δ	7.9 ± 1.0	7.9 ± 1.0	13.0 ± 1.3	13.0 ± 1.3	17.3 ± 1.4	17.3 ± 1.4

2.4. Metrics for Evaluation

All nodes are uniformly distributed on the plane A. Every node is being woken up in time slot $t_0 = 0$. We measure the number of time slots it takes until the communication graph is properly colored. If the algorithm comes to a halt but the communication graph is not colored properly, we count the run as a failure. This might happen if urgent message transmissions do not succeed. We tune the parameters of the algorithms in such a way that the success rate $s \geq 99\%$.

Apart from the number of time slots it takes an algorithm to produce a proper node coloring, another interesting metric is the number of used colors. We do not have to measure the number of colors since we know it a priori for all evaluated algorithms. Nevertheless we want to keep this metric in mind, since in general it is harder to produce a proper coloring with fewer colors and such a coloring is more useful for the resulting TDMA schedule.

3. Simple Coloring

The Simple Coloring algorithm as described by Fuchs and Prutkin in [FP15] consists of two separate parts. In the first part a proper node coloring of 4Δ colors is computed. In the second part the number of colors is reduced to $\Delta + 1$. The general idea is to quickly erect a non-optimal coloring in the first part that is then used to coordinate medium access in the second part. In the SINR model every message transmission is only successful with a certain (low) probability. In Simple Coloring, nodes counteract this problem by drawing colors from a comparatively large set of colors in a random manner. The chance of two neighboring nodes choosing the same color from that set is very low. It is still possible of course, so nodes do have to communicate their choices. However, since conflicts are rare and if a conflict occurs it can be resolved by both conflicting nodes, it is not urgent that every message is received with certainty.

Algorithm 3.1: Rand4DeltaColoring for node v

```
   Data: Color of node v c_v, color set F_v
   // Initialization
1  F_v ← [4Δ]
2  c_v ← F_v.getRandom()
   // Code
3  for d_1 time slots do
4      if c_v ∉ F_v then
5          c_v ← F_v.getRandom()
6      F_v ← [4Δ]
7      transmit a message containing c_v with probability p_1
8      foreach received color c_w from a neighbor w do
9          F_v ← F_v \ {c_w}
```

Simple Coloring depends on four parameters p_1, d_1, p_2 and d_2. We later explain how to find a parameter combination that leads to the best overall performance of this algorithm.

In the first part of the algorithm (Rand4DeltaColoring) as shown in Algorithm 3.1, every node v draws a random color c_v from a set of colors F_v of size $4\Delta + 1$. It then transmits the chosen color with probability p_1 in every time slot until the algorithm comes to a halt after d_1 time slots. If a message is received in which a neighbor u tells v about

Algorithm 3.2: ColorReduction for node v

Data: The color obtained from Rand4DeltaColoring c_v and a color set F_v

```
// Initialization
F_v ← [Δ]
// Code
foreach color c_i = i ∈ [4Δ] do
    if c_v ≠ i then
        listen for d_2 time slots
        foreach received color c_w from a neighbor w do
            F_v ← F_v \ {c_w}
    else
        c_v ← F_v.getRandom()
        transmit a message containing c_v with probability p_2 for d_2 time slots
```

its color c_u, v has to remove that color from its color set. In the next time slot v checks whether $c_v \in F_v$. If it is not, v has to draw a new random color and transmit that color to its neighbors again. After a certain amount of time slots d_1, all nodes will have decided on a final color and no more conflicts occur. That is when the nodes transition to the second part of the algorithm.

In the second part (ColorReduction) as shown in Algorithm 3.2, nodes only transmit during a time interval that is defined by the color on which they decided in the first part. Nodes with color 0 only transmit during the first time interval, nodes with color 1 only transmit during the second one and so on. In consequence, the number of concurrently transmitting nodes is reduced significantly. In addition, neighboring nodes never transmit at the same time since the colors from Rand4DeltaColoring already induce a proper node coloring. As a result, nodes are able to transmit with much higher probabilities p_2 in the second part of the algorithm. Every round is of length d_2.

At the beginning of ColorReduction, a node initializes its color set F_v as $F_v \leftarrow [\Delta]$ so that the resulting coloring is of size $\Delta + 1$. In its active round a node chooses a random color from that set and transmits its final color with probability p_2. During all the other time intervals, v only listens for color messages and removes the received colors from its color set F_v accordingly. Since a proper $\Delta + 1$ coloring is always possible, even the nodes that are only allowed to act at the very end will still have at least one color left that they can draw from their color set.

In order to understand the interplay of transmitting probabilities and durations, we introduce the notion of a local broadcasting round. Since we want to restrict the total number of concurrently transmitting nodes due to the SINR constraint, nodes have to transmit with a low *transmission probability* p in every time slot. Assume every node is transmitting with the same transmission probability p. After a certain period of time d, that we call the *transmission duration*, all messages will have arrived successfully at their respective recipient. We call the act of transmitting a single message with probability p for d time slots (the simulation of) a *local broadcasting round*. A very simple way of translating any algorithm that relies on reliable communication, for example any message-passing algorithm, to the SINR model is to communicate only via local broadcasting rounds. This has a big disadvantage though, since local broadcasting rounds always account for the worst case. On average, a message arrives successfully after a shorter period of time. The sender, however, does not know whether its message has already been received and has to assume the worst case scenario. The goal is then to search for the optimal transmission probability

that leads to the lowest transmission duration with respect to a local broadcasting round. If the transmission probability is too low, we waste bandwidth because the medium is not saturated. On the other hand, if the transmission probability is too high, too many messages have to be dropped due to high interference.

Rand4DeltaColoring does not use entire local broadcasting rounds in order to communicate color choices. We are still interested in the optimal transmission probability p_1 that takes advantage of the full potential of the medium without causing too much interference. In order to determine this optimal transmission probability we set up Experiment 3.3.

Experiment 3.3: Local Broadcasting

Input	: Set of transmission probability values p defined by a lower bound p^{start}, an upper bound p^{end} and an increment p^{step}, number of iterations j
Output	: A mapping of transmitting probabilities p to corresponding transmission durations d
Environment	: A unit disk graph $G = (V, E)$ of $\|V\| = n$ nodes on an area A with a broadcasting range r_B and SINR parameters α, β and N

Let max() be the 99.9th percentile of a set.
For every input probability value we run j iterations of the experiment. Let p be the currently tested probability value and let i be the current test iteration. Every node is transmitting its ID in every time slot with probability p. For a node v we note the number of time slots $d_{v,i,p}$ it takes until the node's ID has been successfully received by all of its neighbors. At the end of test run i we note the maximum duration over all nodes $\max_{i,p} \leftarrow \max(d_{v,i,p} | v \in V)$ for probability p. The result of the experiment is a mapping of each transmission probability p to the maximum over all corresponding test runs $\max_p \leftarrow \max(\max_{i,p} | i \in [1, j])$.

We run Experiment 3.3 adopting the probabilities given in Table 3.1 with 1000 iterations each and note the 10 lowest resulting transmission durations as well as the corresponding transmitting probabilities. The best results are shown in Table 3.2, the top 10 results are shown in Table A.1 in the appendix. In order to get an overview of the interplay of n and the local broadcasting behaviour, refer to Figure 3.1 and Figure 3.2. We can see that determining an optimal transmission probability is more important in the denser graphs. For $n = 1500$, if we pick a probability value that is only slightly too high or low, transmission durations increase dramatically. This aligns well with our intuition. Independently from the number of nodes, if p approaches zero, the transmission duration tends towards infinity because if nodes are transmitting very seldomly, nothing happens. On the other side, since interference is accumulative, more nodes correspond to more interference and therefore lower optimal transmitting probabilities. This factor pushes the range of acceptable transmitting probabilities closer towards zero.

Another expected result of the local broadcasting experiment is that a local broadcasting round takes considerably less time in scenarios where $\alpha = 6$. A stronger signal decay means less interference from nodes farther away.

We now run Rand4DeltaColoring adopting the top ten probability values obtained from the local broadcasting experiment (Table A.1). The results are shown in Table 3.3. We set p_1 as the top probability value from the Rand4DeltaColoring experiment and set d_1 as the corresponding maximum runtime. The final values for p_1 and d_1 are summarized in Table 3.5.

We notice that Rand4DeltaColoring takes considerably less time than the minimum transmission duration obtained from the local broadcasting experiment. This speedup is

due to the randomized manner in which nodes choose their colors in Rand4DeltaColoring. It is not necessary that every node transmits its color successfully to every neighbor.

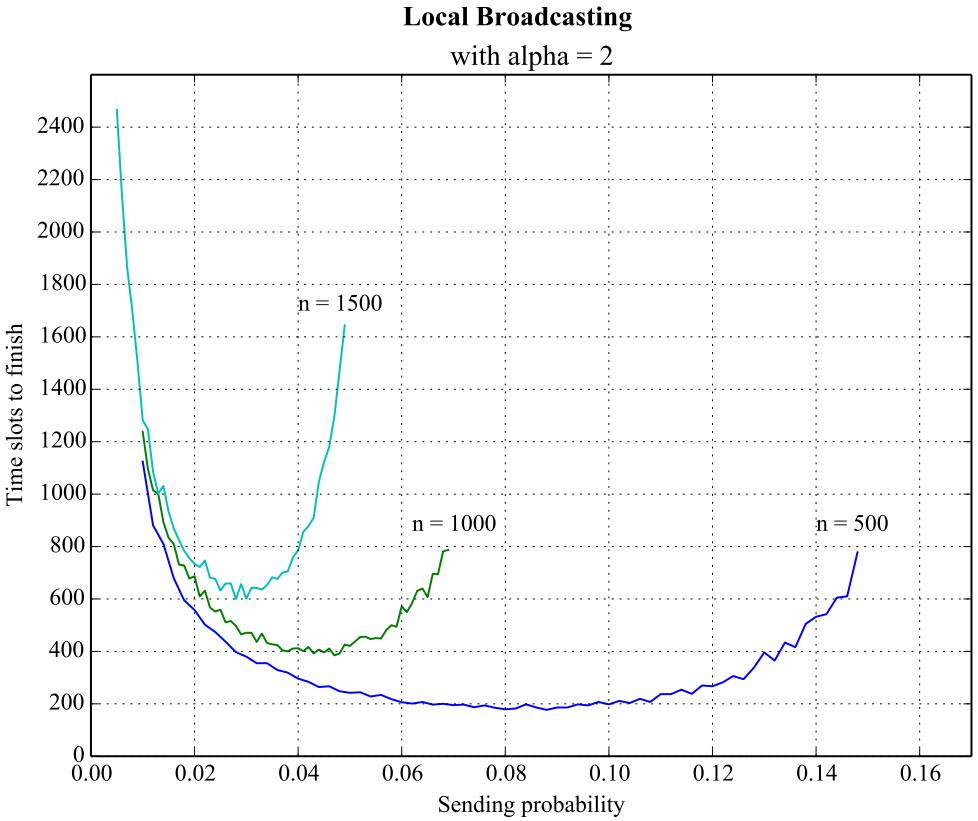


Figure 3.1.: Comparison of local broadcasting results with $n \in \{500, 1000, 1500\}$ and $\alpha = 2$

Table 3.1.: Parameter ranges for Local Broadcasting

Scenario	1	2	3	4	5	6
n	500	500	1000	1000	1500	1500
α	2	6	2	6	2	6
p_{start}	0.010	0.02	0.010	0.01	0.005	0.01
p_{end}	0.150	0.35	0.070	0.30	0.050	0.20
p_{step}	0.002	0.01	0.001	0.01	0.001	0.01
Iterations	1000	1000	1000	1000	1000	1000

Table 3.2.: Best results of Local Broadcasting

Scenario	1	2	3	4	5	6
n	500	500	1000	1000	1500	1500
α	2	6	2	6	2	6
p	0.088	0.14	0.047	0.1	0.028	0.08
d	177.0	126.0	385.0	245.0	600.0	361.0

For ColorReduction we need two more parameters p_2 and d_2. Unlike in the first part of Simple Coloring the relationship of p_2 and d_2 really is that of a local broadcasting round. ColorReduction works on the foundation of rounds. Each round consists of d_2 time slots. A node is only allowed to transmit in the single round associated with the color it obtained

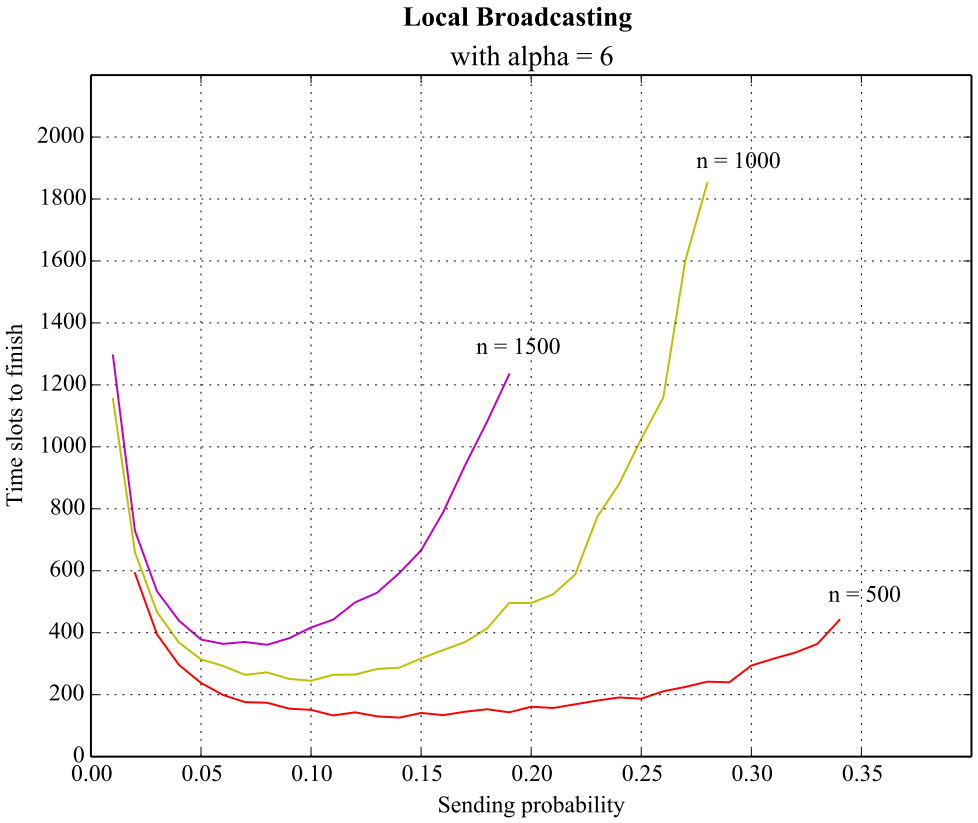


Figure 3.2.: Comparison of local broadcasting results with $n \in \{500, 1000, 1500\}$ and $\alpha = 6$

Table 3.3.: Results of Rand4DeltaColoring

Scenario	1	2	3	4	5	6
n	500	500	1000	1000	1500	1500
α	2	6	2	6	2	6
p_1	0.086	0.150	0.048	0.130	0.032	0.100
Runtime	72.0	41.0	142.0	72.0	230.0	106.0

Table 3.4.: Parameter ranges for Color Reduction

Scenario	1	2	3	4	5	6
n	500	500	1000	1000	1500	1500
α	2	6	2	6	2	6
p_2^{start}	0.30	0.30	0.30	0.30	0.30	0.30
p_2^{end}	0.70	0.70	0.80	0.80	0.70	0.70
p_2^{step}	0.01	0.01	0.01	0.01	0.04	0.04
d_2^{start}	6	2	2	2	14	14
d_2^{end}	30	30	30	30	24	24
d_2^{step}	2	2	2	2	2	2

in Rand4DeltaColoring. Furthermore the set of colors is now only of size $\Delta + 1$. It is therefore urgent that every message arrives successfully. The coloring obtained in the first part helps to reduce the transmission duration d_2. In every round, only one out of $4\Delta + 1$

Table 3.5.: Results of Simple Coloring with Color Reduction

Scenario	1	2	3	4	5	6
n	500	500	1000	1000	1500	1500
α	2	6	2	6	2	6
p_1	0.086	0.150	0.048	0.130	0.032	0.100
d_1	72.0	41.0	142.0	72.0	230.0	106.0
p_2	0.56	0.63	0.61	0.56	0.50	0.58
d_2	14	14	16	16	20	20
Average runtime	561.3 $\pm$59.6	534.3 $\pm$62.6	1046.0 $\pm$84.8	947.94 $\pm$78.5	1690.0 $\pm$106.3	1577.8 $\pm$130.6

nodes is transmitting. Furthermore, no two neighboring nodes are transmitting in the same round. Nodes can therefore adopt higher transmitting probabilities p_2.

We determine the values for p_2 and d_2 by adopting the values given in Table 3.4 and evaluating the resulting overall runtimes for Simple Coloring. If the duration d_2 is too low for the current value of p_2, the algorithm might fail. We disregard all combinations with a failure rate of more than 1%. The remaining combinations are sorted by the total runtime they yield. The best results are summarized in Table 3.5. The runtimes in this table are also the final results of our evaluation of Simple Coloring. In order to get a feel for these results, we again adduct the results from the local broadcasting experiment given in Table 3.2. If we look at Scenario 1 for example, we notice that the runtime of Simple Coloring is only about three times as high as the duration of a local broadcasting round. We could therefore intuitively say that Simple Coloring takes only three successful message transmissions between each pair of sender and receiver in order to set up a proper node coloring.

Simple Coloring has a very practical feature. Since every node knows about the total number of nodes in the graph n and the maximum degree Δ, it can deduce the runtime of the coloring algorithm a priori. Nodes can deduce p_1, d_1, p_2 and d_2 from n and the total runtime is always $d_1 + (4\Delta + 1)d_2$.

Simple Coloring is already as simple as possible, so the only room for improvements is in fine-tuning the parameters. In our implementation, we see that the ColorReduction part takes considerably more time than the Rand4DeltaColoring part. One approach for further improvement could be to balance this unevenness. In Rand4DeltaColoring, instead of calculating a 4Δ-coloring, we could calculate a $\lfloor c\Delta \rfloor$-coloring for any $c \in [1..4]$. Of course, this would take longer, but in consequence, ColorReduction would take less time, hopefully in such a way that overall runtime is reduced. Since Simple Coloring is very fast already, we do not evaluate such improvements in this thesis.

4. Moscibroda-Wattenhofer Coloring

MW Coloring is based on a coloring algorithm by Moscibroda and Wattenhofer ([MW08]) which is set in the so called graph-based model. In [DT10] the algorithm is transferred to the SINR model by Derbel and Talbi. In the SINR model the algorithm has an asymptotical upper runtime bound of $\mathcal{O}(\Delta \log n)$ and is therefore at par with Simple Coloring. The number of assigned colors however is only asymptotically bounded by $\mathcal{O}(\Delta)$ in contrast to Simple Coloring which always produces a $(\Delta + 1)$-coloring.

At the beginning of MW Coloring a maximal independent set is being constructed. All nodes compete for being in the MIS by means of a counting mechanism. Every node holds an integer counter that is incremented or reset according to a set of rules. A node whose counter exceeds a certain threshold wins the counting competition and enters the MIS. Nodes that are in the MIS are called *leaders* and nodes that are not in the MIS are called *background nodes*. If a node v receives a message from a leader node w it becomes a background node and bookmarks w as its assigned leader $L(v) \leftarrow w$. Background nodes ask their respective leaders for a color. A leader distributes ranges of colors to its requesting neighbors in an ascending manner. Since a background node may have a neighboring background node that has been assigned the same color range, background nodes must now compete for colors inside their assigned ranges. This competition works in the same way as the initial competition for being in the MIS.

There are four different messages that are being exchanged. $M_A^i(w, c_w)$ is telling the receiver that the sender w is competing for color i and the counter value of w is currently c_w. $M_C^i(w)$ is telling the receiver that the sender w has decided on a final color i. A spin-off of that type of message is $M_C^0(v, w, t_c)$ which is sent by a leader v to a background node w as an answer to a color request where t_c is the assigned color. $M_R(v, w)$ is a color request message from the background node v to its leader w.

Each node can be in three state classes A_i (*Competing*), R (*Requesting*) or C_i (*Colored* and in the case of $i = 0$ also *Leading*). State A_0 is special in that different constants are used than in other A_i states. State C_0 is special in that it is only entered by leaders and thus holds the leader code.

MW Coloring depends on four parameters p_{leader}, $p_{\text{background}}$, d_{leader} and $d_{\text{background}}$. In Section 4.2 we explain how to find a parameter combination that leads to the best overall performance of this algorithm.

State A_0 is the state in which all nodes wake up in the beginning and compete for leadership. The pseudo-code for A_i is shown in Algorithm 4.2. Since in our environment, all nodes

wake up at the same time, every node skips the listening part (Line 7) and starts with the counting part (Line 16) right away. During the listening part a node does not transmit, which reduces interference. Nodes only listen for counter updates or color messages. Note that nodes do not skip the listening part in $A_{i>0}$. After listening for $d_{\text{background}}$ time slots a node is competing for color i and therefore in the case of $i = 0$ for leadership. The counting mechanism is the heart of MW Coloring. It works on the foundation of local broadcasting rounds. At the beginning of the counting procedure, a node v chooses $c_v = 0$ (Line 17). In every following time slot, this counter is incremented by one (Line 19). The goal is to reach the threshold of $d_{\text{background}}$ which means transitioning to a *Colored* state C_i (Line 23). Node v tracks the counters of all neighboring nodes. Let w be a neighbor of v, then $d_v(w)$ denotes the counter value that v currently knows of w. The competitors' counters are also incremented in every time slot (Line 21). Node v transmits its counter with probability $p_{\text{background}}$ in every time slot (Line 24). If v receives a counter value c_w from w, it updates its $d_v(w)$ record and checks if the counter conflicts with its own counter c_v (Line 32). A conflict occurs if c_w is too close to c_v: $|c_v - c_w| \leq d_{\text{leader}}$. In the case of a conflict, v has to choose a new counter value which has to be negative and must not conflict with any neighbors' counter values it knows of. This way it is guaranteed that counter values are always at least d_{leader} steps apart from each other. That means if a node v successfully hurdles the threshold $d_{\text{background}}$ it takes the fastest neighbor at least d_{leader} time slots to reach the threshold as well. Before that can happen, however, node v successfully transmits a $M_C^i(w)$ message that tells a competing neighbor w that v won the competition thus forcing w to acknowledge its defeat and enter state R in the case of $i = 0$ or state A_{i+1} in the case of $i > 0$ (Line 27).

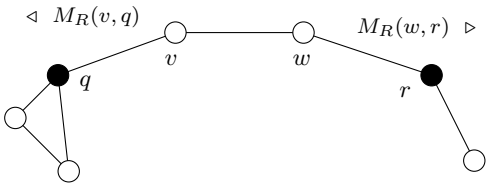
Leaders transition to state C_0 while background nodes transition to R (Algorithm 4.4) where they request a color range from their respective leaders. As soon as they have successfully obtained a color range, they transition to $A_{i>0}$ where i defines their assigned range. In $A_{i>0}$ background nodes are competing for a color within their range and transition to C_i as soon as they decide on a final color.

Background nodes in C_i (Algorithm 4.3) only transmit their final color for the remainder of the algorithm. Leaders in C_0 add requesting background nodes to a queue $\mathcal{Q}$ (Line 10). If said queue is empty, a leader v advertises its leadership via a $M_C^0(v)$ message (Line 12). Otherwise it dequeues a node w from $\mathcal{Q}$ and assigns this node a color range defined by t_c via a $M_C^0(v, w, t_c)$ message (Line 17). By incrementing t_c (Line 14) it is guaranteed that every requesting node is assigned a different color range. The process of color requests and responses is illustrated in Figure 4.1.

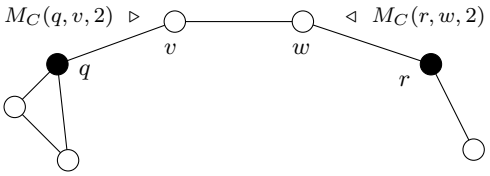
Since we are not interested in proving asymptotical runtime bounds in this thesis, we reduce the number of constants to four. These constants are $d_{\text{background}}$, d_{leader}, $p_{\text{background}}$ and p_{leader}. The naming of the four parameters already suggests the general idea. Leaders and background nodes transmit with different probabilities which leads to different transmission durations. We might want $d_{\text{leader}}$to be lower than $d_{\text{background}}$ because multiple background nodes in state R are waiting for a single leader to assign colors to them. The total runtime of the algorithm might therefore depend more on the efficient communcation from leaders to background nodes than the other way around. [DT10] suggests a ratio of Δ, i.e. $p_{\text{leader}} = \Delta p_{\text{background}}$ and $d_{\text{leader}} = \frac{1}{\Delta} d_{\text{background}}$. In practice this theoretical ratio is too high. We set up a separate experiment in order to determine optimal values for $d_{\text{background}}$, d_{leader}, $p_{\text{background}}$ and p_{leader}.

4.1. MIS Broadcasting

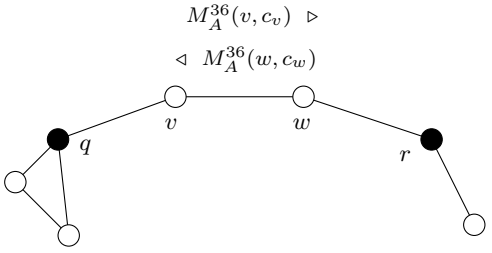
We set up Experiment 4.1 (MIS Broadcasting) that is quite similar to the local broadcasting experiment. At the beginning of the experiment, a maximal independent set



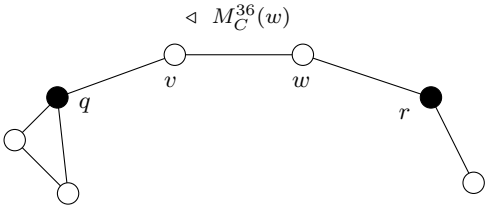
(a) Nodes v and w lose the competition for leadership. Node q is the leader of v and r is the leader of w. Both v and w request a color range from their respective leaders.



(b) Both leaders assign the color range $[2 \cdot 18, 3 \cdot 18 - 1] = [36, 53]$ defined by $t_c = 2$ to their respective requesters.



(c) Both v and w compete for the first color in range 2, which is $2 \cdot 18 = 36$.



(d) Node w wins the competition, chooses 36 as its final color and transmits a Color message to v, which has to proceed to the next color 37 and compete again.

Figure 4.1.: The process of color requests, leader responses and competition

is computed greedily. Again, all nodes transmit their IDs to their neighbors. In contrast to Local Broadcasting, nodes in the MIS adopt a different transmission probability p_{leader} than all other nodes which adopt $p_{\text{background}}$. We pick a random leader node and a random background node and note the number of time slots it takes for them to successfully broadcast their IDs. We run the experiment adopting the probability values given in Table 4.1 for 500 iterations each. In contrast to Simple Coloring, in MW Coloring we are interested both in the maximum transmission duration and the average transmission duration. Therefore Experiment 4.1 gives us a mapping of $(p_{\text{leader}}, p_{\text{background}}) \to (d_{\text{leader}}, d_{\text{background}}, d^{\text{avg}}_{\text{leader}}, d^{\text{avg}}_{\text{background}})$.

4.2. Analyzing the flow of MW Coloring

In order to pick the optimal combination of transmitting probabilities, we have to weight the 4-tuples obtained from MIS Broadcasting according to the influence these values have on the total runtime of MW Coloring. For example, if we determine that $d_{\text{background}}$ has four times as much influence on the total runtime as d_{leader} we want to weight $d_{\text{background}}$ four times as high. We therefore analyze the flow of sample runs of MW Coloring. An overview of our analysis is shown in Figure 4.2 and Figure 4.3.

The total runtime of MW Coloring depends directly on the time it takes the last background node to decide on its final color. Since the runtime of the slowest background node depends on all nodes in its vicinity which again depend on their respective neighbors, we have to consider the runtime of every node in the graph. It simplifies our analysis drastically if we

Experiment 4.1: MIS Broadcasting

Input : Set of transmission probability values p_{leader}defined by a lower bound $p^{\text{start}}_{\text{leader}}$, an upper bound $p^{\text{end}}_{\text{leader}}$ and an increment $p^{\text{step}}_{\text{leader}}$ and a set of transmission probability values $p_{\text{background}}$defined by a lower bound $p^{\text{start}}_{\text{background}}$, an upper bound $p^{\text{end}}_{\text{background}}$ and an increment $p^{\text{step}}_{\text{background}}$, number of iterations j

Output : A mapping of transmission probability tuples $(p_{\text{leader}}, p_{\text{background}})$ to corresponding maximum and average transmission duration tuples $(d_{\text{leader}}, d_{\text{background}}, d^{\text{avg}}_{\text{leader}}, d^{\text{avg}}_{\text{background}})$

Environment : A unit disk graph of n nodes on an area A with a broadcasting range r_B and SINR parameters α, β and N

At the beginning of this experiment, a MIS is calculated using a greedy algorithm. Nodes inside the MIS adopt p_{leader} as transmission probability, nodes not in the MIS adopt $p_{\text{background}}$.
For every combination of input probabilities we run j iterations of the experiment. Let i be the current iteration.
Every node is broadcasting its ID in every time slot with its assigned probability. We randomly choose a leader v and a background node w to watch. We measure the number of time slots it takes until all neighbors of v have received v's ID successfully. We also measure the number of time slots it takes until all neighbors of w have received w's ID successfully.
By running multiple iterations of this experiment, we obtain a maximum as well as an average transmission duration for leaders and background nodes, which we enter into the output tuple (d_{leader}, $d_{\text{background}}$, $d^{\text{avg}}_{\text{leader}}$, $d^{\text{avg}}_{\text{background}}$).

Table 4.1.: Parameter ranges for MIS Broadcasting

Scenario	1	2	3	4	5	6
n	500	500	1000	1000	1500	1500
α	2	6	2	6	2	6
$p^{\text{start}}_{\text{leader}}$	0.040	0.080	0.020	0.020	0.010	0.040
$p^{\text{end}}_{\text{leader}}$	0.140	0.240	0.100	0.100	0.060	0.200
$p^{\text{step}}_{\text{leader}}$	0.010	0.010	0.005	0.005	0.005	0.010
$p^{\text{start}}_{\text{background}}$	0.040	0.080	0.020	0.020	0.010	0.040
$p^{\text{end}}_{\text{background}}$	0.140	0.240	0.100	0.100	0.050	0.200
$p^{\text{step}}_{\text{background}}$	0.010	0.010	0.010	0.010	0.005	0.010
Iterations	500	500	500	500	500	500

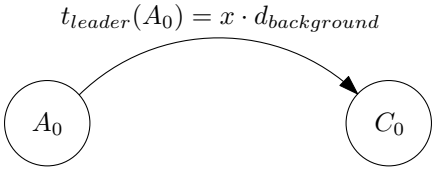


Figure 4.2.: Analysis of leaders in MW Coloring

look at the control flow of an *average node* starting at time $t = 0$ and going on until the node decides on a final color.

Let $t_{\text{leader}}(S)$ be the time an average leader spends in state S. Let $t_{\text{background}}(S)$ be the time an average background node spends in state S. For all combinations of p_{leader}, $p_{\text{background}}$,

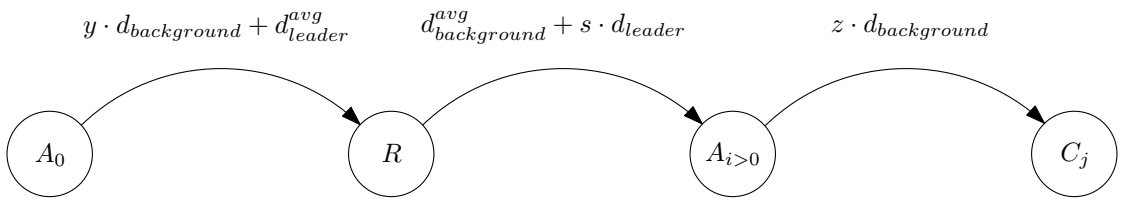


Figure 4.3.: Analysis of background nodes in MW Coloring

d_{leader}, $d_{\text{background}}$, $d^{\text{avg}}_{\text{leader}}$ and $d^{\text{avg}}_{\text{background}}$, that we used in MIS Broadcasting, we run four iterations of MW Coloring adopting these parameters. Every node logs the time it spends in each state it is going through. We therefore learn the values for $t_{\text{leader}}(A_0)$, $t_{\text{background}}(A_0)$, $t_{\text{background}}(R)$ and $t_{\text{background}}(A_{i>0})$ by examining the log of MW Coloring. Please note that the value for $t_{\text{background}}(A_{i>0})$ is a sum over all $i > 0$ as depicted in Figure 4.3.

We now have to decide how exactly d_{leader}, $d_{\text{background}}$, $d^{\text{avg}}_{\text{leader}}$ and $d^{\text{avg}}_{\text{background}}$ contribute to $t_{\text{leader}}(A_0)$, $t_{\text{background}}(A_0)$, $t_{\text{background}}(R)$ and $t_{\text{background}}(A_{i>0})$. In order to do this we analyze leaders and background nodes separately. In the following paragraphs, let l be the number of leaders and let b be the number of background nodes in a greedily calculated MIS for each testing scenario.

The analysis for leader nodes is simple. An overview is given in Figure 4.2. Leaders only perform a single transition from A_0 to C_0. They decide on their final color right after they transition to C_0. The time it takes leaders to leave state A_0 only depends on $d_{\text{background}}$ since this is the threshold they have to reach. For an average leader node it takes $t_{\text{leader}}(A_0) = x \cdot d_{\text{background}}$ time slots in order to reach this threshold.

The analysis for background nodes is more involved. An overview is given in Figure 4.3. In state A_0, nodes that end up as background nodes lose the competition for leadership. They get informed about their defeat by receiving a message from a neighboring leader. That message takes $d^{\text{avg}}_{\text{leader}}$ time slots to be delivered. The remainder of the time is therefore spent counting and listening which both counts towards $d_{\text{background}}$. For an average background node it therefore takes $t_{\text{background}}(A_0) = y \cdot d_{\text{background}} + d^{\text{avg}}_{\text{leader}}$ in order to leave state A_0 and enter state R.

When a node is in state R it transmits a request to its leader which takes an average time period of $d^{\text{avg}}_{\text{background}}$. If the total number of leaders is l and the total number of background nodes is b then an average leader has to serve b/l background nodes. The average time it takes a leader to serve a background node in state R is $s \cdot d_{\text{leader}}$ where the factor s can also be calculated as follows.

$$s = \frac{1 + 2 + \cdots + b/l}{b/l} = \frac{\frac{(b/l)^2 + (b/l)}{2}}{b/l} = \frac{(b/l + 1)}{2}$$

For an average background node it therefore takes $t_{\text{background}}(R) = d^{\text{avg}}_{\text{background}} + s \cdot d_{\text{leader}}$ in order to leave state R.

After a background node has been assigned a color range it transitions to a state A_i with $i > 0$. For the remainder of the algorithm background nodes only work with $d_{\text{background}}$ so all the time spent in $A_{i>0}$ counts towards $d_{\text{background}}$. An average background node spends $t_{\text{background}}(A_{i>0}) = z \cdot d_{\text{background}}$ time slots in $A_{i>0}$.

We rearrange the obtained equations and get the values for x, y, s and z.

$$
\begin{aligned}
x =& \frac{t_{\text{leader}}(A_0)}{d_{\text{background}}} \\
y =& \frac{t_{\text{background}}(A_0) - d^{\text{avg}}_{\text{leader}}}{d_{\text{background}}} \\
s =& \frac{t_{\text{background}}(R) - d^{\text{avg}}_{\text{background}}}{d_{\text{leader}}} \\
z =& \frac{t_{\text{background}}(A_{i>0})}{d_{\text{background}}}
\end{aligned}
$$

The weighting we get from x, y, s and z is then as follows.

$$
\begin{aligned}
w_{d_{\text{background}}} =& \frac{l}{l+b} \cdot x + \frac{b}{l+b} \cdot y + \frac{b}{l+b} \cdot z \\
w_{d_{\text{leader}}} =& \frac{b}{l+b} \cdot s \\
w_{d^{\text{avg}}_{\text{background}}} =& \frac{b}{l+b} \cdot 1 \\
w_{d^{\text{avg}}_{\text{leader}}} =& \frac{b}{l+b} \cdot 1
\end{aligned}
$$

We calculate these values for every run of MW Coloring adopting the parameter combinations obtained from MIS Broadcasting. An extract of these results for Scenario 3 can be seen in Table 4.2. By looking at this extract, we already notice a trend. The weighting seems to depend very little on the transmission probability and duration values. In Table 4.3 we list the normalized average weightings as well as the standard deviation over all samples. Keep in mind that the values in Table 4.2 are averaged over only four runs each.

Now we have two possibilities to weight the tuples from MIS Broadcasting. We can either use the individual weightings as shown in the extract in Table 4.2 or we can use the averaged weightings shown in Table 4.3. We sort the combinations according to the individual weightings. We take the top five combinations of the individually ranked combinations and perform 100 iterations of the MW algorithm with each of these. We now sort the combinations according to the average weighting. We also take the top five of the average ranked combinations and perform 100 iterations of the MW algorithm with each of these as well. The best results for each testing scenario are shown in 4.4. For every testing scenario, the best individually ranked parameter combination also yields the best overall runtime when applied to MW Coloring. For every testing scenario with the exception of Scenario 2, these combinations are also the best average ranked combinations. In the case of Scenario 2, the best individually ranked combination is the third-best average ranked combination.

4.3. Practical improvements

The original algorithm is described in a way that makes it easy to prove its asymptotical upper bound. There are some improvements that do not improve on any theoretical bounds but yield a better runtime in our testing scenarios.

First of all, we abandon the listening phase in state A_i. Nodes waste a lot of time listening for conflicting color messages without much benefit.

Table 4.2.: Extract of results for MW weighting in testing scenario 3

p_{leader}	p_{backg}	d_{leader}	d_{backg}	$d^{\text{avg}}_{\text{leader}}$	$d^{\text{avg}}_{\text{backg}}$	$w_{d_{\text{background}}}$	$w_{d_{\text{leader}}}$	$w_{d^{\text{avg}}_{\text{background}}}$	$w_{d^{\text{avg}}_{\text{leader}}}$
0.050	0.02	222	646	25	64	1.114	3.807	0.711	0.711
0.050	0.03	255	449	26	46	1.200	4.194	0.716	0.716
0.050	0.04	272	368	28	38	1.214	4.336	0.709	0.709
0.050	0.05	540	523	31	35	1.211	4.006	0.707	0.707
0.050	0.06	459	519	35	33	1.208	4.451	0.706	0.706
0.050	0.07	836	584	41	35	1.232	4.610	0.706	0.706
0.050	0.08	930	1033	51	39	1.261	4.520	0.707	0.707
0.050	0.09	1746	1457	68	47	1.241	4.210	0.703	0.703
0.055	0.02	253	692	23	64	1.143	3.822	0.712	0.712
0.055	0.03	305	466	24	47	1.118	4.201	0.707	0.707
0.055	0.04	247	405	26	39	1.214	4.151	0.710	0.710
0.055	0.05	338	440	29	35	1.199	4.255	0.710	0.710
0.055	0.06	389	654	33	34	1.237	4.246	0.710	0.710
0.055	0.07	931	929	39	36	1.201	4.325	0.704	0.704
0.055	0.08	1851	1347	50	41	1.202	4.505	0.702	0.702

Table 4.3.: Results of the MW analysis

Scenario	1	2	3	4	5	6
n	500	500	1000	1000	1500	1500
α	2	6	2	6	2	6
$w_{d_{\text{leader}}}$	0.111 ±0.008	0.108 ±0.007	0.178 ±0.01	0.175 ±0.011	0.224 ±0.013	0.214 ±0.014
$w_{d_{\text{background}}}$	0.645 ±0.052	0.655 ±0.053	0.615 ±0.035	0.620 ±0.046	0.596 ±0.039	0.611 ±0.043
$w_{d^{\text{avg}}_{\text{leader}}}$	0.122 ±0.001	0.118 ±0.001	0.103 ±0.001	0.102 ±0.001	0.09 ±0.0004	0.087 ±0.0005
$w_{d^{\text{avg}}_{\text{background}}}$	0.122 ±0.001	0.118 ±0.001	0.103 ±0.001	0.102 ±0.001	0.09 ±0.0004	0.087 ±0.0005

Inspired by the heavy usage of randomization in Simple Coloring we introduce *random color range allocation* for MW Coloring. In the leader code of Algorithm 4.3, instead of assigning color ranges in an ascending manner, we randomize these decisions by picking t_c from the interval $[0, \Delta]$. We have to keep track of the color ranges we assigned already, so we introduce another integer set G_v for every leader v. G_v is initialized as $G_v \leftarrow [\Delta]$ and every time a leader dequeues a node from $\mathcal{Q}$, it randomly picks and removes a color from G_v. The effect of this improvement is the same as with the randomization in Simple Coloring. If all leaders distribute color ranges in an ascending manner, it is very likely that two neighboring background nodes are assigned the same color range and thus have to compete for a color inside that range. If the color range distribution is randomized however, less of these conflicts occur.

Another limitation that we want to overcome is that leaders have to transmit for the total duration of a local broadcasting round when they distribute color ranges. A background node may have received the message a long time ago, but the corresponding leader is still transmitting. After a background node v receives the message $M^0_C(L(v), v, t_c)$ it transitions to state $A_{t_c \cdot 18}$ where it transmits a $M^i_A(v, c_v)$ message. Leaders can safely stop transmitting as soon as they receive such a message. This way the next node in the leader's queue is served faster.

The improved leader code is shown in Algorithm 4.5.

Table 4.4.: Results of the evaluation of MW Coloring

Scenario	1	2	3	4	5	6
n	500	500	1000	1000	1500	1500
α	2	6	2	6	2	6
p_{leader}	0.060	0.150	0.050	0.080	0.050	0.070
$p_{\text{background}}$	0.110	0.160	0.040	0.100	0.025	0.080
d_{leader}	221	91	272	226	270	287
$d_{\text{background}}$	138	113	368	207	616	283
Average runtime	1909.7 ±239.8	1215.8 ±122.7	5589.7 ±533.7	3393.5 ±346.6	10567.4 ±812.3	6032.2 ±417.9

We implement all these features and improvements and rerun the experiment. The results are shown in Table 4.5. If we compare these results with the results of the unimproved version of MW Coloring in Table 4.4, we see that our improvements have a big impact on the average runtime of the algorithm. In every testing scenario we have a speedup of at least 20%. In Chapter 6 we compare these results in more detail.

Algorithm 4.2: MW COLORING for node v in state A_i

Data: Set of competing nodes P_v, dictionary $d_v(w)$ mapping counter values to nodes, dictionary $L(w)$ mapping leaders to nodes, own counter value c_v

```
// Initialization
P_v ← ∅
if i = 0 then
    d ← d_leader
    A_suc ← R
else
    d ← d_background
    A_suc ← A_{i+1}

// Listening
for d_background time slots do
    forall w ∈ P_v do
        d_v(w) ← d_v(w) + 1
    if M_A^i(w, c_w) received then
        P_v ← P_v ∪ {w}
        d_v(w) ← c_w
    if M_C^i(w) received then
        L(v) ← w
        transitionTo(A_suc)

// Counting
c_v ← χ(P_v), where χ(P_v) is the maximum value such that
χ(P_v) ∉ {d_v(w) − d, d_v(w) − d + 1, ..., d_v(w) + d − 1, d_v(w) + d} for each w ∈ P_v
and χ(P_v) ≤ 0
while state = A_i do
    c_v ← c_v + 1
    forall w ∈ P_v do
        d_v(w) ← d_v(w) + 1
    if c_v ≥ d_background then
        transitionTo(C_i)
    transmit M_A^i(v, c_v) with probability p_background
    if M_C^i(w) received then
        L(v) ← w
        transitionTo(A_suc)
    if M_A^i(w, c_w) received then
        P_v ← P_v ∪ {w}
        d_v(w) ← c_w
        if |c_v − c_w| ≤ d_leader then
            c_v ← χ(P_v)
```

Algorithm 4.3: MW Coloring for node v in state C_i

Data: Color of node v color_v, color counter t_c, queue of neighboring nodes $\mathcal{Q}$

```
// Initialization
color_v ← i
t_c ← 0
Q ← ∅
// Code
if i > 0 then
    for the remainder of the algorithm do
        transmit M_C^i(v) with probability p_background
else
    for the remainder of the algorithm do
        if M_R(w, v) received and w ∉ Q then
            Q.ENQUEUE(w)
        if Q.EMPTY() then
            transmit M_C^0(v) with probability p_leader
        else
            t_c ← t_c + 1
            Let w ← Q.DEQUEUE()
            for d_leader time slots do
                transmit M_C^0(v, w, t_c) with probability p_leader
```

Algorithm 4.4: MW Coloring for node v in state R

Data: Color of node v color_v, color counter t_c, queue of neighboring nodes $\mathcal{Q}$

```
while in state R do
    transmit M_R^0(v, L(v)) with probability p_background
    if M_C^0(L(v), v, t_c) received then
        TRANSITIONTO(A_{t_c·18})
```

Table 4.5.: Results of the evaluation of improved MW Coloring

Scenario	1	2	3	4	5	6
n	500	500	1000	1000	1500	1500
α	2	6	2	6	2	6
p_{leader}	0.06	0.15	0.05	0.08	0.050	0.07
$p_{\text{background}}$	0.11	0.16	0.04	0.10	0.025	0.08
d_{leader}	221	91	272	226	270	287
$d_{\text{background}}$	138	113	368	207	616	283
Average runtime	1372.0 ±147.4	933.9 ±114.6	4774.7 ±382.5	2684.3 ±251.1	8321.5 ±835.4	4416.8 ±274.8

Algorithm 4.5: IMPROVED CODE FOR NODE v IN STATE C_i

Data: Color of node v color_v, queue of neighboring nodes $\mathcal{Q}$, set of assigned colors G_v

```
// Initialization
color_v ← i
G_v ← [Δ]
Q ← ∅
// Code
if i > 0 then
    for the remainder of the algorithm do
        transmit M_C^i(v) with probability p_background
else
    for the remainder of the algorithm do
        if M_R(w, v) received and w ∉ Q then
            Q.ENQUEUE(w)
        if Q.EMPTY() then
            transmit M_C^0(v) with probability p_leader
        else
            Let t_c ← G_v.GETRANDOMANDREMOVE()
            Let w ← Q.DEQUEUE()
            for d_leader time slots do
                transmit M_C^0(v, w, t_c) with probability p_leader
                if M_A^i(w, c_w) received then
                    break out of for loop
```

5. Yu et al. Coloring

This third algorithm as described by Yu, Wang, Hua and Lau in [YWHL14] is similar to the MW-based algorithm of the previous section. The basic idea remains that neighbors compete for colors by incrementing a counter until they reach a threshold or receive a conflicting counter from a neighboring competitor. Yu Coloring has another similarity with MW Coloring in that nodes compete to get into a maximal independent set. We thus borrow the terms from MW and say that nodes in the MIS are called leaders and all other nodes are called background nodes.

The original version of Yu Coloring as described in [YWHL14] is designed to work without the knowledge of Δ. The algorithm uses a slow start mechanism in order to determine optimal transmission probabilities dynamically during runtime. A node v starts with an extremely low transmission probability p_v and doubles that value after a certain period of time. Node v may also halve the transmission probability if it comes to its knowledge that its neighborhood's additive transmission probability is too high. Due to this slow start mechanism, Yu Coloring has an asymptotical runtime bound of $\mathcal{O}(\Delta \log n + \log^2 n)$.

With the slow start mechanism, Yu Coloring has seven parameters that could only be determined by running the algorithm with all possible combinations of those parameters. Even if we constrained the number of different values for each parameter to ten, we would have to run Yu Coloring with $10^7 = 10,000,000$ combinations, which is not feasible. Furthermore, in our testing environment, algorithms know n and Δ and are therefore able to deduce optimal transmission probabilities right from the start. In consequence, we simplify Yu Coloring and abandon the slow start mechanism. Keep in mind that we cannot prove that our modified implementation performs better than the original algorithm in all cases.

In contrast to MW Coloring, Yu Coloring calculates a MIS in terms of $3r_B$. This is done by setting the broadcasting range to $r_B^{\text{hi}} = 3r_B$ and using an appropriately higher radio power of $P_{\text{hi}} = 3^\alpha P$ where α is the geometric decay in the SINR model. The nodes that currently use the higher radio power and broadcasting range are called *strong nodes*, the others are called *weak nodes*. In order to avoid ambiguity, we use $r_B^{\text{lo}} = r_B$ for the low broadcasting range and $r_B^{\text{hi}} = 3r_B$ for the high broadcasting range in this chapter. We also use $P_{\text{hi}} = 3^\alpha P$ for the high transmission power and $P_{\text{lo}} = P$ for the low transmission power. Yu Coloring depends on four parameters p_{hi}, p_{lo}, d_{hi} and d_{lo}. In Section 5.1 we explain how to find a parameter combination that leads to the best overall performance of this algorithm.

Algorithm 5.1: Yu Coloring state B for node v

Data: Counter c_v, integer step value step

```
1  step ← step + 1
2  if step ≥ d_hi then
3  |  c_v ← c_v + 1
4  |  transmit m_B(c_v) with probability p_hi and power P_hi
5  if c_v ≥ d_hi then
6  |  TRANSITIONTO(M)
7  if m_B(c_w) received then
8  |  if |c_v − c_w| ≤ d_hi then
9  |  |  c_v ← 0
10 |  |  step ← 0
11 if DoNotTransmit_u received then
12 |  F_v.ADD(u)
13 |  TRANSITIONTO(S)
```

Algorithm 5.2: Yu Coloring state M for node v

Data: Integer step value step

```
1  step ← step + 1
2  if step < d_hi then
3  |  transmit DoNotTransmit_v with probability p_hi and power P_hi
4  else
5  |  TRANSITIONTO(G)
```

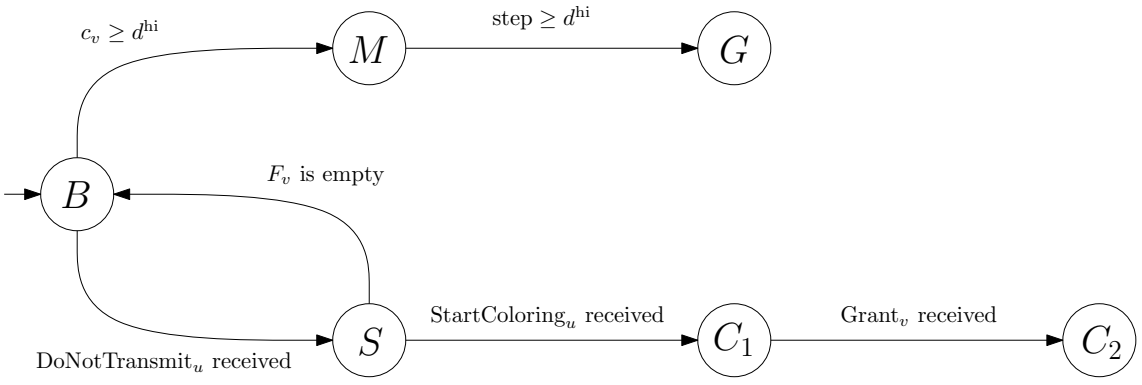


Figure 5.1.: State machine for Yu Coloring. Nodes that win the competition for being in the MIS take the upper path, losing nodes go to state S and may later take the upper path or enter S again.

When a node wakes up in state B (Algortihm 5.1), it immediately begins to execute the MIS algorithm adopting the high transmission power P_{hi} along with the corresponding broadcasting range of r_B^{hi}. A node v that gets in the MIS and thus becomes a leader, transitions to state M (Algorithm 5.2) and transmits a DoNotTransmit$_v$ message to all neighbors in terms of r_B^{hi}. Nodes that receive such a message transition to a passive state S (Algorithm 5.4). The DoNotTransmit$_v$ message works similar to locking a semaphore. Each node u in state S has a *forbidden set* F_u. If u receives a DoNotTransmit$_v$ message it adds v to its forbidden set. Later the leader v transmits a StartTransmit$_v$ message to u which is

Algorithm 5.3: Yu Coloring state G for node v

Data: Counter c_v, queue Q_v

```
1  color_v ← 0
2  for d_lo time slots do
3      transmit StartColoring_v with probability p_lo and power P_lo
4  if Q_v is not empty then
5      let u ← Q_v.DEQUEUE()
6      for d_lo time slots do
7          transmit Grant_u with probability p_lo and power P_lo
8          c_v ← c_v + 1
9  else
10     c_v ← c_v + 1
11 if Q_v is empty and c_v > d_lo then
12     for d_hi time slots do
13         transmit StartTransmit_v with probability p_hi and power P_hi
14 if AskColor_u received then
15     Q_v.ENQUEUE(u)
16     c_v ← 0
```

similar to unlocking the semaphore. Upon receiving such a message, node u removes v from its forbidden set. If thereupon the forbidden set is empty, u leaves the passive state S and begins to execute the MIS algorithm again by transitioning to state B.

After a leader v transmits the DoNotTransmit_v message, it transitions to state G (Algorithm 5.3) where it transmits a StartColoring_v message to all neighbors within broadcasting range r_B^{lo}. Upon receiving such a message, a node w in state S transitions to state C_1 (Algorithm 5.5). The background node w in state C_1 transmit a color request message AskColor_w to its leader. The leader v in state G accepts color requests from neighbors. Each request is added to a queue Q_v. The leader v then dequeues a node x and transmits a Grant_x message. Upon receiving such a Grant_x message, node x transitions to state C_2 (Algorithm 5.6). In C_2 it chooses an available color from its color set and transmits this choice to its neighbors. After leader v has served all neighboring background nodes,

Algorithm 5.4: Yu Coloring state S for node v

Data: Counter c_v, forbidden set F_v, color set $Color_v$ ...

```
1  if F_v is empty then
2      TRANSITIONTO(B)
3  if DoNotTransmit_u received then
4      F_v.ADD(u)
5  if StartTransmit_u received then
6      F_v.REMOVE(u)
7  if Color_u received then
8      Color_v.REMOVE(the color in the Color message)
9  if StartColoring_u received then
10     TRANSITIONTO(C_1)
```

Algorithm 5.5: Yu Coloring state C_1 for node v

Data: Counter c_v, forbidden set F_v, color set $Color_v$ ...

1 transmit AskColor$_v$ message with probability p_{lo} and power P_{lo}
2 **if** *Grant$_v$ received* **then**
3 TRANSITIONTO(C_2)
4 **if** *Color$_u$ received* **then**
5 *Color$_v$*.REMOVE(*the color in the Color message*)

Algorithm 5.6: Yu Coloring state C_2 for node v

Data: Counter c_v, forbidden set F_v, color set $Color_v$ ...

1 choose the first available color from the color list Color$_v$
2 **for** d_{lo}*time slots* **do**
3 transmit Color$_v$ message containing the chosen color with probability p_{lo} and power P_{lo}

it again adopts the higher radio power P_{hi} and broadcasting range r_B^{hi} and transmits a StartTransmit$_v$ message as explained already. This scenario is illustrated in Figure 5.2 and Figure 5.3. An overview of the state transitions is shown in Figure 5.1.

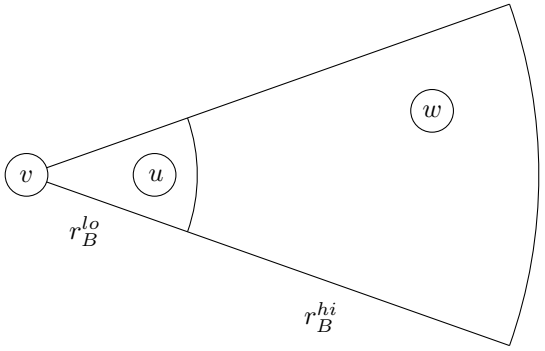


Figure 5.2.: Node u is a neighbor of v in terms of r_B^{hi} and r_B^{lo}, node w is only a neighbor in terms of r_B^{hi}.

As already stated, the competition for becoming a leader (state B, Algorithm 5.1) is based on a counting mechanism similar to that of MW Coloring. Every competing node v holds a counter c_v. After entering state B, v has to wait for d_{hi} time slots before beginning to increment this counter by one in each time slot (Line 3). While incrementing the counter, v transmits its counter to its neighbors (Line 4). If v receives a neighbor's counter value c_w it checks whether this counter conflicts with its own counter c_v. A conflict occurs if $|c_v - c_w| \leq d_{\text{hi}}$. In the case of a conflict, v has to reset its counter (Line 9) and step value (Line 10). By resetting its step value, it again has to wait for d_{hi} time slots before restarting to increment its counter. This way it is guaranteed that the counters of two neighboring nodes are always at least d_{hi} apart from each other. When the fastest node v reaches the threshold of d_{hi}, it takes the next node w at least d_{hi} time slots in order to reach the threshold as well. Before this happens, however, v successfully transmits a DoNotTransmit$_u$ message (Line 6) to all neighboring nodes including w, thus forcing w to quit the competition and enter state S (Line 13).

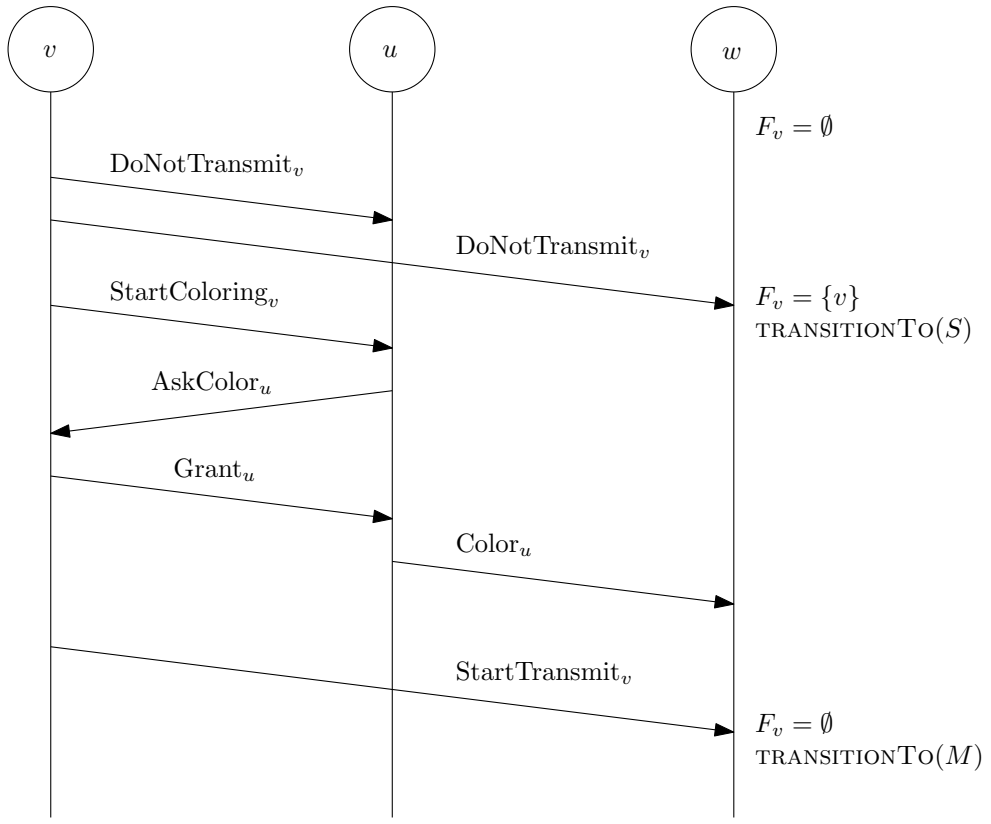


Figure 5.3.: A sample interaction between a leader v, an active background node u and a passive background node w

5.1. Determining optimal parameter combinations

In contrast to MW Coloring, Yu Coloring calculates a MIS in terms of $3r_B$. This has the effect that every background node inside the r_B-radius has exactly one neigboring leader. Once a background node has been assigned a color, it does not have to compete with other nodes anymore. This advantage comes at a price though. Erecting a MIS in terms of $3r_B$ takes significantly longer than erecting a standard MIS. Nodes have to adopt a higher radio power and accept messages from nodes that are farther away. By tripling the broadcasting radius, the broadcasting area becomes nine times as big. Every node therefore has nine times as many neighbors which increases the length of a local broadcasting round significantly.

Yu Coloring has one trait that makes it difficult to optimize for our testing scenarios. In Simple Coloring, we have two distinct and synchronous phases (Rand4DeltaColoring and ColorReduction). All nodes are in the same phase at all times. This makes it easy to determine optimal transmitting probabilities. In MW Coloring, a node is either a leader or a background node and that role does not change throughout the algorithm. Therefore we have no major problem to determine optimal transmitting probabilities for MW Coloring either. In Yu Coloring however, there are no synchronous phases and we can make no guesses about the number of leaders or background nodes since these numbers change multiple times throughout the algorithm and in unpredictable quantities. Yet, we are not able to treat all nodes the same, since some nodes transmit with higher power and broadcasting range than others. So instead of making a distinction between leader and background nodes, we make a distrinction between nodes transmitting with higher power P_{hi} and a higher broadcasting range r_B^{hi} (strong nodes) and those transmitting with lower power P_{lo} and a lower broadcasting range r_B^{lo} (weak nodes). We want strong nodes

to transmit with probability p_{hi} and weak nodes to transmit with probability p_{lo} which leads to transmission durations of d_{hi} and d_{lo} for local broadcasting rounds respectively.

Throughout the course of the algorithm a single node slips into the roles of a strong node and a weak node multiple times and at unpredictable points in time. It is therefore impossible to estimate the number of strong nodes and the number of weak nodes. We are compelled to make a very rough estimation and optimize for the worst case which is that there are only strong nodes accessing the shared medium at the same time. We set up a modified version of Experiment 3.3 (Local Broadcasting) in which we make every node a strong node, transmitting with P_{hi} and r_B^{hi}. We determine the optimal transmission probability by adopting the probability values given in Table 5.1. The results are shown in Table 5.2.

Table 5.1.: Parameter ranges for Local Broadcasting with strong nodes

Scenario	1	2	3	4	5	6
n	500	500	1000	1000	1500	1500
α	2	6	2	6	2	6
p_{start}	0.005	0.010	0.002	0.005	0.002	0.005
p_{end}	0.025	0.070	0.012	0.025	0.008	0.020
p_{step}	0.005	0.005	0.002	0.005	0.001	0.005
Iterations	100	100	100	100	100	100

Table 5.2.: Results of Local Broadcasting with strong nodes

Scenario	1	2	3	4	5	6
n	500	500	1000	1000	1500	1500
α	2	6	2	6	2	6
p_{hi}	0.015	0.030	0.006	0.020	0.005	0.015
d_{hi}	1213	728	2702	1550	4086	2479

As we can see the local broadcasting rounds take a lot longer for strong nodes because every strong node has nine times as many neighbors on average.

Now suppose there are some weak nodes among the strong nodes. The weak nodes have to transmit with $p_{\text{lo}} \leq p_{\text{hi}}$ in order to not cause more interference than if there were only strong nodes. Now suppose there are only weak nodes and no strong nodes. For this scenario we already determined the optimal transmitting probabilities and durations in Chapter 3. Those optimal probabilities are consistently higher than the probabilities for strong nodes in Table 5.2. Since corresponding transmission durations are strictly monotonically decreasing when we approach the optimal transmission probability, we conclude that $p_{\text{lo}} = p_{\text{hi}}$. Keep in mind that this does not necessarily mean that $d_{\text{lo}} = d_{\text{hi}}$, in fact, $d_{\text{lo}}$is considerably lower than d_{hi}. We determine the values for d_{lo} by running Experiment 3.3 (Local Broadcasting), adopting the probability values given in Table 5.2. The results are shown in Table 5.3.

We take these values for p_{hi}, p_{lo}, d_{hi} and d_{lo} and run Yu Coloring with 50 rounds for each testing scenario. The results are shown in Table 5.4.

As stated in Section 2.3, we vary the SINR parameter for the geometric signal decay α in our testing environment in order to see the effect on Yu Coloring in particular. We expect that a higher signal decay ($\alpha = 6$) is more beneficial for Yu Coloring than the other two algorithms, because Yu Coloring uses a higher broadcasting range at times. As we can see in Table 5.4, our expectations seem to be correct. Yu Coloring saves up to 53% of time

Table 5.3.: Results of Local Broadcasting with weak nodes

Scenario	1	2	3	4	5	6
n	500	500	1000	1000	1500	1500
α	2	6	2	6	2	6
p_{lo}	0.015	0.030	0.006	0.020	0.005	0.015
d_{lo}	733	390	2057	660	2466	887

Table 5.4.: Results of the evaluation of Yu Coloring

Scenario	1	2	3	4	5	6
n	500	500	1000	1000	1500	1500
α	2	6	2	6	2	6
p_{hi}	0.015	0.030	0.006	0.020	0.005	0.015
d_{hi}	1213	728	2702	1550	4086	2479
p_{lo}	0.015	0.030	0.006	0.020	0.005	0.015
d_{lo}	733	390	2057	660	2466	887
Average runtime	50745.3 ±3132.3	28559.5 ±1841.2	171009.5 ±8896.0	74494.3 ±3786.3	280140.1 ±12777.8	131550.5 ±5346.0

with $\alpha = 6$ (Scenario 6) in contrast to $\alpha = 2$ (Scenario 5). In Chapter 6 we compare these results and speedups with the ones of the other two algorithms.

5.2. Practical improvements

In contrast to MW Coloring, Yu Coloring is hard to improve. The reason for this is that we cannot make any definitive statements about the number of leaders, background nodes, strong or weak nodes. We therefore have to optimize the parameters p_{hi}, p_{lo}, d_{hi} and d_{lo} for the worst case which is often very far from reality. However, there is one very simple improvement that has a strong impact on the runtime. In state G, a leader v transmits a Grant_u message to a neighbor u for the duration of d_{lo} (Algorithm 5.3, Line 8). After the average transmission duration, which is a lot lower than d_{lo}, node u receives the Grant_u message, transitions to state C_2 and transmits a Color_u message (Algorithm 5.6, Line 3). On receiving that Color_u message, the leader node v can safely stop transmitting the Grant_u message since the recipient has obviously received it. Instead of wasting time transmitting a message that has already been received, leader node v continues to dequeue the next node from its queue of requesting nodes.

The runtime results of the improved Yu Coloring algorithm are shown in Table 5.5. As we can see, the runtimes have improved by a lot, taking only 66% of the time in the case of Scenario 5. It is interesting to see that the speedup increases with increasing average degree Δ. In dense graphs with a high average degree (Scenario 5 and Scenario 6), leaders have more neighbors than in sparse graphs with lower average degree (Scenario 1 and Scenario 2). These leaders have to serve more neighbors and therefore benefit greatly from an early continuation scheme like the one we introduced in this section. It is also interesting to see that the speedup decreases with increasing SINR parameter α. In the case of a higher α parameter, the length of local broadcasting rounds (the maximum transmission duration) is closer to the average duration of a message transmission than with lower α. Since our improvement relies on the gap between average and maximum transmission duration, the speedup is higher if this gap is wider.

In Chapter 6 we compare the results of this chapter in more detail.

Table 5.5.: Results of the evaluation of improved Yu Coloring

Scenario	1	2	3	4	5	6
n	500	500	1000	1000	1500	1500
α	2	6	2	6	2	6
p_{hi}	0.015	0.030	0.006	0.020	0.005	0.015
d_{hi}	1213	728	2702	1550	4086	2479
p_{lo}	0.015	0.030	0.006	0.020	0.005	0.015
d_{lo}	733	390	2057	660	2466	887
Average runtime	43785 ±2403.2	25826.7 ±1988.4	123126.7 ±5457.7	60147.3 ±2924.4	185251.1 ±8291.7	98012.8 ±4705.7
Average speedup	116%	111%	139%	124%	151%	134%

6. Comparison

In this chapter we compare the results of our evaluations of Simple Coloring, MW Coloring and Yu Coloring. For an overview of all average runtimes, refer to Table 6.1. For a comparison of runtimes in relation to Simple Coloring, refer to Table 6.2.

Table 6.1.: Average runtimes of Simple Coloring, MW Coloring and Yu Coloring

Scenario	1	2	3	4	5	6
n	500	500	1000	1000	1500	1500
α	2	6	2	6	2	6
Δ	7.9 ± 1.0	7.9 ± 1.0	13.0 ± 1.3	13.0 ± 1.3	17.3 ± 1.4	17.3 ± 1.4
Simple Coloring	561.3 ±59.6	534.3 ±62.6	1046.0 ±84.8	947.94 ±78.5	1690.0 ±106.3	1577.8 ±130.6
MW Coloring	1909.7 ±239.8	1215.8 ±122.7	5589.7 ±533.7	3393.5 ±346.6	10567.4 ±812.3	6032.2 ±417.9
MW improved	1372.0 ±147.4	933.9 ±114.6	4774.7 ±382.5	2684.3 ±251.1	8321.5 ±835.4	4416.8 ±274.8
Yu Coloring	50745.3 ±3132.3	28559.5 ±1841.2	171009.5 ±8896	74494.3 ±3786.3	280140.1 ±12777.8	131550.5 ±5346
Yu improved	43785 ±2403.2	25826.7 ±1988.4	123126.7 ±5457.7	60147.3 ±2924.4	185251.1 ±8291.7	98012.8 ±4705.7

Table 6.2.: Comparison of average runtimes in relation to Simple Coloring

Scenario	1	2	3	4	5	6
Simple Coloring	1	1	1	1	1	1
MW Coloring	3.4	2.3	5.3	3.6	6.3	3.8
MW improved	2.4	1.7	4.6	2.8	4.9	2.8
Yu Coloring	90.4	53.5	163.5	78.6	165.8	83.4
Yu improved	78	48.3	117.7	63.5	109.6	62.1

Consistent with our expectations, runtimes generally improve with increasing geometric signal decay parameter α and with lower n and therefore lower average Δ. The most harsh conditions are therefore given in Scenario 5 whereas Scenario 2 is the most forgiving one.

As we see, Simple Coloring, the least complex of the three algorithms, outperforms the other two. One reason for this is that Rand4DeltaColoring does not make any assumptions about when a message has been received successfully. No node is transmitting messages for the entire duration of a local broadcasting round and then only proceeding afterwards. Instead, all nodes just react to incoming messages. Since most messages arrive early, most listening nodes can therefore finish their computations early as well. In Rand4DeltaColoring nodes do not have to wait for the worst case transmitting duration. In ColorReduction they have to act according to the worst case, however, thanks to the groundwork done by Rand4DeltaColoring, even that worst-case is acceptable. Nodes are therefore able to transmit with far higher probabilities at far lower transmitting durations.

MW Coloring mostly suffers from the slow counting mechanism which always has to account for the worst-case transmitting duration. That same problem applies to Yu Coloring as well, in fact in an even more severe fashion. Yu Coloring is mostly held back by the need for a maximal independent set in terms of $3r_B$. By tripling the broadcasting range, the average degree in the graph is nine times as high. This leads to far higher transmitting durations. Apart from the counting mechanism, Yu Coloring relies heavily on simulated local broadcasting rounds throughout the algorithm. In almost every step of the algorithm, the worst case transmitting duration has to be considered. In contrast to MW Coloring, Yu Coloring does not leave a lot of room for improvements. The major problem of Yu Coloring is the tripled broadcasting range. If there was a way to calculate a MIS in terms of $3r_B$ without tripling the broadcasting range of every participating node and without introducing too much other overhead, Yu coloring could be vastly improved.

As stated in Section 2.3, we vary the SINR parameter for the geometric signal decay α in our testing environment in order to see the effect on Yu Coloring in particular. We expect that a higher signal decay ($\alpha = 6$) is more beneficial for Yu Coloring than the other two algorithms, because Yu Coloring uses a higher broadcasting range at times. As we can see in Table 6.1, our expectations seem to be correct. Yu Coloring saves up to 50% runtime with $\alpha = 6$ (Scenario 6) in contrast to $\alpha = 2$ (Scenario 5). Simple Coloring only benefits slightly from increased α values and MW Coloring only saves up to 40%.

Since all algorithms were designed to prove asymptotical bounds, the original algorithms are not necessarily optimized to perform well in practice. A fair comparison between the algorithms has to account for improvements that do not improve asymptotical bounds but lead to better performances in practice. As we can see, the improvements we introduce for MW Coloring and Yu Coloring in this thesis have a big impact on the runtime. However, Simple Coloring still beats both.

7. Conclusion

In this thesis we evaluated the runtime of three distributed node coloring algorithms (Simple Coloring, MW Coloring and Yu Coloring) in six different testing scenarios in the SINR model. Since these algorithms are designed with regard to their asymptotical runtime, we had to determine optimal parameters for each algorithm. One fundamental concept in each algorithm is a local broadcasting round. A local broadcasting round is the simulation of a single reliable message transmission in an unreliable network. By running modified local broadcasting experiments, we determined the optimal parameters for each algorithm.

We observed that Simple Coloring is the fastest of the three, while producing an optimal coloring of $\Delta + 1$ colors. MW Coloring is slower by a factor of 2 to 7, producing a coloring of $\mathcal{O}(\Delta)$ colors. Yu Coloring is very hard to optimize for our testing scenarios. We modified the original algorithm in order to make it feasible to find optimal parameters. Our modified version of Yu Coloring is slower than MW Coloring by a factor of about 20 to 30, but produces an optimal coloring of $\Delta + 1$ colors.

For MW Coloring and Yu Coloring we introduced a set of modifications that improve the runtime of these algorithms, considerably narrowing the gap between Simple Coloring and MW Coloring and between MW Coloring and Yu Coloring. The improved version of MW Coloring saves up to 30% of time. The improved version of Yu Coloring saves up to 34% of time.

For our given testing environment, the results are very obvious. However, there might be scenarios where MW coloring or even Yu coloring have an advantage. In our testing environment, nodes wake up at the same time. However, MW and Yu Coloring are designed to handle the case of random wake-ups very well. We also did not consider scenarios where nodes transmit asynchronously or scenarios in which nodes are not stationary but mobile in unpredictable ways. Such properties are common in real-world ad hoc or sensor networks, so further investigation in these directions might be valuable.

Bibliography

[ALPP09] Chen Avin, Zvi Lotker, Francesco Pasquale, and Yvonne-Anne Pignolet. A note on uniform power connectivity in the SINR model. In *Algorithmic Aspects of Wireless Sensor Networks*, pages 116–127. Springer Science Business Media, 2009.

[BE09] Leonid Barenboim and Michael Elkin. Distributed (delta+1)-coloring in linear (in delta) time. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 111–120, 2009.

[BE13] Leonid Barenboim and Michael Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2013.

[DT10] Bilel Derbel and El-Ghazali Talbi. Distributed node coloring in the SINR model. In *2010 IEEE 30th International Conference on Distributed Computing Systems*. Institute of Electrical & Electronics Engineers (IEEE), 2010.

[FP15] Fabian Fuchs and Roman Prutkin. Simple distributed delta + 1 coloring in the SINR model. *CoRR*, abs/1502.02426, 2015.

[FPS04] Irene Finocchi, Alessandro Panconesi, and Riccardo Silvestri. An experimental analysis of simple, distributed vertex coloring algorithms. *Algorithmica*, 41(1):1–23, Sep 2004.

[FW13] Fabian Fuchs and Dorothea Wagner. On local broadcasting schedules and CONGEST algorithms in the SINR model. In *Algorithms for Sensor Systems*, pages 170–184. Springer Science Business Media, Dec 2013.

[G+08] EDC Group et al. Sinalgo-simulator for network algorithms (accessed jan 1, 2015) available: http://www.disco.ethz.ch/projects/sinalgo/, 2008.

[GK00] P. Gupta and P.R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, Mar 2000.

[HM11] Magnús M. Halldórsson and Pradipta Mitra. Nearly optimal bounds for distributed wireless scheduling in the SINR model. In *Automata, Languages and Programming*, pages 625–636. Springer Science Business Media, 2011.

[KMW04] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Radio network clustering from scratch. In *In Proceedings of 12 th Annual European Symposium on Algorithms (ESA*, pages 460–472, 2004.

[KV10] Thomas Kesselheim and Berthold Vöcking. Distributed contention resolution in wireless networks. In *Distributed Computing*, pages 163–178. Springer Science Business Media, 2010.

[MA04] M. Moisio and K. Aschan. The effect of single-antenna interference cancellation on GPRS performance. In *1st International Symposium onWireless Communication Systems, 2004.* Institute of Electrical & Electronics Engineers (IEEE), 2004.

[MW08] Thomas Moscibroda and Roger Wattenhofer. Coloring unstructured radio networks. *Distributed Computing*, 21(4):271–284, 2008.

[Pel00] David Peleg. *Distributed Computing: A Locality-Sensitive Approach.* Society for Industrial & Applied Mathematics (SIAM), Jan 2000.

[SW08] Johannes Schneider and Roger Wattenhofer. A log-star distributed maximal independent set algorithm for growth-bounded graphs. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Principles of Distributed Computing, PODC 2008, Toronto, Canada, August 18-21, 2008*, pages 35–44, 2008.

[YWHL14] Dongxiao Yu, Yuexuan Wang, Qiang-Sheng Hua, and Francis C. M. Lau. Distributed (Δ+1)-coloring in the physical model. *Theor. Comput. Sci.*, 553:37–56, 2014.

Appendix

A. Local Broadcasting

Table A.1.: These are the top ten results of the Local Broadcasting experiment (Experiment 3.3). If all nodes are transmitting with probability p, the length of a local broadcasting round is d.

	Scenario	1	2	3	4	5	6
1.	p	0.088	0.14	0.047	0.1	0.028	0.08
	d	177.0	126.0	385.0	245.0	600.0	361.0
2.	p	0.08	0.13	0.043	0.09	0.03	0.06
	d	179.0	130.0	392.0	251.0	601.0	364.0
3.	p	0.082	0.11	0.048	0.07	0.025	0.07
	d	182.0	133.0	392.0	264.0	632.0	370.0
4.	p	0.078	0.16	0.045	0.11	0.033	0.05
	d	185.0	134.0	396.0	264.0	636.0	378.0
5.	p	0.086	0.15	0.038	0.12	0.032	0.09
	d	186.0	141.0	400.0	265.0	642.0	382.0
6.	p	0.09	0.12	0.041	0.08	0.031	0.1
	d	186.0	143.0	401.0	272.0	643.0	417.0
7.	p	0.092	0.19	0.037	0.13	0.034	0.04
	d	186.0	143.0	404.0	283.0	653.0	439.0
8.	p	0.074	0.17	0.044	0.14	0.029	0.11
	d	187.0	145.0	407.0	287.0	657.0	442.0
9.	p	0.076	0.1	0.039	0.06	0.026	0.12
	d	194.0	151.0	411.0	293.0	659.0	498.0
10.	p	0.096	0.18	0.046	0.05	0.027	0.13
	d	194.0	153.0	411.0	314.0	659.0	529.0