Algorithms for Efficient Communication in Wireless Sensor Networks

Distributed Node Coloring and its Application in the SINR Model

zur Erlangung des akademischen Grades eines

Doktor der Naturwissenschaften

von der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Fabian Fuchs

aus Sinsheim

Tag der mündlichen Prüfung:	19. November 2015
Erster Gutachter:	Prof. Dr. Dorothea Wagner
Zweiter Gutachter:	Prof. Magnús M. Halldórsson, PhD

Fabian Fuchs: *Algorithms for Efficient Communication in Wireless Sensor Networks - Distributed Node Coloring and its Application in the SINR Model*, © 27. Jan. 2016

To my wife Mela
and our daughter Magdalene

The LORD has remembered us;
he will bless us.

— Psalm 115:12

ACKNOWLEDGMENTS

During my PhD, many people supported me in different ways. I would like to take this opportunity to say thank them.

First, I would like to thank Prof. Dr. Dorothea Wagner for providing me with the opportunity of pursuing a PhD and inviting me to join her research group. Thank you for your leadership, your trust and the freedom to choose my own direction of research!

Thank you Magnús, for taking the time to read and evaluate this thesis along with traveling to Karlsruhe for my defence. Thank you also for your previous visit and the invitation to a great Dagstuhl seminar.

Many thanks also to Roman, who was my office mate during almost the entire PhD. I am grateful for many fruitful discussions, collaborations, and for proofreading almost all of my publications; to Markus Völker for introducing me to the topic of sensor networks and many valuable tips right at the start of my PhD; to Andreas for proofreading large parts of this thesis (all errors are mine!), answering any question, and the (almost) never-ending supply of sweets; to our "saboteurs" Franzi and Benjamin; to Thomas B. for advice regarding emacs and ipe; to Moritz for organizing a ton of leisure activities (including soccer, which I recently joined too rarely); to Ignaz for sharing your experience and knowledge; to Ben and Michael for your server and Dokuwiki support; to Tamara for welcoming me in your office when I started; to Julian for your help regarding the MSR-tax; to Tanja for travel tips to Iceland; to Thomas P. for making simple words so much fun; to Andrea for our trip to Israel; to Tobias for your beautiful and precise handcraft for PhD hat; to Marcel and Valentin for pleasant chats; to Lilian and Simone for all the "small things" that are so essential; to the members of the *Escorial Committee* for your solidarity and the fun we had. And simply all of you for being great colleagues!

To my parents Ulrich und Beate: Thank you for raising me the way you did, for your love, and your support in any situation. This means more to me than words can tell. Thanks are also to my siblings Sebastian and Anna-Lena; thank you for being best friends during childhood and beyond.

Finally, to my beloved wife Mela: I am deeply grateful for your enduring support, your patience (especially during the final phase of this thesis) and your love. Magdalene, thank you for being around.

Thank you!

DEUTSCHE ZUSAMMENFASSUNG

Drahtlose Sensornetzwerke bestehen meist aus einer Vielzahl miniaturisierter Computer, sogenannten Netzwerkknoten. Diese können zusätzlich zur Batterie und einer drahtlosen Kommunikationseinheit mit verschiedenen Sensoren ausgerüstet sein. Die in dieser Arbeit betrachteten Algorithmen berechnen Strukturen und Zustände, welche den Aufbau einer effizienten Kommunikationsstruktur innerhalb eines Sensornetzwerkes ermöglichen. Alle in dieser Arbeit vorgestellten Algorithmen arbeiten verteilt, wodurch die Netzwerke ohne zentralen Koordinationscomputer und die entsprechende Kommunikation zu und von diesem Computer auskommen. Ich betrachte grundlegende Probleme, welche direkt nach dem Ausbringen und Starten des Netzwerkes auftreten, zum Beispiel den initialen Austausch von Nachrichten mit den jeweiligen Nachbarn. Um den sich hieraus ergebenden Anforderungen gerecht zu werden, verwende ich das Signal-zu-Interferenz-und-Rausch-Verhältnis (Signal-to-Interference-and-Noise-Ratio, *SINR*), welches die Störung der Signale realistisch modelliert. Das SINR-Modell wird schon seit Jahrzehnten, z.B. im elektrotechnischen Bereich, verwendet, wohingegen es im algorithmischen Bereich erst seit ca. 10 Jahren Verwendung findet. Im geometrischen SINR-Modell wird, anhand der Sendeleistung, und der Dämpfung der Signalstärke über die Sendedistanz die Störung einer Übertragung auf gleichzeitig stattfindende Übertragungen modelliert. Im Gegensatz zu zuvor betrachteten algorithmischen Modellen, werden in SINR-Modell nicht nur lokale Auswirkungen einer Übertragung berücksichtigt, sondern die Auswirkungen von Störungen (durch gleichzeitige Übertragung im Netzwerk) werden global, d.h. auf das komplette Netzwerk bezogen, betrachtet, siehe Abbildung 1 (rechts). Die Analyse von Algorithmen im SINR-Modell ist sehr aufwendig, daher ist sie nur für grundlegende Algorithmen gerechtfertigt. Die Analyse komplexer Algorithmen ist oft nur in abstrakteren Modellen möglich.

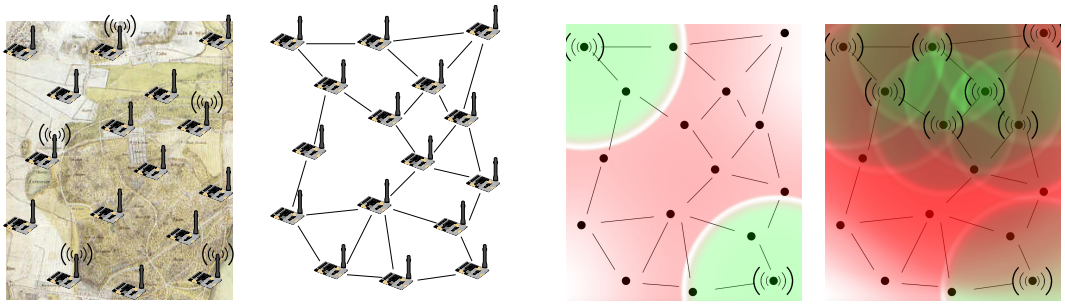


Abbildung 1: Links: Beispiel eines großflächig verteilten Sensornetzwerkes mit den Nachbarschaftsbeziehungen und dem Kommunikationsgraphen, die sich aus dem SINR-Modell ergeben (mitte-links). Rechts: Im SINR-Modell akkumulieren sich Störungen (im Bild rot), womit Übertragungen globalen Einfluss haben. Wenn z.B. alle Knoten der oberen Hälfte senden, kann auch in der unteren Hälfte keine Übertragung stattfinden (rechts-außen).

Hierzu betrachte ich das etablierte message-passing Modell $\mathcal{CONGEST}$, bei welchem ohne Berücksichtigung von Interferenzen rundenbasiert Nachrichten mit den Nachbarn ausgetauscht werden.

Im ersten Teil meiner Arbeit werden zwei sehr grundlegende Probleme betrachtet und im SINR-Modell analysiert. Beim ersten Problem handelt es sich um Local Broadcasting, hierbei soll es jedem Knoten ermöglicht werden eine Nachricht mit seinen Nachbarn auszutauschen. Ich erweitere bestehende Algorithmen sowie deren Analyse, sodass diese auch in Netzwerken mit nicht-einheitlicher Sendeleistung eingesetzt werden können. Im Anschluss betrachte ich das Problem der Knotenfärbung, bei dem eine Zuweisung von Knoten zu je einer Farbe (oder Zahl) gefunden werden soll, sodass sich die Farben von je zwei Nachbarn unterscheiden.

Im zweiten Teil wende ich mich weniger komplexen message-passing Modellen zu, welche durch Abstraktion einiger Aspekte der drahtlosen Kommunikation, die Analyse wesentlich umfangreicherer Algorithmen erlauben. Um die Algorithmen weiterhin im realistischeren SINR-Modell ausführen zu können, gebe ich zunächst eine Methode an, mittels derer beliebige Algorithmen, die für message-passing-Modelle entworfen wurden, im SINR-Modell ausgeführt werden können. Anschließend betrachte ich eine komplexere Methode zur Steigerung der Kommunikationseffizienz im $\mathcal{CONGEST}$-Modell.

INITIALE KOMMUNIKATION Eines der fundamentalsten Problemen in Drahtlosnetzwerken ist die effiziente Kommunikation. Beschränken wir die Kommunikation darauf, dass jeder Knoten im Netzwerk eine Nachricht zu allen seinen Nachbarn senden soll, nennt sich dieses Problem im SINR-Modell das Local Broadcasting Problem. Da zur Lösung dieses Problems wenig Vorwissen und keine Vorberechnung im Netzwerk benötigt wird, ist Local Broadcasting zur initialen Kommunikation in einem Ad-Hoc- oder Sensornetzwerk direkt nach Inbetriebnahme gut geeignet. Für den Fall einer einheitlichen Sendeleistung an allen Knoten im Netzwerk existieren mehrere Algorithmen, die dieses Problem lösen. Gerade im Hinblick auf die Erweiterung existierender Netzwerke zu dichten und heterogenen Netzwerken (z.B. Internet of Things) oder Energieeinsparungen durch eine Reduktion der Sendeenergie ist die Annahme einheitlicher Sendeleistungen nicht mehr für alle Einsatzgebiete ausreichend. In der Arbeit erweitere ich bestehende Local Broadcasting Algorithmen sowie deren Analyse, sodass die Ausführbarkeit der Algorithmen in Drahtlosnetzwerken mit heterogenen Sendeleistungen möglich ist.

KNOTENFÄRBUNG Färbung von Knoten ist ein klassisches Problem, bei dem jeder Knoten im Netzwerk eine von möglichst wenigen Farben wählen soll, sodass keine zwei benachbarte Knoten die selbe Farbe gewählt haben. Eine Knotenfärbung kann zum Beispiel zur Steigerung der Kommunikationseffizienz nach Inbetriebnahme eines drahtlosen Ad-Hoc-Netzwerkes verwendet werden, was ich im zweiten Teil näher betrachten werde. Da es zentralisiert schon NP-schwer ist eine Färbung mit der minimalen Anzahl an Farben zu berechnen, gilt für verteilte Kno-

tenfärbungsalgorithmen eine Färbung mit $d+1$ Farben als ultimatives Ziel, da eine solche Färbung in einem Netzwerk mit maximalem Knotengrad d immer existiert.

In diesem Teil generalisiere ich, basierend auf der zuvor vorgestellten Methode zur initialen Kommunikation, zunächst einen Knotenfärbungsalgorithmus, sodass dieser in Netzen mit heterogenen Sendeleistungen genutzt werden kann. Im Anschluss betrachte ich wieder den Fall einheitlicher Sendeleistungen, und schlage zwei randomisierte Färbungsalgorithmen vor. Der erste Algorithmus nutzt unsichere Kommunikation, und damit einen schnelleren, wenn auch teilweise fehlerbehafteten Nachrichtenaustausch zwischen den Knoten, um eine korrekte Färbung innerhalb kurzer Zeit zu berechnen. Der zweite Algorithmus nutzt die berechnete Färbung um die sichere Kommunikation ausgewählter Knoten zu beschleunigen, und verringert dadurch die Zahl der genutzten Farben auf das gesteckte Ziel von $d+1$ Farben. Beide Algorithmen erreichen eine Laufzeit, die asymptotisch einer einzigen Kommunikationsrunde entspricht, was als untere Schranke gesehen werden kann und bestehende Algorithmen entweder in der Anzahl Farben, der asymptotischen Laufzeit, oder beidem unterbietet. Um auch die praktische Relevanz meiner Algorithmen nachzuweisen, zeige ich mittels einer experimentellen Evaluation in einem Simulator, dass die von mir vorgeschlagenen Algorithmen eine bessere Laufzeit erreichen als bestehende Algorithmen.

EFFIZIENTE KOMMUNIKATION AUF HÖHEREM LEVEL Da es relativ technisch und aufwendig ist, Algorithmen im SINR-Modell zu analysieren, wird im zweiten Teil ein Verfahren beschrieben, welches eine effiziente Ausführung von Algorithmen, die für message-passing Modelle entwickelt wurden, ermöglicht. Damit ermögliche ich die Analyse weiterer Algorithmen in einfacheren, weniger detaillierten Modellen, während gleichzeitig die effiziente Ausführbarkeit im SINR-Modell erhalten bleibt. Dazu stelle ich einen Algorithmus zur Berechnung eines Zeitschlitzverfahrens (Time-Division-Multiple-Access, TDMA) im SINR vor. Mittels eines solchen Zeitschlitzverfahrens kann in einem sogenannten TDMA-Schedule jedem Knoten ein Zeitschlitz zugewiesen werden, in dem dieser seine Nachricht überträgt. Als Grundlage des Algorithmus verwende ich eine bestehende Knotenfärbung, sowie zusätzlich die Position des Knotens. Aufbauend auf der Knotenfärbung wird nun eine zweite, überregionale Knotenfärbung anhand eines geometrischen Gitters berechnet, siehe Abbildung 2. Die Kombination aus lokaler und überregionaler Knotenfärbung ermöglicht die Berechnung des TDMA-Schedules, mittels dem die Knoten Local Broadcasting in asymptotisch optimaler Zeit absolvieren können.

Abschließend stelle ich einen Algorithmus vor, der die globale Kommunikation, also die Verteilung von Nachrichten im Netzwerk effizient gestalten kann. Es handelt sich um die Verbesserung eines bestehenden Algorithmus welche in dünnen und/oder großen Netzwerken Laufzeitvorteile bringt. In einem mehrfach zusammenhängenden Netzwerk berechnet der Algorithmus eine Menge von Connected Dominating Sets (CDS, verbundene dominierende Menge), von denen jede einzelne Menge in einem Schritt das komplette Netzwerk erreichen kann. Insgesamt hat

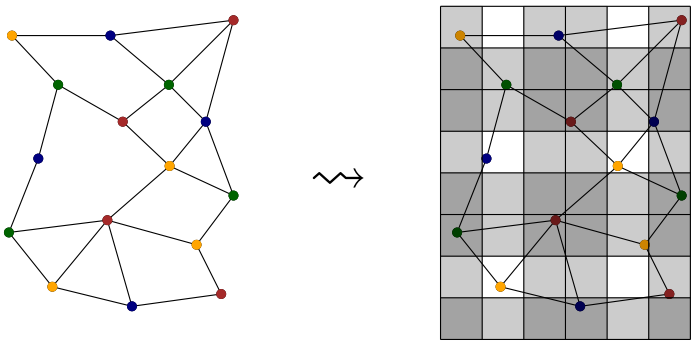


Abbildung 2: Links: Eine valide Knotenfärbung, bei der keine zwei Nachbarn die selbe Farbe haben. Rechts wird basierend auf der Knotenfärbung und einem geometrischen Gitter ein TDMA Schedule errechnet, der Kommunikation im SINR-Modell in asymptotisch optimaler Zeit ermöglicht. Es dürfen hier z.B. alle gelben Knoten in einem weißen Feld gleichzeitig senden. Dies wird im zweiten Teil der Arbeit verwendet um Algorithmen abstrakterer Modelle im SINR-Modell zu simulieren.

also jeder Knoten im Netzwerk einen Nachbarn in jedem CDS, wodurch diese für den Nachrichtentransport verwendet werden können. Die Berechnung geschieht in einem mehrschichtigen Ansatz, wobei in jeder Schicht versucht wird, die Menge an nicht-verbundenen Teilstücken der (zukünftigen) CDS zu verringern. Meine Verbesserung besteht darin, die Auswahl der zu wählenden Pfade über ein Matching in einem konstruierten virtuellen Graphen zu bestimmen.

CONTENTS

1

INTRODUCTION

Wireless networks are ubiquitous. At home, in the office, in-between, and even in nature, we are constantly connected using wireless or mobile networks. This constant connectivity allows us to access all kinds of information from virtually everywhere and connect with friends, family and colleagues even at remote places. Regardless of the immense progress made for wireless networks in the last decades, we are already on the verge of yet another technology: Large-scale wireless ad hoc and sensor networks. In contrast to most current wireless systems, which are organized according to the client-server model[1], wireless sensor networks consist of countless small sensor devices that cooperate to achieve connectivity and build a reliable network without any infrastructure. The sensor nodes have a limited battery capacity, which requires them to conserve energy whenever possible. Thus, communication is limited to relatively small transmission ranges requiring the nodes to communicate using multi-hop transmissions, i.e. messages are transmitted only to close-by nodes, which relay the message until the message reaches its destination. Such multi-hop transmissions and the self-organizing structure of the network are the main structural characteristics of wireless sensor networks.

"We envision that, in future, wireless sensor networks will be an integral part of our lives, more so than the present-day personal computers." ([1], 2002)

A notable application for this technology are sensor nodes that can used in disaster relief, e.g. [22, 53]. In the future such nodes could be deployed using a plane flying over a disaster area. Although the infrastructure may be destroyed the self-coordinating nature of wireless sensor networks establishes multi-hop communication, which allows to monitor and inspect the disasters impact, aggregates sensed information, and support first responders on the scene by providing communication as well as the aggregated sensor data. We depict an example situation in Figure 1.1. Another example of how such networks can be used is the tracking and observation of animals, as depicted in Figure 1.2 for white storks trav-

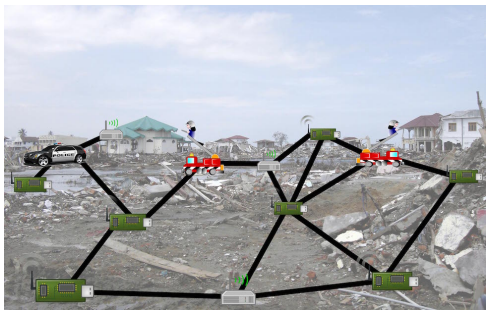


Figure 1.1: Example of a wireless sensor network in a disaster situation. The network provides sensed information (e.g. audio, movement) and multi-hop communication to aid helpers.

1 In the client-server model a designated central server (e.g. wireless access point, cellphone transmitter towers) communicates with several potentially mobile clients.

eling from southern Germany to Spain and Africa. Such networks may include many smaller sensors with limited transmission power and only few more powerful nodes that aggregate the data and transmit it to a base station [77].

Figure 1.2: Tracks of white storks from Germany obtained using wireless sensors (https://www.movebank.org/panel_embedded_movebank_webapp[2]).

Currently, large scale deployment of wireless sensor networks is delayed due to "closed or proprietary systems [that] are connectivity islands with limited communication to the external world" [99]. Future wireless networks, however, may benefit from being integrated into the "Internet of Things" (IoT) concept, enabling interoperability between different sensor networks [99]. Alcaraz et al. expect that "the potential of the wireless sensor networks (WSN) paradigm will be fully unleashed once it is [..] becoming part of the Internet of Things" [2]. Several other applications can be found in surveys on wireless sensor networks [1, 116, 132].

In this thesis we consider wireless sensor networks directly after their deployment and propose algorithms that either enable initial communication among the nodes in the network or solve problems and compute structures that aid the nodes in establishing more efficient communication. We analyze our algorithms in the realistic Signal-to-Interference-and-Noise-Ratio (SINR) model of interference and mathematically prove their correctness and runtime bounds. The geometric SINR model models interference based on the signal transmissions of all nodes in the network and the attenuation of the signals with distance.

The SINR model is widely used in the electric engineering community for decades, and has been adopted by the algorithmic community in recent years, starting with the seminal work of Gupta and Kumar [60]. In contrast to earlier algorithmic models for wireless communication such as the protocol model, the SINR model incorporates the global nature of interference instead of considering it a merely local problem, as illustrated in Figure 1.3. Based on the SINR model we derive neighborhood relations to model the wireless network as a graph and tackle the problems of establishing initial or efficient communication algorith-

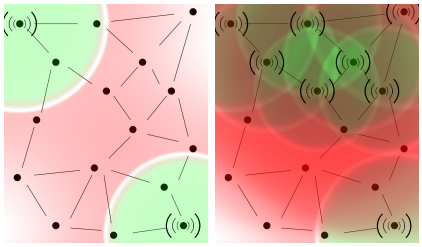


Figure 1.3: Left: Two nodes successfully transmit to their neighbors. Right: Interference from the upper part prohibits a single transmission in the lower part.

2 Using the dataset "Life Track White Stork SWGermany 2014-2015", with kind approval of the Max-Planck-Institute of Ornithology, Radolfzell, Germany.

mically. The algorithms considered in this thesis do not require a central authority or centralized computation, but are executed distributively on each node in the network. The relatively high effort of designing algorithms in the SINR model and especially proving their correctness is justified for network tasks that establish initial or efficient basic communication among the nodes in the network. However, for higher level-task and more complex algorithms one wishes for a more abstract model of communication. Therefore we additionally consider the well-known message-passing model (vertex-) $\mathcal{CONGEST}$, in which interference is not considered and communication is realized in synchronized rounds. This more abstract model allows that one message of limited size can be exchanged between two neighboring nodes in each round.

OVERVIEW AND CONTRIBUTION

In this thesis we consider distributed algorithms that enable efficient communication in wireless ad hoc and sensor networks. For most part of the thesis we use the SINR model of interference to analyze the algorithms. We divide the thesis in two parts. In the first part we consider algorithms that establish initial communication in the network and solve the basic problem of distributively computing a valid node coloring. In the second part we work towards allowing more complex and abstract algorithms to be efficiently executed in wireless sensor networks. We use a node coloring as computed in the first part to compute an asymptotically optimal local broadcast schedule, which allows to efficiently execute $\mathcal{CONGEST}$ algorithms in the SINR model. Additionally, we present a $\mathcal{CONGEST}$ algorithm that computes an efficient broadcast backbone in networks of high vertex-connectivity. In the following we describe each chapter of this thesis in more detail.

Chapter 2: Preliminaries

We describe basic concepts of graph theory and models of wireless communication that are used throughout the thesis. Also, we give an overview on related work in the SINR model and introduce the simulation environment used to experimentally evaluate our algorithms of Chapters 3 and 4. Let us pre-empt that n denotes the number of nodes in the network and Δ the maximum degree in the communication graph, as this is already used in this overview.

Part I: Algorithms for the SINR Model

Chapter 3: Arbitrary Transmission Powers in the SINR Model

Most algorithmic work in the SINR model considers the case of uniform transmission power. In this chapter we allow the nodes to use arbitrary transmission powers, which may be required in heterogenous wireless sensor networks. We show theoretical bounds on the probabilistic interference of the network based on local guarantees on the sum of transmission probabilities from within each neigh-

borhood. This allows us to generalize multiple local broadcasting algorithms to this more general case. These algorithms can be used to communicate and thereby learn about neighboring nodes as depicted in Figure 1.4. We additionally apply our bounds on the interference by generalizing a known distributed node coloring algorithm to this setting, in which unidirectional communication makes the color selection more complicated.

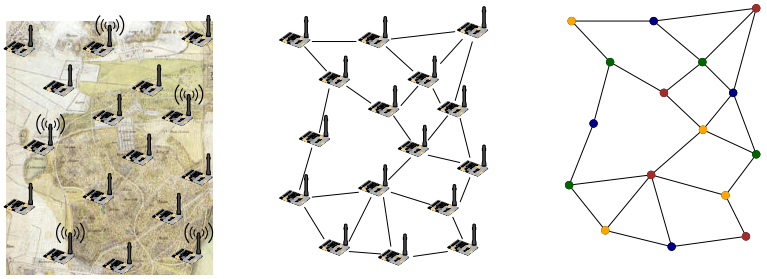


Figure 1.4: Nodes in a wireless network (left) can use local broadcasting to learn their neighborhood relations (center) as considered in Chapter 3. Using initial communication, we consider the distributed computation of node colorings (right) in Chapters 3 and 4.

This chapter is based on joint work with Dorothea Wagner. Preliminary results have been published in [46].

Chapter 4: Distributed $(\Delta+1)$-Coloring in the SINR Model

Node coloring is one of the most popular graph problems, as it has many applications both in the centralized and the distributed setting, e.g. [61, 101, 131]. In our setting, the most popular application of node coloring is the scheduling the transmissions of nodes (as considered in Chapter 6). We propose two algorithms that are both based on simple and well-known algorithms from the message-passing models. The effort we undertake in this chapter is mostly making these algorithms efficient in the SINR model. Our first algorithm, Rand4DColoring, is based on phases and each node selects a new color at the end of a phase whenever it detected a conflict in the phase. Apart from our second algorithm, this simple algorithms already achieves the best coloring (i.e., the lowest number of colors) if restricted to algorithms of the same runtime in the SINR model. However, our main contribution in this chapter is the second algorithm, ColorReduction, which uses a $\mathcal{O}(\Delta)$-coloring to compute a $(\Delta+1)$-coloring. As it is NP-hard to color a graph with the minimum number of colors even in a centralized way [85], a $(\Delta+1)$-coloring (which is always possible) is considered the ultimate goal in distributed computing. Both algorithms achieve a runtime of $\mathcal{O}(\Delta \log n)$ time slots, which is optimal unless faster local broadcasting can be achieved.

This chapter is based on joint work with Roman Prutkin. Preliminary results have been published in [40, 44].

Chapter 5: Experimental Evaluation of Distributed Node Coloring Algorithms

In this chapter we experimentally evaluate the distributed node coloring algorithms proposed in the previous chapter using a network simulator. We compare them against other coloring algorithms proposed for the SINR model using a set of different deployment strategies. Additionally we consider heuristic improvements to the algorithms that achieve a significantly faster execution of the algorithms. We observe that Rand4DColoring is very fast, requiring even less time than required to successfully complete one local broadcast. Regarding our $(\Delta + 1)$-coloring algorithm ColorReduction, we found that its practical performance does not rely on a valid node coloring as input but random colors are sufficient. Additionally, it is a lot faster than competing $(\Delta + 1)$-coloring algorithms (apart from a heuristic based on our own Rand4DColoring algorithm). The considered heuristic improvements reduce the runtime, while preserving the relative ordering among the algorithms.

Part II: Towards $\mathcal{CONGEST}$ Algorithms

Chapter 6: Simulating $\mathcal{CONGEST}$ Algorithms in the SINR Model

The previous part was concerned with establishing initial communication and distributively computing a node coloring. In this chapter we propose a method to compute a local broadcasting schedule of optimal length, which enables the efficient execution of higher-level algorithms in the network. We use a distributed node coloring algorithm as e.g. computed in Chapter 4 and combine it with a method called dilution to make the schedule implied by the coloring feasible in the SINR model. We use position information and compute the schedule in time $\mathcal{O}(\Delta \log n)$ time slots. We additionally show that the runtime of the execution of higher-level algorithms is essentially optimal.

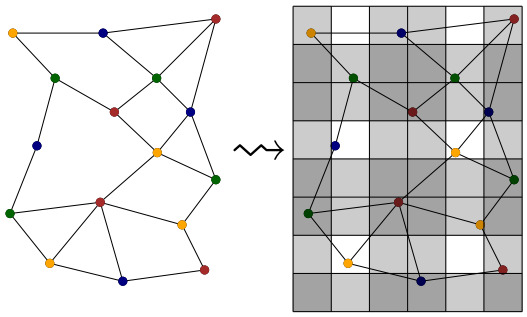


Figure 1.5: A valid node coloring combining it with position information can be used to compute a valid local broadcasting schedule.

This chapter is based on joint work with Dorothea Wagner. Preliminary results have been published in [47].

Chapter 7: Improved Distributed Construction of FCDS packings.

In this chapter we improve an algorithm that operates in the $\mathcal{CONGEST}$ model and computes a broadcasting backbone in networks of high vertex-connectivity. Intuitively, the higher the vertex-connectivity, the more routes exist between any two nodes in the network. This can be utilized by computing multiple so-called fractional connected dominating sets (FCDSs), which can be seen as a network

backbones. We improve the runtime of the algorithm for large and sparse networks by explicitly computing which paths should be selected to connect existing (but not yet connected) dominating sets. To achieve this we construct a helper graph on which a maximal matching algorithm selects a sufficient number of connecting paths.

This chapter is based on joint work with Matthias Wolf.

Part III: Conclusion

Chapter 8: Conclusion

In the last chapter of this thesis we conclude with a brief summary of our main contributions and give an outlook on interesting research directions.

2

PRELIMINARIES

In this chapter we introduce basic concepts and notations used throughout the thesis. We start with graph theory, followed by models for distributed and wireless communication. Additionally we give an overview on related work in the SINR model and introduce the simulation framework used in our experimental evaluation. In the remainder of this thesis we assume some familiarity with basic mathematical concepts. A more thorough introduction to graph theory and, e.g. the big-O-notation can be found in the books "Graph Theory" by Diestel [31] and "Introduction to Algorithms" by Cormen, Leiserson, Rivest and Stein [26]. For more background on probability theory we refer to "Basic Probability Theory" by Ash [5].

2.1 GRAPH THEORY

Modeling real-world problems using graphs has a rich history, starting with Euler, who modeled the famous Eulerian problem using a graph. A *directed graph* $G = (V, E)$ consists of sets V and $E \subseteq V \times V$. An element $v \in V$ is called *vertex* or *node*, and $(u, v) \in E$ is called *edge* or *link* and directed from u to v. For an *undirected graph* $G = (V, E)$ we require that for each $(u, v) \in E$ the edge (v, u) is also in E. Modeling a network using a graph is very natural as each computer or sensor node in the network can be represented by a node in the graph. For wired networks, the network cable between computer A to computer B is modeled as an edge between the corresponding nodes in the graph, while for wireless networks two nodes in the graph are connected if the corresponding nodes in the network can communicate (based on the communication model). This notation is very helpful, as it allows to consider and solve problems based on the abstract graph and thus independent from the actual network. We denote $n := |V|$ as the number of nodes in the network. Two nodes $u, v \in V$ are *adjacent* if there is an edge $(u, v) \in E$. If u is adjacent to v, we additionally call v a *neighbor* of u. We denote N_v as the set of neighbors of v, $N_v^+ = N_v \cup \{v\}$, and say that $\Delta := |N_v|$ is the *degree* of v.

directed graph

undirected graph

adjacent

neighbor

degree

A *path* of length k from u to w in G is a sequence $(u = v_1, \ldots, v_k = w)$ of distinct nodes $v_1, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E$ for any $i \in \{1, \ldots, k-1\}$. For two nodes $u, v \in V$ the *hop-distance* describes the number of edges that must be traversed from u to reach v. Thus, the graph-theoretic distance between two adjacent nodes is 1 hop or simply 1. The *diameter* D of a graph (or network)is the maxi-

path

hop-distance

diameter

cycle mal hop-distance of any two nodes in the graph. A *cycle* is a sequence $(v_1, \ldots, v_k)$ of nodes $v_1, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E$ and $v_1 = v_k$. In a *simple* cycle the nodes $v_1, \ldots, v_{k-1}$ are distinct.

independent A set $C \subseteq V$ is *connected* if for each pair $u, v \in C$ there is a path from u to v. Two nodes $u, v \in V$ are *independent* if they are not adjacent, and a set $I \subseteq V$ is independent if all nodes in the set are pairwise independent. If the set is a *MIS* maximal set in G we call it *maximal independent set* (*MIS*). A set $D \subseteq V$ is *dominating* *dominating* if for each node $u \in V$ it holds that either $u \in D$ or a neighbor of u is in D. We denote a connected dominating set by CDS. Note that a MIS is dominating but *matching* not necessarily connected. A set $M \subseteq E$ is a *matching* in G if no two edges in M contain the same vertex. A matching is *maximal* if for each edge $e \in V \backslash E$ it holds that $M \cup \{e\}$ is not a matching anymore. A *maximum matching* for G is one with maximum cardinality among all matchings for G.

coloring A *coloring* is an assignment of integers (so-called colors) to nodes of a graph. A *valid* color c_v of v is *valid* or proper, if no neighbor of v has the same color as v. The *node coloring problem* is the classical problem of assigning each node a valid color. Note that in a valid coloring the nodes of one color form an independent set. We denote the set of integers $\{0, \ldots, d\}$ by $[d]$ and say that a coloring is a *d-coloring* if uses only colors from $[d]$. Let us now introduce graph-theoretic problems that are related to wireless networks but mostly independent of the considered interference model.

local broadcasting In the *local broadcasting* problem, each node in the graph must transmit one (identical) message to its neighbors in the network. A time division multiple access *TDMA* (*TDMA*) schedule, is an assignment of nodes to time slots. A TDMA schedule is feasible in the interference model if each node successfully transmits to its neighbors once during the schedule. A node coloring corresponds to a TDMA schedule [1] by assigning each color to a different time slot, however, even a valid node coloring does usually not correspond to a feasible schedule (depending on the interference model). A related problem is to compute a *local broadcasting schedule*, which ensures that each node in the network is successful at local broadcasting at least once during the schedule. Broadcasting can be seen as the global equivalent to local *broadcast* broadcasting. In the often considered single source *broadcast* problem the message of only one source node must be distributed to each node in the network.

2.2 MODELS FOR DISTRIBUTED COMPUTING AND WIRELESS COMMUNICATION

In this section we introduce models considered for distributed computing and wireless networks. Apart from the models directly used to describe our algorithms we briefly mention related models. We use a top-down approach and start with the most abstract models.

1 We often only refer to TDMA, however, the techniques usually transfer to related techniques such as frequency division multiple access (FDMA) and code division multiple access (CDMA) [113].

2.2.1 $\mathcal{LOCAL}$ and $\mathcal{CONGEST}$

In these standard message-passing models we assume a communication graph to be given. The $\mathcal{LOCAL}$ model is used to study the locality of problems. Each node in the network is allowed to transmit one message of unlimited size to each of its neighbors. The computational power of the nodes is potentially unlimited, however, often simple computations are sufficient. The $\mathcal{CONGEST}$ model focuses on the effects of congestion in distributed networks and restricts messages to a size of $\mathcal{O}(\log n)$. Hence, with one message only a constant number of node IDs in the range $[n]$ can be transmitted in this model. There are two variants of this model. In the first variant, edge-$\mathcal{CONGEST}$, the nodes may transmit a different message to each neighbor in each round, while in the second variant, vertex-$\mathcal{CONGEST}$, it must be the same message. The edge-$\mathcal{CONGEST}$ model better fits the conditions of a wired network, while vertex-$\mathcal{CONGEST}$ fits the specifics of wireless transmissions. Thus, we use $\mathcal{CONGEST}$ to refer to the vertex-$\mathcal{CONGEST}$ model. For a thorough introduction we refer to the book "Distributed Computing: A Locality-Sensitive Approach" by Peleg [109].

2.2.2 *Protocol Model*

The term protocol model subsumes graph-based interference models that base the decision whether a transmission is feasible or not on a transmission and a interference range [60]. The first graph-based model was introduced by Chlamtac and Kutten [23] to study the broadcasting problem. In these models a transmission by a node v can successfully be received by a node u in v's transmission range iff u is not in the transmission range of a simultaneously transmitting node. The transmission range is usually based on the signal-to-noise-ratio and implies a communication graph. A so-called conflict graph is determined by the interference range, which is usually set to be at least the transmission range, however, it is an open problem to determine how this range should be set optimally [121]. Both ranges are usually uniform, which implies that the communication graph and the conflict graph are both a unit disk graph [24]. Interference in these models is binary, as a node that is in the interference range of a currently transmitting node cannot receive another message, while the model assumes that there is no interference once the node is not in the interference range of a transmitting node. These simplifications led to widely use (e.g. [112, 129]), however, Shi et al. notes that if used "blindly" the solutions obtained using this model "are likely to be infeasible in practice" [121].

2.2.3 *SINR Model*

In the *Signal-to-Interference-and-Noise-Ratio* (SINR) model, also denoted as the physical model due to its common use in electrical engineering, wireless communication is modeled based on the signal transmission and a geometric decay of the signal strength. It improves on the protocol model, which considers interference as a lo-

cal and binary property. It has been shown that protocols explicitly designed for the SINR model can achieve a better performance than theoretically possible for graph-based models in [105].

feasible

In the SINR model a transmission from a sender to a receiver is *feasible* if it can be decoded by the receiver. It depends on the ratio between the desired signal and the sum of interference from other nodes plus the background noise whether a certain transmission is successful. Let each node v in the network use the same transmission power P. Then a transmission from u to v is feasible if and only if

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

where $\alpha \in [2,6]$ is the attenuation coefficient, the constant $\beta > 1$ depends on the hardware, N denotes the environmental noise, $\text{dist}(u,v)$ the Euclidean distance between two nodes u and v, and $I \subseteq V$ is the set of nodes transmitting simultaneously to u. The size of messages is restricted to $\mathcal{O}(\log n)$, as for example in the $\mathcal{CONGEST}$ model. The *broadcasting range* r^B of a node v defines the range around v up to which v's messages should be received. Based on the SINR constraint, the *transmission range* $r^T \leq (\frac{P}{\beta \mathrm{N}})^{1/\alpha}$ is an upper bound for the broadcasting range (with $r^B < r^T$ to allow multiple simultaneous transmissions). Let the *broadcasting region* B_v be the disk with range r^B centered at v. We use the broadcasting range to define the neighborhood relation, thus for a node v it holds that $N_v = \{w \in V \mid \text{dist}(v,w) \leq r^B\}$.

broadcasting range

transmission range

To prove successful communication within the broadcasting range, we need the concept of a *proximity range* $r^A > 2r^B$ around a node v, as introduced in [55]. Let Δ_v^A be the number of nodes with distance less than r^A to v, and $\Delta^A := \max_{v \in V} \Delta_A^v$. To establish communication using local broadcasting, sometimes Δ^A is required, however, as $\Delta^A \in \mathcal{O}(\Delta)$ knowing only Δ introduces only a constant increase in the runtime. We use the following mathematical fact in our analysis, which can be found, for example, in the mathematical background section of [106].

proximity range

Fact 2.1. *For all n, t, such that $n \geq 1$ and $|t| \leq n$,*

$$e^t \left(1 - \frac{t^2}{n}\right) \leq \left(1 + \frac{t}{n}\right)^n \leq e^t.$$

In the *synchronous setting*, we assume that nodes start the algorithm at the same time. In the more realistic *asynchronous setting*, arbitrary wake-up of nodes is allowed, and we do not require synchronized time slots; precise clocks, however, are assumed. With the so-called ALOHA trick [115], e.g. as used in [55], we use time slots in our analysis, although the nodes do not have common time slots. We do not require *carrier sensing* or free acknowledgements (ACKs) for our algorithms.

no carrier sensing

no ACKs

In the SINR model the medium access as well as other algorithms are often randomized, thus they are successful with a certain probability. We require the algorithms in this thesis to be correct *with high probability* (*w.h.p.*), i.e. with probability at least $1 - \frac{1}{n^c}$, where n is the number of nodes, and $c \geq 1$ a constant. With such

w.h.p.

results, we can easily transfer local results to global results using a union bound. Let an event $\mathcal{E}_v$ hold locally for each node in the network with probability $1 - \frac{1}{n^2}$ (i.e., w.h.p. with constant $c = 2$) and let $\neg\mathcal{E}_v$ be the complementary event of $\mathcal{E}_v$. We bound the probability of a failure at any node of the network by

$$\Pr(\cup_{v \in V} \neg\mathcal{E}_v) \leq \sum_{v \in V} \Pr(\neg\mathcal{E}_v) \leq n \cdot \frac{1}{n^2} \leq \frac{1}{n^1},$$

which implies that the intersection of the events has probability at least $1 - \frac{1}{n^1}$, implying that the event holds globally w.h.p., with a decreased constant $c = 1$.

2.2.4 *Related Work in the SINR Model*

In this section we give an overview on related work and related problems considered in the SINR model. This section does not replace related work stated in the introduction of each chapter, but tries to give a broader perspective.

Gupta and Kumar initiated the study of the SINR model in the algorithmic community with their seminal work on how the throughput of a multi-hop network scales under both the graph-based protocol model and the more realistic SINR model [60]. They studied the throughput capacity in the average case, which became known as scaling laws in large ad hoc or sensor wireless networks [107]. Studying worst case capacity in wireless ad hoc networks under the SINR was initiated by Moscibroda and Wattenhofer [73]. Since then, a lot of work in the SINR model aimed at understanding fundamental properties of wireless communication in these networks, such as connectivity [8, 68, 73] and capacity [32, 54, 58, 66, 67, 68, 87, 88]. For capacity one tries to select a maximal number of links that are simultaneously feasible in the SINR model. This is related to the so-called scheduling problem, which is the problem of scheduling a given set of links using as few time slots as possible. Using a constant approximation of capacity, a simple greedy algorithm gives a $\mathcal{O}(\log n)$ approximation of scheduling [87]. The scheduling problem has been proven to be NP-complete [56, 86] and hence several approximation algorithms were proposed [35, 62, 64, 69, 88, 126], among which [64, 88] are distributed algorithms. We refer to Goussevskaia et al. [57] for a survey on approximation algorithms (mostly for capacity and scheduling) in the SINR model, although some recent results are not covered. Another interesting line of research is on so-called SINR-diagrams [6, 7, 83], which can be seen as maps of the reception zones of simultaneously transmitting nodes.

Most of the by now referenced works studied fundamental problems in a centralized matter, however, to operate on a wireless sensor networks algorithms must be distributed. Hence we focus on distributed algorithms in the following. The research on local broadcasting in the SINR model was initiated by Goussevskaia, Moscibroda and Wattenhofer [55]. They gave an algorithm that achieves local broadcast in $\mathcal{O}(\Delta \log n)$ time slots if the competition (or alternatively Δ) is known, and $\mathcal{O}(\Delta \log^3 n)$ if not. Several works considered this problem [16, 65, 133, 137]. The related problem of computing a local broadcasting schedule of length $\mathcal{O}(\Delta)$ is

considered by [16, 30, 78]. In this work we consider local broadcasting using arbitrary transmission powers in Chapter 3, and the computation of local broadcasting schedules in Chapter 6. Thus, we defer detailed review of this line of research to Section 3.1 and especially Section 6.1.

Distributed node coloring was first considered by Derbel and Talbi in the SINR model [30], who transferred the MW-Coloring algorithm [104] to this model. Yu et al. consider the problem in the case that the competition (or Δ) is not known in [136]. We consider distributed node coloring algorithms in Chapters 3 and 4 and refer to Section 4.1 for a detailed related work review.

The broadcasting problem is frequently studied in the SINR model [28, 63, 79, 80]. Currently the best results for randomized single source broadcasting (without position information) are obtained by Jurdzinski et al. [80] with a runtime of $\mathcal{O}(D \log^2 n)$ time slots, where D is the diameter of the network, and by Halldórsson, Holzer and Lynch [63] achieving $((D + \log n) \log^{\alpha+1} \Lambda)$ time slots, where Λ is the ratio between the maximal and minimal length of a communication link in the network. Other studied variants are deterministic broadcasting [81] and multi-message broadcasting [63, 134, 135].

Other related problems that are studied in the SINR model are data aggregation [19, 71, 94, 95], in which each node in the network has a message that must be communicated to a specified sink node, maximal independent sets [30, 136] and (connected) dominating sets [78, 117, 138]. Another recent line of research is on using multiple input multiple output (MIMO) techniques that utilize multiple antennas to increase wireless communication efficiency [75, 76]. Finally, Bodlaender and Halldórsson recently proposed a generalization of the geometric SINR model by using a decay space. In the generalization a metricity parameter (intuitively) indicates how heavily the decay of a network deviates from the geometric decay [18]. Additionally they show that results obtained in the SINR model with general metrics can directly be transferred to decay spaces.

2.3 EXPERIMENTS WITH SINALGO

In this section we introduce the simulation framework and basic simulation settings used in the experimental evaluations in Chapters 3 and 5

We conduct our experiments using the current version 0.75.3 of Sinalgo [33], an open-source simulation framework for networks algorithms in Java. Sinalgo has built-in support for a variety of communication and interference models, and is implemented in a modular fashion, making it easy to add customized models or algorithms. The main simulation framework offers both round-based and asynchronous or event-based simulation. Apart from that connectivity, interference, mobility, reliability, distribution, and message transmission models implement a wide variety of settings. We shall briefly introduce the relevant parts of the simulation framework in the following. In the round-based setting, in every time slot each node is considered once. Thus, the node handles all successfully received messages and performs one step. Both the message handling and the step depend on the im-

plementation of the network algorithm. In the event-based simulation, each action of the algorithm must be invoked by a timer. Despite the absence of rounds or slots, we denote the time required for one transmission as a time slot. Mobility is not supported in this setting due to the high number of events and the corresponding updates of all positions (required for example for connectivity and interference computations). Let us now consider the models used in our experiments.

Connectivity: To determine the neighborhood relations we implemented a new model, which calculates the broadcasting range directly based on the SINR parameters used in the simulation. Based on the broadcasting range, the neighbors of a node v are determined as the nodes within this range of v.

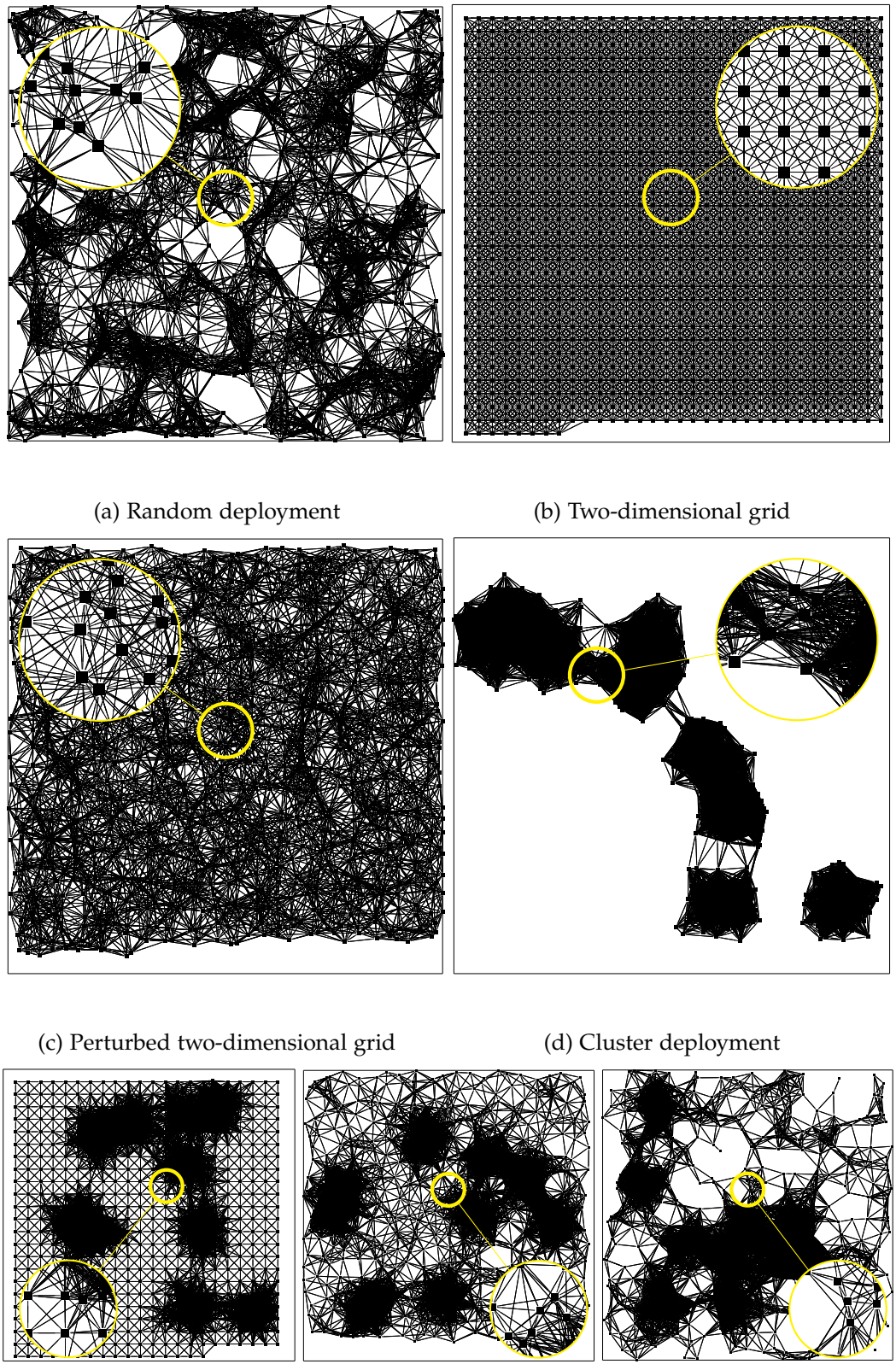
Interference: Successful reception of the nodes transmission is determined by the standard geometric SINR model in all our simulations. A flaw in the SINR-module delivered with Sinalgo in version 0.75.3 drops some transmissions although they are feasible in the SINR model. We show how to correct this in Appendix A.1. For more details on the SINR model itself we refer to Section 2.2.3.

Distribution: We mostly deploy our nodes on a square area of $1000\,\mathrm{m} \times 1000\,\mathrm{m}$ using several distribution strategies. We use the built-in random and grid model, which deploy the nodes uniformly at random and according to a regular two-dimensional grid, respectively. Custom models we use are a perturbed grid, in which each nodes grid position is uniformly at random drawn from a $1\,\mathrm{m}^2$ area centered at the original grid position, and a cluster distribution which distributes all nodes in a predefined number of clusters (we use 10 clusters). Additionally, we use a combination of the cluster model with the other models, in which 50 % of the nodes are distributed according to the cluster model and the remaining nodes according to either the random, grid or perturbed grid model. Our distribution models are illustrated in some example networks in Figure 2.1. To increase comparability of our experiments we use a precomputed set of position files.

Message transmission: Messages are transmitted with a certain probability in each time slot. We usually compute the transmission probability based on a transmission constant, and a parameter depending on the density of the network, which we call transmission parameter. We introduce the parameters used to determine the transmission probability in the next section. Regarding the time required for a message transmission we assume constant time message transmission. As all messages are of size at most $\mathcal{O}(\log n)$ in our algorithms, this fits the algorithms requirements. To ensure that a message can be transmitted in one time slot, which is of length 1, we use a transmission time of 0.999 time slots[2]. We do neither allow simultaneous transmission and reception nor the simultaneous reception of several packets.

Reliability: Throughout our simulations we use so-called reliable transmission, which implies that transmitted messages are received unless they are discarded by the interference model.

2 Despite the transmission time of less than 1, we set 1 as the duration of a time slot in the asynchronous case as well.



(a) Random deployment

(b) Two-dimensional grid

(c) Perturbed two-dimensional grid

(d) Cluster deployment

(e) 50 % of nodes in cluster deployment, 50 % as two-dimensional grid

(f) 50 % of nodes in cluster deployment, 50 % as perturbed grid

(g) 50 % of nodes in cluster deployment, 50 % as random deployment

Figure 2.1: Sample networks illustrating our deployment strategies.

Mobility: Sinalgo supports mobility based on either random waypoints, or random directions. We use the latter model, as it consistently provides a balanced distribution of the nodes on the area, while the random waypoint model would lead to a high concentration of nodes in the center of the deployment area [10].

Our algorithms are implemented in subclasses of the node class, which plugs into the described models and provides standard transmission and reception features.

2.3.1 *Experimental Setup and Transmission Probabilities*

The parameters for the geometric SINR model of interference are set as follows. We use a value of $\alpha = 4$ for the attenuation coefficient, which is assumed to be between 2 for a free field environment and 6 for buildings in practice [114]. We use a threshold of $\beta = 10$ and an environmental noise value of 10^{-9}, which are both within the ranges often reported [123, 139]. We use a uniform transmission power, which is set to $P = 1$ for all nodes, unless described otherwise in the specific experiment. These parameters lead to a transmission range of 100 m, an additional broadcasting range parameter of 2 leads to the broadcasting range of about 84 m, cf. Section 2.2.3. We often use a deployment area of $1000\,\mathrm{m} \times 1000\,\mathrm{m}$ with 1000 nodes. All experiments are conducted using 100 runs with differing random seeds, where randomization is mainly used to decide whether a node transmits or not in a round based on the transmission probability.

To determine the transmission probability the nodes usually know the maximum degree Δ in the network. For initial communication in the SINR model, as for example in the local broadcasting algorithm by Goussevskaia, Moscibroda and Wattenhofer [55], a similar value $\bar{\Delta}$ is used. This value denotes the number of nodes in the so-called proximity region of the node and is asymptotically the same as Δ. We use a parameter txConst to compute the transmission probability as $\frac{\text{txConst}}{\Delta}$ or $\frac{\text{txConst}}{\bar{\Delta}}$. The runtime of local broadcasting using both Δ and $\bar{\Delta}$ is depicted in Figure 2.2 for a range of transmission constants. For this experiment we use 1000 nodes uniformly at random deployed on the standard deployment area of $1000\,\mathrm{m} \times 1000\,\mathrm{m}$. As local broadcasting requires all nodes to transmit one message to their respective neighbors, we measure the number of time slot until the last node successfully transmitted its message to its neighbors as runtime of local broadcasting. We show the results in Table 2.1, and additionally in Figure 2.2 using standard *boxplots*. The boxes range from the first to the third quartile, while the marker in the box represents the median. The whiskers extend to the largest and smallest value within 1.5 times the inner-quartile range. Results not in this range are shown as outlier, marked by a circle.

txConst

boxplot

We observe that using both $\bar{\Delta}$ and Δ results in similar runtimes with Δ achieving slightly faster results. The lowest runtime of 5014 and 4592 time slots are obtained for $\text{txConst} = 0.3$ using $\bar{\Delta}$ and $\text{txConst} = 0.15$ using Δ, respectively. The higher txConst is partially compensated by the higher $\bar{\Delta}$, resulting in similar transmission probabilities. The best runtimes for example are achieved in both cases using an

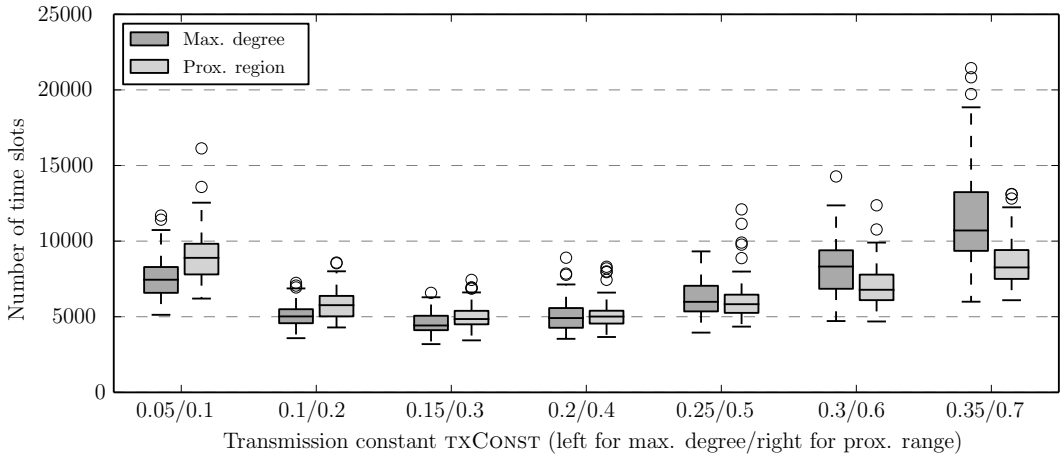


Figure 2.2: Runtime of local broadcasting using Δ (max. degree) and $\bar{\Delta}$ (number of nodes in the proximity region) for several transmission constants. The best average runtimes are achieved using a transmission constant of 0.15 for Δ and 0.3 for $\bar{\Delta}$. txConst divided either by Δ or $\bar{\Delta}$ results in the transmission probability.

Table 2.1: Comparison of local broadcasting using $\bar{\Delta}$ and Δ to determine the transmission probability and duration.

Using $\bar{\Delta}$	txConst	0.1	0.2	**0.3**	0.4	0.5	0.6	0.7
	runtime	9017	5846	**5014**	5113	6066	7036	8585
Using Δ	txConst	0.05	0.1	**0.15**	0.2	0.25	0.3	0.35
	runtime	7580	5104	**4592**	5075	6197	8112	11 495

average transmission probability of 4.11×10^{-3}. Based on the experimental results, the fact that $\bar{\Delta} \in \mathcal{O}(\Delta)$ and the impossibility to distributively compute $\bar{\Delta}$ based only on the network topology or the neighborhood relations, we use Δ for our experiments.

We repeat the experiment using Δ for the remaining distributions and report the best transmission constant txConst along with the maximum and average degree, the resulting average runtime and the resulting duration value in Table 2.2. We observe that the optimal value of txConst varies slightly depending on the deployment strategy, however, this is only marginal. The required changes in the transmission probability for local broadcasting to operate in the different deployments is mostly done automatically by incorporating the maximum degree Δ. Based on the runtime we set the parameter duration, which denotes the number of time slots accounted for local broadcasting for the deployment strategies.

Table 2.2: Parameters that achieve successful local broadcasting in the different distributions. R=Random, G=Grid, PG=PerturbedGrid, C=Cluster

Distribution	R	G	PG	C	C&R	C&G	C&PG
TXCONST	0.15	0.15	0.10	0.30	0.25	0.20	0.20
Maximum degree Δ	36.6	20.0	27.9	182.0	106.0	101.8	100.8
Average degree	20.6	18.6	20.9	93.8	39.4	38.7	39.9
Transmission probability	4.11×10^{-3}	7.5×10^{-3}	3.59×10^{-3}	1.71×10^{-3}	1.46×10^{-3}	2.02×10^{-3}	2.55×10^{-3}
Avg. runtime of a local broadcast	4592	3345	4845	12 903	8062	8183	8089
DURATION	4600	3400	4900	12 900	8100	8200	8100

Part I

ALGORITHMS FOR THE SINR MODEL

3

ARBITRARY TRANSMISSION POWERS IN THE SINR MODEL

In the light of energy conservation and the expansion of existing wireless networks, wireless networks face the challenge of nodes with heterogeneous transmission powers. For more realistic models of wireless communication, however, only few algorithmic results are known. In this chapter we consider nodes with arbitrary, possibly variable, transmission powers in the so-called physical or SINR model.

Our first result is a bound on the probabilistic interference from all simultaneously transmitting nodes on receivers. This result implies that current local broadcasting algorithms can be generalized to the case of non-uniform transmission powers with minor changes. The algorithms run in $\mathcal{O}(\Delta\Gamma^\zeta \log n)$ time slots if the maximum degree Δ is known, and $\mathcal{O}((\Delta + \log n)\Gamma^{2\zeta} \log n)$ otherwise, where Γ is the ratio between the maximum and the minimum transmission range and ζ is the dimensionality of the metric space (e.g., $\zeta = 2$ for the Euclidean plane, cf. Section 3.2). The broad applicability of our result on bounding the interference is further highlighted by generalizing a distributed node coloring algorithm to this setting.

We generalize the so-called MW-coloring [104] algorithm to directed networks in metric space and achieve $\mathcal{O}(\Gamma^\zeta \Delta)$ colors with a runtime of $\mathcal{O}((\Delta + \ell\Gamma^{2\zeta})\Delta\Gamma^\zeta \log n)$ time slots, where ℓ is what we call the length of the longest unidirectional path in the network. Despite the increased complexity due to unidirectional communication links the algorithm is still operational in the asynchronous setting and can be simplified to a maximal independent set (MIS) algorithm.

This chapter is based on joint work with Dorothea Wagner. Preliminary results are published in [46] and as a full version online available [45].

3.1 INTRODUCTION

One of the most fundamental problems in wireless ad hoc networks is to enable efficient communication between neighboring nodes. This problem recently received increasing attention among the distributed computing community, as more refined models of wireless communication became established in algorithms research. Among these models, the so-called physical or SINR (Signal-to-Interference-and-Noise ratio, cf. Section 2.2.3) model is most prominent and promising due to its common use in the engineering literature, e.g. [127]. So far, however, most algorithmic work in the SINR model is restricted to the centralized setting or the

case of uniform transmission powers. In the second case, local broadcasting [55, 65, 133, 137] provides initial communication by enabling each node to transmit one message such that all intended receivers (i.e., neighbors) are able to decode the message.

In this chapter we consider the problem of local broadcasting in the SINR model under arbitrary transmission powers, i.e., each node has its individual, possibly even variable, transmission power. We are the first to consider this setting from an algorithmic perspective. While some distributed node coloring algorithms do consider the transmission powers to be variable they either increase the transmission powers synchronously and thus effectively operate on an uniform power network [30], or differ by a fixed factor of only 3 [136]. One of the few lines of research that leverages non-uniform transmission powers is on link scheduling and capacity maximization [68, 88]. There, however, each node is usually considered to be either transmitter or receiver. If a node has multiple roles it might be forced to adapt its transmission power frequently. On the other hand, the effects of heterogeneous transmission powers are considered in simulation-based studies for example in [49, 111]. The case of unidirectional communication links, which are a result of heterogeneous transmission powers, is also studied frequently [128, 140].

We assume the general setting of a metric space with bounded growth property (cf. Section 3.2), which generalizes the Euclidean space, and the harsh environment of a wireless ad hoc or sensor network just after deployment. In particular, we consider multi-hop networks where the nodes do initially not have any information about whether other nodes are awake, have already started the algorithm or in which phase of the algorithm they are. The only knowledge they may have is an upper bound on the maximum node degree and a rough bound on the total number of nodes in the network. Note that our model does not assume a collision detection mechanism.

Related Work

The study of local broadcasting, and interference in wireless communication in general, has only emerged recently. Especially in classical distributed message passing models such as $\mathcal{LOCAL}$ or $\mathcal{CONGEST}$, the transmission of a message to neighbors is guaranteed, see Section 2.2.1 or [109]. However, this is not the case for wireless networks. Hence interference in general and local broadcasting in particular must be considered in the more realistic SINR model. Goussevskaia, Moscibroda and Wattenhofer [55] were the first to present local broadcasting algorithms in the SINR model. Their first algorithm assumes an upper bound Δ on the number of neighbors to be known by the nodes and solves local broadcasting with high probability in $\mathcal{O}(\Delta \log n)$ time. Their second algorithm does not assume this knowledge and requires $\mathcal{O}(\Delta \log^3 n)$ time slots using a slow-start technique to find the optimal transmission probability. The second algorithm has subsequently been improved by Yu et al. to run in $\mathcal{O}(\Delta \log^2 n)$ [137], and again to $\mathcal{O}(\Delta \log n + \log^2 n)$ [133]. This bound has been matched by Halldórsson and Mitra in [65] using a more robust algorithm (see Section 3.4.1), along with an algorithm that uses free acknowl-

edgements to achieve a time complexity of $\mathcal{O}(\Delta + \log n)$. This result was recently improved by Barenboim and Peleg to $\mathcal{O}(\Delta + \log n \log\log n)$ time slots [16].

Research on distributed node coloring, which is the problem of assigning colors to nodes such that no two neighbors have the same colors (cf. Section 2.1), dates back to the first days of distributed computing almost 30 years ago. Let us focus on algorithms for the SINR model in the following, we provide more detailed related work for distributed node coloring in Chapter 4. An algorithm that colors the network with $\mathcal{O}(\Delta)$ colors in $\mathcal{O}(\Delta \log n)$ time was presented by Moscibroda and Wattenhofer in [103]. However, they assume a graph-based interference model. The algorithm has subsequently been improved in [104] and [120] and transferred to the SINR model by Derbel and Talbi [30] with the same asymptotic bound on the number of colors and runtime as the original algorithm. Yu et al. consider the problem of coloring the network with only $\Delta + 1$ colors in [136] and present algorithms that run in $\mathcal{O}(\Delta \log^2 n)$ time slots and $\mathcal{O}(\Delta \log n + \log^2 n)$ if the nodes transmission power can be tuned by a constant factor.

Contributions

In this work we are the first to consider arbitrary transmission powers in the SINR model, and thus networks with unidirectional links, for the problems of local broadcasting, distributed node coloring and maximal independent set (MIS). Our first contribution, however, is of more general nature and provides an abstract method for bounding the interference in these networks. We prove that transmissions are feasible based on the sum of local transmission probabilities. This result is widely applicable, as verifying that the sum of local transmission probabilities is bounded as required is relatively simple.

Our second result transfers algorithms for local broadcasting presented in [55, 65] to the case of arbitrary transmission power assignment. We achieve local broadcasting in $\mathcal{O}(\Delta\Gamma^\zeta \log n)$ time slots if the maximum degree Δ is known and $\mathcal{O}((\Delta + \log n)\Gamma^{2\zeta} \log n)$ otherwise, where Γ is the ratio between the largest and the smallest transmission range. Note that these bounds match those for the uniform case if the algorithms are run on such networks. Additionally we discuss the case of variable transmission powers, which achieves similar bounds, but allows nodes to change the transmission power in each time slot instead of fixing it for a complete execution of local broadcasting.

Finally we give an algorithm for distributed node coloring in these harsh environments. The algorithm is based on an algorithm by Moscibroda and Wattenhofer [104], which was adapted to the uniform SINR model by Derbel and Talbi [30]. Fundamental changes to the algorithm itself are required due to the increased complexity of the network structure, such as unidirectional communication links. We introduce a new network parameter ℓ that measures the length of the longest simple unidirectional path in the partially directed network and prove that our distributed node coloring algorithm colors the network with $\mathcal{O}(\Gamma^\zeta \Delta)$ colors in $\mathcal{O}((\Delta + \ell\Gamma^{2\zeta})\Delta\Gamma^\zeta \log n)$ time slots. By simplifying the algorithm we obtain an algorithm that computes an MIS in $\mathcal{O}((\Delta + \ell)\Delta\Gamma^\zeta \log n)$ time slots. Note that our

algorithms are fully operational under asynchronous node wake-up and match the runtime of the algorithms designed for the uniform case apart from a factor of Δ, as ℓ and Γ are 1 in this case.

Outline

In the following section we introduce the notation used in this chapter. We establish a bound on the probabilistic interference of nodes outside the proximity region based on the sum of transmission probabilities from within each transmission region in Section 3.3. Afterwards, we apply this result to previous results on local broadcasting and transfer current algorithms to the more general model in Section 3.4. The applicability of our results is highlighted in Section 3.5, as we consider the problem of distributed node coloring and generalize a well-known algorithm from the case of uniform transmission powers. We conclude in Section 3.6 with some final remarks.

3.2 PRELIMINARIES

We consider a wireless network consisting of n nodes that are placed arbitrarily in a metric space of dimension ζ with bounded growth property (see below) and distance function $\mathrm{dist}(\cdot,\cdot)$. We assume that all nodes in the network know their ID and an upper bound $\tilde{n}$ on n, with $\tilde{n}$ polynomial in n. As the upper bound on n influences our results only by a constant factor, we usually write n even though only the upper bound $\tilde{n}$ may be known by the nodes. Also, we assume that nodes know lower and upper bounds on the transmission powers or the transmission ranges. This assumption is realistic, as lower bounds for reasonable minimal transmission ranges can be computed while upper bounds (for specified frequencies) are often regulated by public authorities.

We shall briefly repeat some notation for the SINR model introduced in Section 2.2.3 as we use heterogeneous transmission power and a general metric space in this chapter. A transmission from node v to node w is successful iff the SINR condition holds: $\frac{\frac{\mathrm{P}_v}{\mathrm{dist}(v,w)^\alpha}}{\sum_{u\in\mathcal{I}}\frac{\mathrm{P}_u}{\mathrm{dist}(u,w)^\alpha}+\mathrm{N}} \geq \beta$, where P_v (P_u) denotes the transmission power of node v (u), α is the attenuation coefficient, $\beta \geq 1$ is a hardware-defined constant, N is the environmental noise and $\mathcal{I}$ is the set of nodes transmitting simultaneously with v. Generalizing the definitions of Section 2.2.3 we define the *transmission range* of a node v to be $r_v^T = (\frac{\mathrm{P}_v}{\mathrm{N}\beta})^{1/\alpha}$. The global maximum transmission range in the network is denoted by $\bar{R}$, the minimum range by $\underline{R}$ and the ratio between $\bar{R}$ and $\underline{R}$ by $\Gamma = \frac{\bar{R}}{\underline{R}}$. Due to the SINR constraints, a node v cannot reach another node w which is located at the maximum transmission range of v, as soon v transmits simultaneously with any other node in the network. As having only one simultaneous transmission in the network is not desired, we use a parameter $\delta > 1$ to determine the distance up to which the nodes messages should be received. We call this distance the *broadcasting range* $r_v^B = (\frac{\mathrm{P}_v}{\delta\mathrm{N}\beta})^{1/\alpha} = \delta^{-1/\alpha} r_v^T$ and the region within this range from v the broadcasting region B_v. We denote the maximum

$\bar{R}$ $\underline{R}$ Γ

number of nodes within the broadcasting range r_v^B of any v by Δ. As the maximum number of nodes in a nodes transmission range is in $\mathcal{O}(\Delta)$, we sometimes denote this value by Δ as well for simplicity. Note that Δ is known by the nodes only if stated with the corresponding algorithms. We define the *proximity region* around v as the area closer than $k\bar{R}$ to v for a constant $k \geq 3$. Let $\bar{R}_k(v)$ denote the set of nodes in the proximity region around v. An illustration of the various ranges and regions around node v is depicted in Figure 3.1. Δ

We denote the transmission probability of a node v by p_v. The probabilistic interference $\Psi_{\mathcal{I}}^u = \sum_{w \in \mathcal{I}} \frac{p_w P_w}{\text{dist}(w,v)^\alpha}$ is the expected interference experienced at node u from the nodes $w \in \mathcal{I}$, which transmit with probability p_v and transmission power P_v. Also, recall that two nodes u, v are *independent* if neither u is in v's broadcasting range nor v in u's.

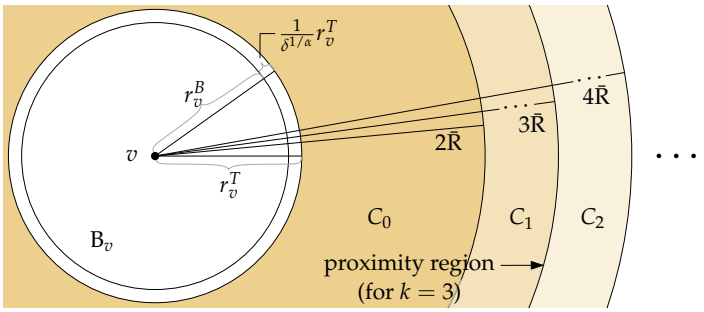


Figure 3.1: The different ranges and regions of v along with the rings around v.

Let us now consider the *metric space* with distance function $\text{dist}(\cdot,\cdot)$ and dimension ζ. We denote the set of nodes in a ball of radius r around a node v by $B(v,r) = \{w \mid \text{dist}(v,w) \leq r\}$. Let $\chi(r_1, r_2)$ be the number of nodes required to cover a ball of radius r_1 with balls of radius $r_2 \leq r_1$. Then the *bounded growth property* holds if for a constant $c > 0$ and an arbitrary radius $r \geq 0$ it holds that $B(v, c \cdot r)$ can be covered by $\mathcal{O}(c^\zeta)$ balls of size $B(v,r)$, i.e. $\chi(c \cdot r, r) = \mathcal{O}(c^\zeta)$. Note that this property holds, for example, for the Euclidean plane, where $\chi(c \cdot r, r) \leq \frac{2\pi}{3\sqrt{3}} \cdot (c+2)^2 = \mathcal{O}(c^2)$[55, Fact 3.3]. This property ensures that the number of independent nodes around a node v is bounded. *metric space* ζ $\chi(\cdot,\cdot)$

Lemma 3.1. *The number of independent nodes in a ball of radius r_1 around v is upper bounded by $\chi(r_1, \underline{R}/2)$*

Proof. Let us consider an arbitrary node w with $\text{dist}(v,w) \leq r_1$. It holds that within distance $\underline{R}$ there cannot be another independent node, as otherwise there must be a communication link between these two nodes. Thus, for balls of radius $\underline{R}/2$ around each node in the ball of radius r_1, these balls do not intersect. It follows that there can be at most $\chi(r_1, \underline{R}/2)$ leader nodes in a maximal transmission range. □

It holds that $\chi(\bar{R}, \underline{R}/2) = \mathcal{O}(\Gamma^\zeta)$. In order to simplify our notation, we assume the constant hidden in $\mathcal{O}(\Gamma^\zeta)$ to be 1. In particular, we say that $\chi(\bar{R}, \underline{R}) \leq \Gamma^\zeta$, $\chi(k\bar{R}, \underline{R}/2) \leq \Gamma^\zeta$ and also that the number of independent nodes required to cover the nodes in "rings" $(B(v,(c+1)\cdot\bar{R}) - B(v, c\cdot\bar{R}))$ around v (cf. the definition of

rings in Section 3.3) is upper bounded by $c\Gamma^{\zeta}$. Note that this simplification does not affect our asymptotic bounds.

We do not require a global clock or synchronized time slots in our algorithm. Decent local clocks are sufficient, while time slots are only required in the analysis. Let us conclude this section by stating a mathematical facts used in the analysis.

Fact 3.2 ([82]). *Given a set of probabilities $p_1, \ldots, p_n$ with $\forall i : p_i \in [0, \frac{1}{2}]$, it holds that:*

$$\left(\frac{1}{4}\right)^{\sum_{k=1}^{n} p_k} \leq \prod_{k=1}^{n} (1 - p_k)$$

3.3 BOUNDING THE INTERFERENCE

In contrast to other models for interference in wireless networks, such as the protocol model, the SINR model captures the global aspect of interference and reflects that even interference from far-away nodes can add up to a level that prevents the reception of transmissions from relatively close nodes. To ensure that a given transmission can be decoded by all nodes within the broadcasting range, one usually proves that reception within a certain time interval is successful w.h.p. Such a proof can be split in two parts.

1. The probability that a node transmits within a proximity region around a sender is constant.

2. Let $P_{\text{2high}}(v)$ be the event that the interference from all nodes outside of the proximity region of v on nodes in the broadcasting region of v is too high. Show that $P_{\text{2high}}(v)$ has constant probability.

We shall follow this scheme by considering the transmission of an arbitrary node, and prove that both conditions hold with constant probability in each time slot. Hence a local broadcast is successful with high probability after a sufficient number of time slots.

In order to make the result general and applicable to many different settings, we make only one very general assumption. In particular we assume the sum of transmission probabilities from within a broadcasting region to be bounded by a constant. This is very common and allows us to apply the analysis from this section in the following Sections 3.4 and 3.5 to generalize algorithms designed for the uniform transmission powers case to the more general case of arbitrary transmission powers considered in this chapter[1].

Definition 3.3. *Assume a network of n nodes with at most Δ nodes in each transmission region. Let γ be the upper bound on the sum of transmission probabilities from within a transmission range.*

γ

1 Note that we can directly apply the result presented in this section to many algorithmic results in the SINR model. The algorithms themselves, however, often rely on bidirectional communication links and thus cannot be translated to the more general case directly.

In this section we assume the upper bound on the sum of transmission probabilities from within each transmission region to be

$$\gamma := \frac{\delta - 1}{\beta \Gamma^{\zeta} \sum_{i=1}^{\infty} \frac{1}{i^{\alpha-1}}}. \tag{3.1}$$

This bound can be realized, for example by requiring all nodes to transmit with probability γ/Δ. Another option is the so-called slow-start technique, cf. Section 3.4.1. The constant is carefully selected to help bound the interference from all simultaneously transmitting nodes in the network in the proof of Theorem 3.6. As $\gamma \leq 1$ may not be true for a large δ, we use $\gamma := \frac{1}{\beta \Gamma^{\zeta} \sum_{i=1}^{\infty} \frac{1}{i^{\alpha-1}}}$ if otherwise $\gamma > 1$. Let us now prove a bound on the probability that a close-by node transmits, which is required for the main theorem of this section.

Lemma 3.4. *Assume the sum of transmission probabilities from within a transmission range is bounded by γ. For an arbitrary node v the probability that no node in the proximity region of v transmits in a given time slot is at least $1/4$.*

Proof. The probability that no node of $\bar{R}_k(v)$ transmits in a single time slot (apart from possibly v) is at least

$$P_{\text{none}}^{\bar{R}_k(v)} \geq \prod_{u \in \bar{R}_k(v)} (1 - p_u) \geq \left(\frac{1}{4}\right)^{\sum_{u \in \bar{R}_k(v)} p_u} \geq \left(\frac{1}{4}\right)^{\Gamma^{\zeta} \cdot \gamma} \geq \left(\frac{1}{4}\right),$$

where the second inequality holds due to Fact 3.2, the third inequality since the maximal number of independent nodes within distance $k\bar{R}$ of v is upper bounded by Γ^{ζ} by considering the number of balls of radius $\underline{R}/2$ required to cover a ball of radius $k\bar{R}$, and the bound γ on the sum of transmission probabilities from within each transmission region. The last inequality holds since $\Gamma^{\zeta} \cdot \gamma \leq \frac{1}{\beta \sum_{i=1}^{\infty} \frac{1}{i^{\alpha-1}}} < 1$. □

Let us now consider nodes that are not in the proximity region of the transmitting node. To bound the interference originating from these nodes, we use rings around the transmitting node and bound the probabilistic interference from within each ring. Note that although our definition of the proximity region and rings differ, similar arguments are made, for example, in [55, 65].

Definition 3.5. *For a node v, the* ring C_i^v, *$i \geq 0$, is defined as the set of nodes with distance at least $(i+1) \cdot \bar{R}$ and at most $(i+2) \cdot \bar{R}$. For a ring C_i^v, the* extended ring C_{i+}^v *is defined as the set of nodes with distance at least $i \cdot \bar{R}$ and at most $(i+3) \cdot \bar{R}$.*

ring C_i^v

Note that for a ring C_i^v, the extended ring C_{i+}^v is defined such that the nodes in the transmission region of an arbitrary node $w \in C_i^v$ are contained in C_{i+}^v. If it is clear to which node v the rings refer, we write C_i and C_{i+} for brevity. An illustration of the concept of rings in our context is depicted in Figure 3.1.

Theorem 3.6. *Let the sum of transmission probabilities from within each transmission region of the network be upper bounded by γ. Given a node v of the network, the probabilistic interference $\Psi_{\bar{R}_k(v)}^u$ from nodes outside the proximity region of v on nodes $u \in B_v$ is upper bounded by $(\delta - 1)N$.*

Proof. Let us first bound the interference from a single ring C_i on a node $u \in B_v$. By a simple geometric argument and the simplification described in Section 3.2 it holds that the maximal number of independent nodes in the extended ring C_{i+} is at most $i \cdot \Gamma^\zeta$. By combining this number with the sum of transmission probabilities from within each broadcasting region, we can bound the interference from the nodes in C_i on u. As each node in the ring C_i has distance greater than $i \cdot \bar{R}$ from u, it follows that the probabilistic interference $\Psi^u_{C_i}$ caused by nodes in C_i on u is at most

$$\Psi^u_{C_i} \leq \sum_{w \in C_{i+}} \frac{p_w \mathrm{P}_w}{(i\bar{R})^\alpha} \leq \frac{i \cdot \Gamma^\zeta \gamma\beta\mathrm{N}}{i^\alpha} \cdot \left(\frac{\bar{R}}{\bar{R}}\right)^\alpha \leq \frac{\gamma\beta\mathrm{N}\Gamma^\zeta}{i^{\alpha-1}}.$$

Summing over all rings it follows

$$\Psi^u_{\bar{R}_k(v)} \leq \sum_{i=k-1}^{\infty} \Psi^u_{C_i} \leq \gamma\beta\mathrm{N}\Gamma^\zeta \sum_{i=1}^{n} \frac{1}{i^{\alpha-1}} \leq (\delta - 1)\mathrm{N},$$

where the second inequality holds by inserting the bound on $\Psi^u_{C_i}$ and the fact that there are at most n non-empty rings (empty rings with a lower i potentially substitute for higher rings that contain a node). The last inequality follows from the upper bound on γ, stated in Equation 3.1. □

3.4 LOCAL BROADCASTING

In the previous section we have shown how to bound the probabilistic interference from simultaneously transmitting nodes based on an upper bound on the sum of transmission probabilities from within each broadcasting region. Such bounds are known for many algorithms in the case of uniform transmission powers, and hence we can plug our results into a large body of related work and transfer results with minimal additional effort to the case of arbitrary but fixed transmission powers. The results hold in any metric space with bounded growth property, although the original analysis is often restricted to the Euclidean plane. In the following section we briefly state our results regarding local broadcasting along with proof sketches as required. In Section 3.4.2 we discuss our results regarding variable transmission powers.

3.4.1 *Arbitrary but Fixed Transmission Powers*

The current results on local broadcasting with the knowledge of Δ are based on transmitting with a fixed probability in the order of $1/\mathcal{O}(\Delta)$ for a sufficient number of time slots in $\mathcal{O}(\Delta \log n)$, while results that do not assume the maximum degree Δ to be known are usually based on a so-called slow-start mechanism. Let us first consider the case in which each node knowns the maximum degree Δ.

With knowledge of the maximum degree Δ

Using the result on local broadcasting by Goussevskaia, Moscibroda and Wattenhofer [55], it is easy to show that local broadcasting can be realized in $\mathcal{O}(\Delta\Gamma^{\zeta}\log n)$ time slots by simply adapting the transmission probability to our requirements.

Theorem 3.7. *Let the transmission probability of each node be $p = \gamma/\Delta$, and $c > 1$ an arbitrary constant. A node v that transmits with probability p for $8(c/p)\log n = \mathcal{O}(\Delta\Gamma^{\zeta}\log n)$ time slots successfully transmits to its neighbors w.h.p.*

Proof. Since the transmission probability is chosen such that the sum of transmission probabilities from within each proximity range is at most γ, we can directly apply Theorem 3.6. Using the theorem, combined with the standard Markov inequality, the probability that the interference from nodes outside of the proximity region is too high (i.e., higher than $(\delta-1)\mathrm{N}$) is less than $1/2$. Lemma 3.4 states that the probability that no node within the proximity range of a node transmits is greater than $\frac{1}{4}$. Combining both probabilities with the transmission probability p implies that the probability of a successful broadcast is at least $p/8$ in each time slot. Thus, transmitting for $8(c/p)\log n$ time slots results in a successful local broadcast with probability at least $1-\frac{1}{n^c}$. □

Without knowledge of Δ

Let us now consider the case that the nodes are not given a bound on the maximum degree Δ. In contrast to the previous algorithm for local broadcasting, the "optimal" transmission probability is initially unknown.

In order to create local broadcasting algorithms for this model, a slow start mechanism can be used [55, 65, 133, 137]. Using such a mechanism each node starts with a very low transmission probability in the order of $1/n$ and doubles the probability until a certain number of transmissions are received, and the probability is reset to a smaller value. With such a mechanism, local broadcasting in the (uniform-powered) SINR model can be achieved in $\mathcal{O}(\Delta\log n+\log^2 n)$ [65, 133]. Although different forms of the slow start mechanisms are used they reset the transmission probabilities such that the sum of transmission probabilities in each transmission region can be upper bounded by a constant.

Let us now consider the algorithm of Halldórsson and Mitra, described in [65]. We adapt the algorithm so that local broadcasting provably works with high probability in the more general model considered in this chapter. This is done by modifying the maximum transmission probability to be $\gamma/(16\Gamma^{\zeta})$ instead of $1/16$ by simply changing Line 7 of Algorithm 1 in [65] from $p_y \leftarrow \min\{\frac{1}{16}, 2p_y\}$ to $p_y \leftarrow \min\{\frac{\gamma}{16\Gamma^{\zeta}}, 2p_y\}$. This small change allows us to bound the sum of transmission probabilities as required.

Lemma 3.8. *Let $\mathcal{N}$ be a network with arbitrary transmission power assignment, asynchronous node wake-up and let all nodes execute Algorithm 1 from [65] with maximal transmission probability set to $\gamma/16\Gamma^{\zeta}$. Then the sum of transmission probabilities from within each proximity region is upper bounded by $\gamma/2$.*

Proof. The corresponding result is proven in [65, Lemma 2]. The proof of Lemma 2 as stated holds only for the broadcasting range corresponding to $\underline{R}$, and states that the sum of transmission probabilities from such a broadcasting range is upper bounded by $\gamma/2\Gamma^\zeta$. However, Lemma 2 can be extended to the broadcasting range of an arbitrary node v. For an arbitrary node v the sum of the transmission probabilities in r_v^T is at most

$$\sum_{w \in r_v^T} p_w \leq \Gamma^\zeta \frac{\gamma}{2\Gamma^\zeta} \leq \frac{\gamma}{2},$$

where the first inequality follows from Lemma 3.1, which bounds the number of independent nodes in r_v^B, and the upper bound on the sum of transmission probabilities from within minimal broadcasting ranges (cf. [65, Lemma 2] with modified maximal transmission probability). □

By combining this result with Theorem 3.6, Lemma 3.4, and a similar argumentation as in the case of known Δ, the transmission is successful at least once with high probability. The correctness of the algorithm follows with the original argumentation in [65]. Using the modified Algorithm 1 from [65], we get for the more general case of arbitrary transmission power assignment.

Theorem 3.9. *There exists an algorithm for which the following holds w.h.p.: Each node v successfully performs a local broadcast within $\mathcal{O}((\Delta + \log n)\Gamma^{2\zeta} \log n)$.*

The results presented in this section generalize the corresponding results for the uniform case to the case of arbitrary transmission powers. As $\Gamma = 1$ in the uniform case, the original bounds on the runtime are matched with our more general analysis.

Remark: Note that the local broadcasting algorithm by Yu et al. [133] has the same runtime guarantees as the algorithm by Halldórsson and Mitra [65], but was proposed slightly earlier. However, their algorithm cannot be transferred to the case of arbitrary transmission powers as it relies on bidirectional communication to operate. Specifically, their algorithm computes an MIS, acquires information about dominated nodes and then assigns transmission intervals to the dominated nodes.

3.4.2 *Variable Transmission Powers*

To achieve a local broadcast with high probability, the transmission power of a node is required to be fixed for at least one full round of the respective local broadcasting algorithm. In this section, we consider a more general setting and allow the nodes to change the transmission power for each time slot. As it is not initially clear which nodes should be considered as intended receivers in such a setting, our result states the achieved broadcasting range, based on the number of times certain transmission power levels were exceeded within the considered time interval. Note that we assume Δ to be known to the nodes in this section. We shall now briefly discuss the required notation.

We consider the time slots in an interval $[1,t]$. For the case of multiple time intervals, a transmission power of 0 can be added to fill the gaps between the intervals. Let $\{0, \underline{R} \hat{=} \mathrm{P}_v^{[0]}, \mathrm{P}_v^{[1]}, \ldots, \mathrm{P}_v^{[k]}\}$ be the set of transmission powers used by v, such that $\mathrm{P}_v^{[j]} < \mathrm{P}_v^{[j+1]}$ for $j = 0, 1, \ldots, k-1$. We denote the number of time slots v used a transmission power of at least $\mathrm{P}_v^{[j]}$ by $T_v^{[j]}$. Let $r_v^{[j]}$ be the broadcasting range corresponding to $\mathrm{P}_v^{[j]}$ for the remainder of this section.

Theorem 3.10. *Let all the nodes in the network transmit with probability at most $p = \gamma/\Delta$ and a variable transmission power in the range between $\underline{R}$ and $\bar{R}$. Let v be an arbitrary node that transmits with probability p and variable transmission powers during the interval $[1,t]$. For j maximal such that $T_v^{[j]} > 8(c/p)\log n$, all nodes closer to v than $r_v^{[j]}$ received v's message w.h.p. for an arbitrary constant $c > 1$.*

Proof. Let us first note that the transmission power of 0 may be implemented by setting the transmission probability for these time slots to 0 and ignoring them towards $T_v^{[j]}$. Thus $\underline{R} \neq 0$ and we can apply the results of Section 3.3. We shall now prove the theorem.

Let j be maximal such that $T_v^{[j]} > 8(c/p)\log n$. It holds that v transmits with probability p and transmission power at least $\mathrm{P}_v^{[j]}$ in at least $8(c/p)\log n$ time slots. Let us consider such a time slot i. As the sum of transmission probabilities from within each proximity range is obviously bounded by γ, we can apply our method to bound the interference. It holds due to Theorem 3.6 and Lemma 3.4 that a message transmitted by v in time slot i is received by nodes closer to v than $r_v^{[j]}$ with probability at least $1/8$. Combined with the transmission probability p and considered over $8(c/p)\log n$ time slots, this results in a success probability of at least $1 - \frac{1}{n^c}$ with an argumentation similar to that in the proof of Theorem 3.7. □

This shows that local broadcasting can also be achieved using variable transmission powers.

3.4.3 *On the Γ^ζ Factor*

We argue in this section that for the currently used analysis scheme for local broadcasting algorithms a runtime of $\Omega(\Gamma^\zeta\Delta)$ is inevitable; however, we do not establish a lower bound on the runtime of local broadcasting in networks with arbitrary transmission powers (apart from the trivial $\Omega(\Delta)$ bound by [55]).

The dependence on Γ is due to the fact that the analysis of all current local broadcasting algorithms [55, 65, 133, 137] is based on the property that with considerable probability, a node v must be the only node transmitting from within some proximity region of v. The number of nodes transmitting from the proximity region is bounded by the maximal number of nodes in an independent set in the region and the number of neighbors those independent nodes have.

We depict such a scenario for the Euclidean plane in Figure 3.2. Assume that all nodes transmit with probability $1/\Delta$ and let v transmit in time slot t. Then the expected number of nodes that transmit in the proximity area in the worst

case is $\Delta\Gamma^2$. Thus, the resulting sum of transmission probabilities from within the considered broadcasting range is $\Delta\Gamma^2 \cdot /\Delta = \Gamma^2$, which is more than current techniques can handle. If we reduce the transmission probability to $1/(2\Gamma^2\Delta)$, the sum of transmission probabilities from within the broadcasting range is $1/2$, thus we can achieve constant probability for the event that no (other) node in the proximity area transmits in a time slot at the cost of having the multiplicative factor of Γ^2 in the runtime.

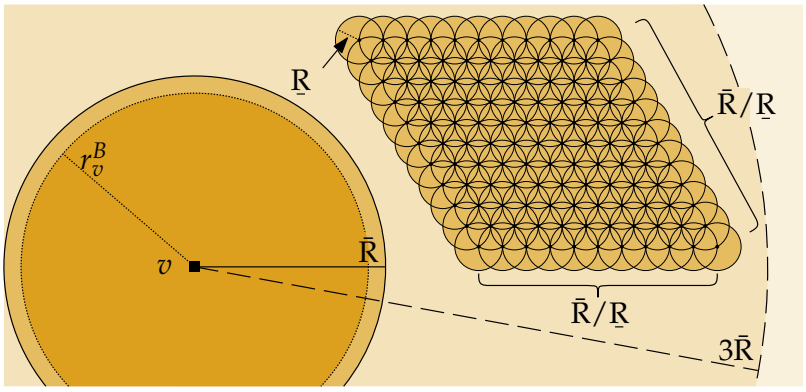


Figure 3.2: The square node v has maximum transmission range, while the remaining nodes, depicted by solid disks with finer circles, have minimum transmission range. It holds that all remaining nodes are independent in the Euclidean plane, thus the independent set is of size $\Theta(\Gamma^2)$.

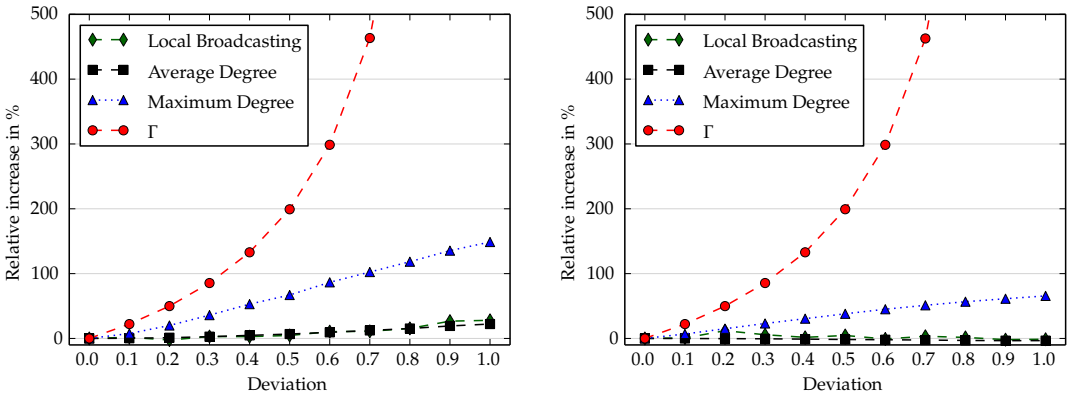
3.4.4 *Experimental Evaluation*

In this section we evaluate the runtime of local broadcasting in simulations to study the influence of non-uniform transmission power on the runtime of local broadcasting. Before diving into the experiments, let us first recall the main parameters of our simulation environment. We use the network simulator Sinalgo [33] as described in Section 2.3. The used SINR parameters are $\alpha = 4$, $\beta = 10$, $N = 1^{-9}$ and a default transmission power of $P = 1$, which results in a broadcasting range of 84 m. In our experiment we use a square area of 1000 m $\times$ 1000 m to deploy 1000 nodes according to pre-computed deployment schemes. We simulate each instance for 100 runs and report average values.

To simulate non-uniform transmission powers, we select each nodes transmission power uniformly at random from a fixed range around the default transmission range. We consider increasing intervals around the default transmission range, varying from the fixed range of 84 m to the range of $[0\,\text{m}, 168\,\text{m}]$. We denote this increase in the available transmission ranges by *deviation*, and increase it in steps of 10 % in each direction. The maximum available range of $[0\,\text{m}, 168\,\text{m}]$ corresponds to 0 % and 200 % of the default value and has a deviation of 100 %, or simply 1. The realized Γ value, which is the ratio between the maximal and minimal transmission range in the network, along with the average and maximum degree and the average

deviation

runtime of one round of local broadcasting and is shown in Figure 3.3 for random and cluster deployment.



(a) Random deployment of nodes

(b) Cluster deployment of nodes

Figure 3.3: The relative change of Γ, compared to the change in runtime, and the average and maximum degree for an increasing deviation, i.e., an increasing range of available transmission ranges. We illustrate the increase relative to the values for deviation 0.

We observe that, despite the rapid increase in the Γ value, the increase in the runtime does not, or at most marginally depend on that value. Despite the fact that the cluster deployment somehow mimics the case which leads to the Γ factor in our analysis (at least to some degree), we cannot conclude any correlation between the Γ factor and the runtime even under this deployment scheme.

3.4.5 *Discussion*

In Section 3.4.1 we applied the bound on the probabilistic interference from outside the proximity region, which we established in Section 3.3, to generalize existing local broadcasting algorithms to the case of arbitrary transmission powers. This provably allows nodes to achieve initial communication despite heterogenous transmission powers in the SINR model. We proposed a model for variable transmission powers in Section 3.4.2 that guarantees success, based on the number of time slots transmitted with certain transmission powers. As the bound on the interference is based on geometric arguments, the variance in the transmission ranges leads to a multiplicative factor of Γ in the runtime of local broadcasting. We argued that current proof techniques require this factor, although our experiments show that the runtime does at most marginally depend on the Γ factor.

3.5 DISTRIBUTED NODE COLORING AND MIS

In this section we demonstrate the applicability of our results to a distributed node coloring algorithm designed for the uniform SINR model with known Δ. We generalize the so-called MW-Coloring algorithm by Moscibroda an Wattenhofer [104],

which has been transferred to the SINR model by Derbel and Talbi [30], to the case of heterogenous transmission powers. We use techniques of the previous sections to achieve feasible communication and additionally tackle challenges that arise with the potentially directed communication graph of the heterogenous network. Distributed node coloring is a fundamental problem in wireless networks, as a node coloring can be used to compute a schedule of transmissions by assigning each color to a different time slot. Thus, efficient transmissions based on a TDMA schedule can be reduced to a node coloring. The algorithm we consider computes a node coloring that ensures that two nodes with the same color cannot communicate directly. This does not necessarily result in a transmission schedule that is feasible in the SINR model already, however, one can use additional techniques like those described in Chapter 6 to transform such a node coloring to a local broadcasting schedule that is feasible in the SINR model.

Our main result in this section is a distributed $\mathcal{O}(\Gamma^{2\zeta}\Delta)$-coloring algorithm with a runtime of $\mathcal{O}((\Delta + \ell\Gamma^{2\zeta})\Delta\Gamma^{\zeta}\log n)$ time slots, where ℓ is the length of the longest directed path (we provide a definition in the following section). The algorithm is fully operational in networks of arbitrary transmission powers.

3.5.1 *Directed Communication Graphs*

Let us first define some notation required for the coloring problem in directed networks. For two nodes $v, u \in V$ we say that there is a *communication link* from v to u if u is in the broadcasting region of v. We say that there is a *unidirectional* communication link from v to u if there is a communication link from v to u, but not from u to v. If both communication links are available we say that it is *bidirectional*. Recall that two nodes are *independent* if there is no communication link between them, and a set is independent if every two nodes in the set are pairwise independent. We call a node coloring *valid* if each color forms an independent set.

unidirectional

bidirectional

Before stating the algorithms, we shall characterize the communication graph implied by arbitrary transmission powers in the SINR model. Obviously, it is still a disk graph as the geometric SINR model forms disk-shaped broadcasting regions, however, it is not a unit disk graph as it is in the uniform case. Additionally, there are two main characteristics that are introduced by directed communication links and are relevant for graph-based algorithms in this setting. First, unidirectional communication links can form long directed paths. This is formalized in the following definition.

Definition 3.11. *Given a network N and the induced communication graph $G = (V, E)$. Let G' be the graph that remains after deleting all bidirectional edges from G. The* longest directed path *in the network is defined as the longest path in G'. We denote the length of the longest directed path in a network by ℓ.*

longest directed path

The second characteristic is that such directed paths cannot form a directed circuit in the network. This holds since in any circuit in the communication graph, there must be at least one bidirectional communication link. Consider for example

a directed path consisting of the nodes $(v_1, \ldots, v_\ell)$. It holds that the broadcasting range decreases monotonically, i.e., $r^B_{v_i} \geq r^B_{v_{i+1}}$ for $i = 1, \ldots, \ell - 1$. If a node v_i can directly be reached from v_j with $i \leq j$, there must also be a communication link from v_j to v_i as $r^B_{v_i} \geq r^B_{v_j}$.

3.5.2 *The Coloring Algorithm*

Let us now state our coloring algorithm. The core of our algorithm is based on the coloring algorithm by Moscibroda and Wattenhofer designed for unstructured radio networks in [104]. It has been adapted to the case of uniform transmission powers in the SINR model by Derbel and Talbi in [30]. In this section we extend the algorithm to work in the case of arbitrary transmission power assignments. A state diagram of the algorithm can be found in Figure 3.4, and pseudocode of the states of the algorithm can be found in Algorithms 3.1 - 3.6. Note that some technical details regarding the asynchronous wake-up of nodes and its impact on the algorithm are omitted here and in the state diagram for simplicity, but are discussed in Section 3.5.8.

Let us now give an overview of the algorithm. The algorithm starts with a three-way handshake protocol, which we call neighborhood learning. This allows each node to learn which of its incoming edges are effectively bidirectional communication links. After this learning stage, we allow a node v to participate in the (modified) coloring algorithm only if v is not *dominated*, i.e., if there is no other uncolored node w such that w reaches v but v does not reach w.

dominated

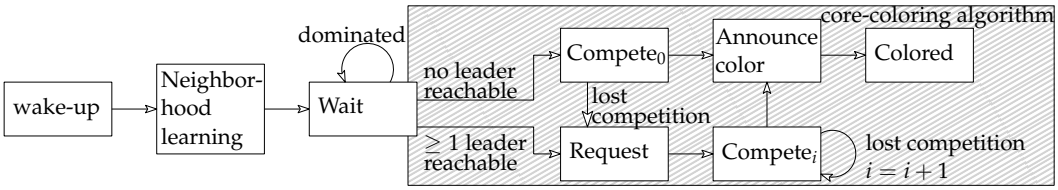


Figure 3.4: State diagram of our coloring algorithm. The core-coloring algorithm is mostly based on the MW-Coloring algorithm [30, 104]. The state diagram is slightly simplified and does not consider continuous neighborhood learning and fall-backs due to awaking nodes (see Section 3.5.8).

We call the main coloring states of the algorithm (Compete, Request, Announce and Colored), which are based on the MW-Coloring algorithm [30, 104], the *core-coloring algorithm* in this section. Once v is not dominated, it starts executing the core-coloring algorithm with a listening phase that is long enough to ensure that v knows the current status of all other nodes that are awake and can reach v. Afterwards, if there is a leader w to which bidirectional communication is possible, v enters the request state, and requests a color from w. After w answers the request by assigning a color j, v tries to verify the assigned color j. If this is not successful (i.e., v loses against another node that reaches v and also competes for j), v increases j by 1 and retries. Once v is successful, it announces its success so that

core-coloring algorithm

the nodes that hear v are informed about v's status. Hence they know that v is about to color itself with color j.

If v is not aware of a leader that can bidirectionally communicate with v, v competes for the leader status itself. If this is not successful, v enters the request state (and proceeds as above) as v must now be aware of a leader with bidirectional communication link to v. Note that v does not lose the leader competition against leader nodes that dominate v, as v cannot request a color from them. If v is successful in becoming leader, it selects a free leader color and announces its choice so that all nodes that can be reached by v are informed. After the announcement phase, the node colors itself. It will now periodically transmit its color and serve color requests as they arrive.

Our presentation of some parts of the algorithms is based on the presentation in [30]. The pseudocode for the algorithm can be found in algorithms 3.1 - 3.6.

Algorithm 3.1: Distributed node coloring for node v in networks with non-uniform transmission powers

```
1  [ In(v), Out(v)] ← Neighborhood learning()
2  Let the node store each received node & color in C_taken throughout the
   algorithm
3  Listen for κ_s time slots
4  while c_v = −1 do                                          // This is state Wait
5      if (In(v)\Out(v))\C_taken = ∅ then                     // If v is not dominated
6          if In(v)∩Out(v)∩C_taken ≠ ∅ then // v can bidirectionally communicate to leader
7              Request(w) for a w ∈ In(v)∩Out(v)∩C_taken // Transition to Algorithm 3.3
8          else
9              Compete_0()                                    // Transition to Algorithm 3.5
10     else
11         wait for κ_s time slots
```

Algorithm 3.2: Neighborhood-Learning() for node v

1 The neighborhood learning algorithm is a simple three-way-handshake protocol as introduced in [125]. A node v starts the neighborhood learning algorithm and thus sends a learning request with its own ID with probability p_s for κ_s time slots. For each reply it receives (at most Δ), v itself acknowledges the reception.

Let us briefly introduce the different messages transmitted by the algorithm. There are three types of messages: $M_A^i(v, c_v)$ means that node v with current counter value c_v competes for color i. $M_C^i(v)$ implies that node v is colored with color i (and that nodes competing for this color should resign from competing for it). Also, a variant of this message may be used to serve color requests: With the message $M_C^i(v, w, j)$, a leader v additionally assigns the color j (i.e., the block of

Algorithm 3.3: Request(w) for node v

1 Transmit $M_R^v(w)$ with probability p_s for κ_s slots
2 Wait for $\Delta\kappa_l$ rounds to receive color assignment from w
3 **if** received color j from w **then** goto Compete$_j$()

Algorithm 3.4: Announce$_i$() for node v

1 Transmit the $M_C^i(v)$ announcement with $p_l(p_s)$ for $\kappa_l(\kappa_s)$ slots for leader / non-leader nodes
2 Goto Algorithm 3.6: Colored$_i$()

Algorithm 3.5: Compete$_i$() for node v (based on the presentation in [30])

1 $P_v \leftarrow \emptyset$, $\kappa_i \leftarrow \begin{cases} \kappa_l & \text{if } i = 0 \\ \kappa_s & \text{otherwise} \end{cases}$, Next $\leftarrow \begin{cases} \text{Request} & \text{if } i = 0 \\ \text{Compete}_{i+1} & \text{otherwise} \end{cases}$
2 **for** κ_s *time slots* **do**
3 **for each** $w \in P_v$ **do** $d_v(w) \leftarrow d_v(w) + 1$
4 **if** $M_A^i(w, c_w)$ received **then** $P_v \leftarrow P_v \cup \{w\}$; $d_v(w) \leftarrow c_w$
5 **if** $M_C^i(w)$ received **then** goto Next; leader $\leftarrow w$
6 $c_v \leftarrow \Xi(P_v)$ with $\Xi(P_v) \leq 0$ such that $\Xi(P_v) \notin \{d_v(w) - \kappa_i, \ldots, d_v(w) + \kappa_i\}$ for each $w \in P_v$
7 **while** *true* **do**
8 $c_v \leftarrow c_v + 1$
9 **if** $c_v > \kappa_s$ **then**
10 Announce$_j$() for $j \leftarrow \begin{cases} \text{minimal} \notin \mathrm{C}_{\text{taken}} & \text{if } i = 0 \\ i & \text{otherwise} \end{cases}$
11 **for each** $w \in P_v$ **do** $d_v(w) \leftarrow d_v(w) + 1$
12 transmit $M_A^i(v, c_v)$ with probability p_s
13 **if** $M_C^i(w)$ received **then** goto Next; leader $\leftarrow w$
14 **if** $M_A^i(w, c_w)$ *received* **then**
15 $P_v \leftarrow P_v \cup \{w\}$; $d_v(w) \leftarrow c_w$
16 **if** $|c_v - c_w| \leq \zeta_i$ **then** $c_v \leftarrow \Xi(P_v)$

colors from j to $j + c_2\Gamma^\zeta$) to w. Finally the request message $M_R^v(w)$ is transmitted from v to a leader w to inform w that v selected w as its leader and requests to get a block of colors assigned.

As leaders must handle up to Δ color requests, we allow leaders to communicate with a higher transmission probability. Before determining the transmission probabilities, let us define two constants c_1 and c_2. We use them to describe the constant factor in the number of leader nodes in either $\bar{R}$ and $2\bar{R}$, and set them so that $\chi(\bar{R}, \underline{R}/2) = c_1\Gamma^\zeta$ and $\chi(2\bar{R}, \underline{R}/2) = c_2\Gamma^\zeta$. Let the transmission proba- c_1, c_2

Algorithm 3.6: Colored$_i$() for node v

```
1  count ← 0, current ← −2, serveCount ← 1, Q ← ∅   // current gets dummy request
2  if i = 0 then i ← [c1Γ^ζ]\C_taken                 // Select leader color from available pool
3  while coloring protocol runs do
4      if i > c1Γ^ζ then                             // non-leader nodes
5          Transmit M_C^i(v) with probability p_s
6      else                                          // leader nodes
7          count ← count+1
8          Transmit M_C^i(v) with probability p_s
9          if M_R^w(v) received then Q.add(w)
10         if count > κ_l or current = −1 then       // Serve the next request
11             count ← 0, serveCount ← serveCount +1
12             j ← serveCount ·c2Γ^ζ
13             current ← Q.first() (or -1 if Q empty)
14         if current ≠ −1 then                      // Serve the current request
15             Transmit M_C^i(v, current, j) for κ_l slots with prob. p_l, where j is a free
               color (block). Afterwards set current ← −1.
```

bility used by non-leader nodes be $p_s = \gamma/(2\Delta)$ and the transmission probability $p_l = \gamma/(2c_1\Gamma^\zeta)$ reserved for special leader tasks (for example the announcement of winning a leader competition, or answering color requests). As the number of leaders in each broadcasting range is limited, successful message transmission can still be achieved (cf. Corollary 3.15).

3.5.3 *MW-Coloring for Arbitrary Transmission Powers*

Before diving into the analysis, let us first consider why the MW-Coloring algorithm does not work in the case of arbitrary transmission powers without our modifications and extensions. Let us consider the example network in Figure 3.5. The

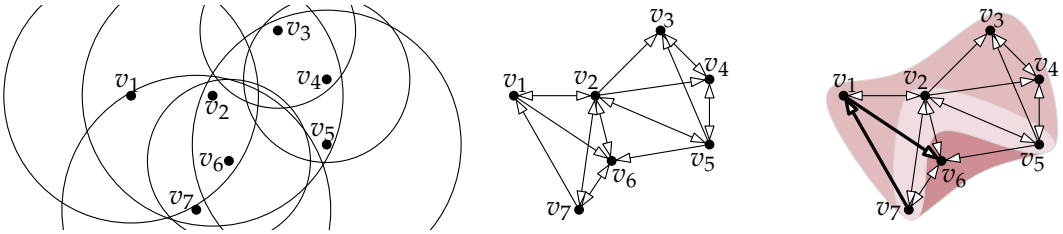


Figure 3.5: A network with heterogenous transmission powers. On the left the network is shown along with the broadcasting ranges, in the center the corresponding communication graph, and on the right the three levels of domination along with a longest directed path (in bold) are depicted.

main issue is that communication can be unidirectional and hence one of the two

neighbors dependent of the other. Let us consider some examples. During the execution of the original algorithm v_2 might become leader. However, then v_3 and v_4 are stuck as they cannot request a color from v_2. Otherwise, if v_3 would become leader, it cannot inform v_2 and v_5 about this, and thus they might become leader as well and take the same color.

Such cases are circumvented by our algorithm as the core-coloring algorithm (which essentially is the MW-Coloring algorithm with minor modifications) is only executed on nodes of the network that are not dominated by an uncolored node. Thus, at first the nodes v_2, v_5, v_7 execute the algorithm, then v_1, v_3, v_4, and finally v_6. This ensures that the coloring algorithm is always executed on a set of nodes such that each pair has either a bidirectional or no communication link at all.

3.5.4 *Analysis*

Let us now begin with the analysis of the algorithm, which is split in two parts. In the first part we show that the transmissions conducted in the algorithm are successful with high probability. In the second part we show that the algorithm computes a valid $\mathcal{O}(\Gamma^\zeta \Delta)$ coloring, and terminates after at most $\mathcal{O}((\Delta + \ell\Gamma^{2\zeta})\Delta\Gamma^\zeta \log n)$ time slots.

A significant difference in the analysis of our algorithm compared to the previous analysis is that leader nodes do not necessarily form an independent set, as unidirectional communication links between leaders are allowed in the algorithm. In order to argue about the number of leaders surrounding a node, we bound the number of leaders in a transmission range in the following. This result is used throughout our analysis. A simple modification of the statement bounds the number of neighboring nodes that compete for the same color (cf. Lemma 3.13).

Lemma 3.12. *Let v be an arbitrary leader node. Then there are at most $c_1\Gamma^\zeta$ other leader nodes in the transmission range of v.*

Proof. Let us consider an arbitrary leader node v. Although leaders do not form an independent set, it holds that there are no bidirectional communication links between leaders. This holds since otherwise one of the two leaders must have been aware of the other leader and would not have become leader but requested a color from the other leader. Thus, it holds that for balls of radius $\underline{R}/2$ around each leader node in v's neighborhood, these balls do not intersect. It follows that there can be at most $\chi(\bar{R}, \underline{R}/2) = c_1\Gamma^\zeta$ leader nodes in a maximal transmission range. □

The bound on the number of leaders is tight up to constant factors. This can be shown with a construction similar to the one in Figure 3.2. The following lemma is required for the algorithms proof of correctness. As it can be proven with the same argumentation as Lemma 3.12, we state it without separate proof.

Lemma 3.13. *Let v be an arbitrary node. Then at a given time slot at most $c_2\Gamma^\zeta$ leader nodes can be within a distance of $2\bar{R}$ of v.*

Let us show that the transmissions in the algorithm are successful.

3.5.5 *Transmissions are Successful*

In order to apply the bound on the interference shown in Section 3.3, we need to bound the sum of transmission probabilities from within each transmission region.

Lemma 3.14. *Let leader nodes transmit with probability p_l and non-leader nodes with probability p_s, then the sum of transmission probabilities from within each transmission region is upper bounded by γ.*

Proof. Let us consider an arbitrary node v and the sum of the transmission probabilities from within v's transmission region

$$\sum_{w \in B_v} p_w \leq c_1 \Gamma^\zeta p_l + \Delta p_s \leq \gamma$$

This holds as at most $c_1\Gamma^\zeta$ leader nodes from each transmission region may transmit with probability p_l due to Lemma 3.12, while at most Δ other nodes in v's neighborhood transmit with probability at most p_s. □

The next corollary follows from the lemma with the argumentation for Theorem 3.7. It shows that the limited number of leader nodes are able to communicate to their neighbors in $\kappa_l = 8c/p_l \log n = 16cc_1\Gamma^\zeta\gamma^{-1}\log n$ time slots, while non-leader nodes require $\kappa_s = 8c/p_s \log n = 16c\Delta\gamma^{-1}\log n$ time slots. Overall it implies that all transmissions in the algorithm are successful w.h.p.

Corollary 3.15. *A message that is transmitted with probability p_l (p_s) for $\kappa_l = \mathcal{O}(\Gamma^{2\zeta}\log n)$ ($\kappa_s = \mathcal{O}(\Delta\Gamma^\zeta \log n)$) time slots reaches its intended receivers w.h.p.*

This shows that communication is successful with high probability even in this more general case. The result enables us to transfer the MW-Coloring algorithm to the case of arbitrary transmission powers, and highlights the applicability of our method to bound the interference in networks of nodes with arbitrary transmission powers. We show that the modified MW-Coloring algorithm finishes in $\mathcal{O}((\Delta + \ell\Gamma^{2\zeta})\Delta\Gamma^\zeta \log n)$ time slots in the next section, while the bound of $\mathcal{O}(\Gamma^\zeta\Delta)$ on the number of colors and the correctness is established in Section 3.5.7.

3.5.6 *Runtime of the Algorithm*

In this section we consider the runtime of the distributed node coloring algorithm. As the algorithm is state-based, we prove a runtime bound for each state of the algorithm (see an overview in Table 3.1). The theorem follows from a consideration about the maximum number of times each state can be visited by a node.

Theorem 3.16. *After executing Algorithm 3.1 for $\mathcal{O}((\Delta + \ell\Gamma^{2\zeta})\Delta\Gamma^\zeta \log n)$ time slots all nodes are colored.*

Proof. We show in Lemma 3.17 that the initial neighborhood learning protocol requires $(2\Delta + 1)\kappa_s$ time slots. The Wait state takes the longest if it is dominated for a

maximum number of time slots. Let us therefore consider the length of the longest directed path. According to Definition 3.11, it is bounded by ℓ. As nodes that are not dominated execute the core coloring algorithm, they are colored and have communicated their final color to their neighbors after one execution of the core coloring algorithm according to Lemma 3.24. Thus, those nodes cannot dominate other nodes anymore (as only uncolored nodes dominate), and hence the length of the longest directed path must decrease by one.

As v might have to wait until all other nodes in the longest directed path selected a final color before executing the core-coloring algorithm, it may take at most ℓ times the runtime of the core coloring algorithm until v is not dominated anymore. Once v is not dominated it starts executing the core coloring algorithm. The runtime of the core coloring algorithm is $\mathcal{O}(\Gamma^{2\zeta}\kappa_s)$ time slots according to Lemma 3.18 (see Table 3.1 for an overview). Thus, the worst case runtime for a node to select a final color is at most $\mathcal{O}((\Delta + \ell\Gamma^{2\zeta})\kappa_s) = \mathcal{O}((\Delta + \ell\Gamma^{2\zeta})\Delta\Gamma^{\zeta}\log n)$ time slots. □

Table 3.1: Runtime of the algorithm. CC stands for parts of the core coloring algorithm

State	**Runtime**	**Proof**
Neighborhood learning	$(2\Delta+1)\kappa_s)$	Lemma 3.17
CC (overall)	$\mathcal{O}(\Gamma^{2\zeta}\kappa_s)$	Lemma 3.18
CC: Compete$_0$	$3\kappa_s + \Delta\kappa_l$	Lemma 3.19
CC: Compete$_i$	$(c_2\Gamma^{\zeta}+3)\kappa_s$	Lemma 3.20
CC: Max. Compete$_i$'s	$\mathcal{O}(\Gamma^{2\zeta})\kappa_s$	Lemma 3.21
CC: Request	$\kappa_s + \Delta\kappa_l$	Lemma 3.22
CC: Announce	κ_l or κ_s	-

In the following we prove the results stated in Table 3.1. In the next lemma the runtime of the initial neighborhood learning protocol is bounded.

Lemma 3.17. *Let a node v execute Algorithm 3.2. After the execution, both v and its neighbors w_i know about their communication link and whether the link is bidirectional. The algorithm finishes within $(2\Delta+1)\kappa_s$ time slots.*

Proof. As node v transmits a neighborhood learning request (in κ_s time slots) to all nodes w_i in its transmission region, the nodes w_i answer within $\Delta\kappa_s$ slots after receiving the request (each w_i may serve at most $\Delta - 1$ other requests in the meantime). If the transmission of w_i reaches v, v receives the message and completes the three-way-handshake by acknowledging the reception of the message to w_i, again within at most $\Delta\kappa_s$ slots. Summing over the time slots required results in $(2\Delta+1)\kappa_s$. □

After finishing the neighborhood learning, each node listens to other nodes transmitting their final color for κ_s time slots (we shall call these phases the initialization phase). After learning these final colors, each node is required to wait until it is no

longer dominated. Afterwards the core coloring algorithm is executed. As the core-coloring algorithm is based on the MW-algorithm adapted to the SINR model in [30], our proofs are in parts based on of their analysis despite the different setting. We clearly mark these parts or directly refer to the original analysis.

In contrast to their algorithm, where the set of leaders forms an independent set, the set of leaders in our algorithm is not independent (cf. Lemma 3.12). This is one of the major differences and requires various adaptations in the algorithm and the analysis. We shall first state the overall result in the next lemma and then analyze each state separately in the remainder of this section.

Lemma 3.18. *The worst case runtime of the core coloring algorithm is* $\mathcal{O}(\Gamma^{2\zeta}\kappa_s)$ *time slots*

Proof. Let v be an arbitrary node. There are several ways for v to reach the announce color state. We shall argue that the scenario "(a) and (b)", which is depicted in Figure 3.6, is indeed the worst case:

(a) there is no leader reachable

(b) v does not win the Compete$_0$ state

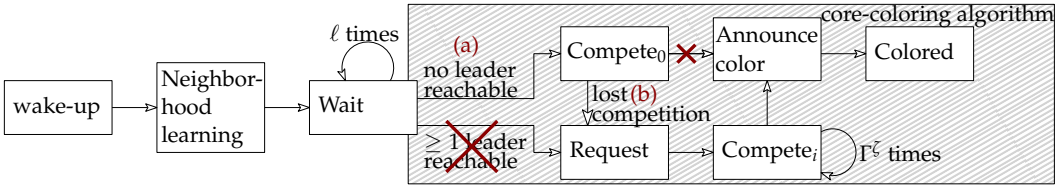


Figure 3.6: Worst case execution of the coloring algorithm.

For case (b) is is clear from the runtimes reported in Table 3.1 that the execution of the algorithm is faster if the Announce state is reached by winning the Compete$_0$ state. Case (a), however, does only lead to an increased runtime if case (b) occurs. As both (a) and (b) can happen in an execution, this leads to a worst case runtime, as depicted in Figure 3.6 (if a maximal number of time slots spend at the various Compete$_i$ states). The overall runtime follows by summation over the runtimes reported in Table 3.1. □

After proving the overall runtime based on the results in Table 3.1, we provide the remaining proofs for the results reported in the table in the following. The next three lemmas deal with the compete states, while the last lemma in this section considers the request state. The argumentation for the following lemmas is similar to the argumentation in [30] with minor changes to adapt to the number of leaders in a transmission range, the number of nodes competing for the same color and the time slots required for successful message transmission. Thus we shall only highlight the changes in the proofs and refer to [30] for complete proofs.

Lemma 3.19. *Let v be a node entering the Compete$_0$ state. At most $3\kappa_s + \Delta\kappa_l$ slots after entering Compete$_0$, v leaves the Compete$_0$ state.*

Proof (based on Lemma 6 in [30]). There are two cases. Either v wins the competition and becomes leader, or loses and enters the request state afterwards. In both cases the initial listen phase takes κ_s slots. However, as soon as v successfully transmits once (which is after at most another κ_s slots according to Corollary 3.15), it cannot be reset anymore according to [30, Proof of Lemma 6]. Hence either c_v reaches κ_s and v becomes leader, or a neighbor of v reaches κ_s first and forces v to transition to the request state. The counter c_v of v may at worst be reset to $\Delta\kappa_l$, thus the overall runtime of state Compete_0 is $3\kappa_s + \Delta\kappa_l$. Note that the referenced Lemma 6 holds since we restricted the nodes competing for a color (or the local leadership) to nodes with bidirectional communication links. □

Lemma 3.20. *Let v be a node entering the Compete_i state. At most $(c_2\Gamma^\zeta + 3)\kappa_s$ slots after entering Compete_i, v leaves the state.*

Proof. The lemma follows from an argumentation similar to that of Lemma 3.19. Note, however, that the counter of each node may at worst be reset to $c_2\Gamma^\zeta\kappa_s$. This is due to two facts: First, at most $c_2\Gamma^\zeta$ neighbors of v have the same block of colors assigned (see Lemma 3.13). Thus at most $c_2\Gamma^\zeta$ nodes compete with v for color i. Second, the nodes transmit probability p_s instead of p_l if they succeed in competing for color i, which leads to successful transmission in κ_s instead of κ_l time slots. □

If a non-leader node v got color i assigned by its leader and fails to verify it, v tries to verify $i+1$, then $i+2$ and so on. Thus non-leader nodes may be in more than one consecutive compete states. We will now bound the number of consecutive compete states a node may be forced to visit before being able to verify a color.

Lemma 3.21. *A node may be in at most $c_2\Gamma^\zeta$ consecutive compete states. It leaves the last compete state at most $(c_2\Gamma^\zeta) \cdot (c_2\Gamma^\zeta + 3)\kappa_s = \mathcal{O}(\Gamma^{2\zeta})\kappa_s$ slots after entering the first.*

Proof. Let us consider a node v that got color $j \neq 0$ assigned by its leader. The node tries to verify j and consecutive colors, until it wins a competition and enters the announce state. Only neighbors that have the same block of colors assigned can force v to move on to the next color by selecting the color v competes for as a final color themselves. By Lemma 3.13, the number of such neighbors is bounded from above by $c_2\Gamma^\zeta$. Hence after at most $c_2\Gamma^\zeta$ compete states all nodes that may compete with v for the same color selected their final color, and thus, v succeeds in the following competition. Note that this bound does also take nodes into account that selected a final color prior to v's execution of the core-coloring algorithm. The lemma follows since the compete states are consecutive as once a compete state is left for the announce state, the node does not return to a compete state[2]. □

After bounding the runtime of the compete states, let us consider the request state.

Lemma 3.22. *A node v that enters the request state leaves the request state at most $\kappa_s + \Delta\kappa_l$ time slots afterwards.*

2 This may not hold for the asynchronous case, as a nodes color may be reset by a dominating neighbor that takes the same color, see Section 3.5.8.

Proof. The node v first sends its request in κ_s slots, and subsequently will be served by its leader. As the leader can have at most Δ requests, v will be served at most $\Delta\kappa_l$ slots after the leader received the request. □

3.5.7 *Correctness of the Algorithm*

In order to prove the correctness of the algorithm it remains to show that the algorithm indeed computes a valid node coloring with at most $\mathcal{O}(\Gamma^\zeta\Delta)$ colors.

Theorem 3.23. *The coloring algorithm (Algorithm 3.1) computes a coloring with $\mathcal{O}(\Gamma^\zeta\Delta)$ colors such that each color forms an independent set.*

We show the theorem in two steps. We shall first argue that the core-coloring algorithm operates on a undirected communication graph, and thus the correctness of the coloring computed by the core-coloring algorithm follows from [30, Theorem 2] in Lemma 3.24. Afterwards we bound the number of colors used by the algorithm in the proof of the theorem.

Lemma 3.24. *Let v execute the core coloring algorithm. Then afterwards v has selected a valid color and communicated its color to its neighbors.*

Proof. Due to the state Wait in Algorithm 3.1, nodes that are dominated are not allowed to execute the core-coloring algorithm. Thus, each node that executes the core-coloring algorithm either has a valid color or has no uncolored neighbor that dominates it (or both). This holds since nodes that selected a final color do not dominate other nodes (cf. Section 3.5.1).

As dominated nodes do not start executing the core-coloring algorithm, it holds that the core coloring algorithm operates on a set of nodes that form an undirected communication graph. Thus the independence of each color follows from [30, Theorems 1 and 2]. □

We are now ready to prove Theorem 3.23.

Proof of Theorem 3.23. Let us consider two nodes u and v that are colored with the same color i. Let us first assume there is a bidirectional communication link between u and v. If u and v executed the core-coloring algorithm at the same time, the contradiction follows from Lemma 3.24. If on the other hand u and v did not compete in the core-coloring algorithm at the same time, let v be the node colored earlier. Due to the bidirectional communication, v was able to communicate to u that it is colored with i and thus prevented u from verifying i, which contradicts the assumption that both are colored with i. If there is a directed link between u and v, we assume w.l.o.g. that it is directed from u to v. As only unblocked nodes participate in the core coloring algorithm, u selected and communicated its color i before v, which prevents v from selecting i[3]. This contradicts the assumption, and hence the node coloring is valid.

3 This may happen due to asynchronous wakeup, see Section 3.5.8 for this case.

Let us now bound the number of colors. As $c_1\Gamma^\zeta$ is an upper bound on the number of leader nodes that can be in the transmission range of a leader node (cf. Lemma 3.12), at most $c_1\Gamma^\zeta - 1$ leader colors can be blocked when a leader node selects its color. Hence, $c_1\Gamma^\zeta$ leader colors are sufficient. The number of non-leader color blocks is bound by the number of requests a leader has to serve in the worst case, which is obviously Δ. Due to Lemma 3.21 it holds that for each request at most $c_2\Gamma^\zeta$ consecutive colors are required. According to Algorithm 3.6, $c_2\Gamma^\zeta$ is the first non-leader color that is assigned. Overall at most $c_2\Gamma^\zeta(\Delta+1)$ non-leader colors are used by the algorithm. □

3.5.8 *Asynchronous Node Wake-up*

The MW-Coloring algorithm, which is the core of our coloring algorithm, can deal with the asynchronous wake-up of nodes. Let us briefly discuss how to adapt the extensions presented in this section to handle the asynchronous wake-up of nodes. We use the same notion of asynchronous communication as Goussevskaia et al. [55]. Thus, we divide the time in time slots for our analysis, although time slots are not required in practice. Compared to the synchronous model, we double the time reserved for each time slot to account for potentially overlapping transmissions (this is sometimes known as the Aloha trick, cf. Section Section 2.2.4). The asymptotic complexity is not affected by this trick as the runtime increases only by a factor of two.

The neighborhood learning protocol enables the nodes to know about the directionality of the communication links they are involved in. In order to allow some nodes to start the algorithm after other nodes finished the neighborhood learning protocol, we reserve every second time slot throughout the execution of the algorithm to answer possible requests of the neighborhood learning protocol[4]. As with the Aloha trick this does not affect the asymptotic complexity. An illustration of how the time slots for the neighborhood learning protocol are interweaved with the regular coloring time slots is depicted in Figure 3.7. Transmissions are received regardless of the phase in which they arrived, unless the node transmits itself or the message cannot be decoded due to the SINR constraints. Essentially, we double the available time to ensure the capacity to handle the messages of both interweaving phases and transmit the responses within the time bounds.

We assume in our analysis that two nodes that simultaneously execute the core-coloring algorithm do not have a unidirectional link. However, such a unidirectional link might be introduced due to an awaking node. To prevent this, we require newly awaken nodes to wait for the time of one execution of the core-coloring algorithm before entering the main loop of Algorithm 3.1 to allow potentially dominated nodes to finish their current execution of the core-coloring algorithm. A node v with a unidirectional communication link to w may select the same color as w if v woke up after w already selected its color. To prevent a violation of the validity of the coloring, w resigns from color i if v announces to take color i. To

4 This corresponds to the time required for two transmissions to account for overlapping again.

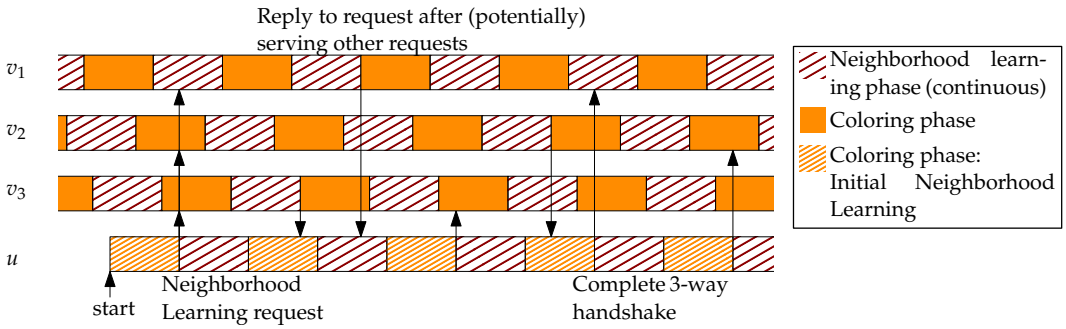


Figure 3.7: Illustration of the alternating time slots for the asynchronous implementation of the algorithm. Node u starts after its neighbors v_1 to v_3 and executes the neighborhood learning protocol (see Algorithm 3.2). Note that we simplified the illustration as we did not depict the probability based nature of message transmission over a period of several of these short and interweaving phases.

achieve the bound on the number of colors, leader nodes store the block of colors they assigned to nodes and reuse the colors accordingly. Note that if a node v is already colored it is not required to stop running the core coloring algorithm. However, if a node that has an unidirectional communication link to v selects the same color as v, v must resign from its color.

Thus, if a node v that is colored with color i receives an announcement from node w that w will take color i, v finishes serving its requests and then resigns from the color and enters the main loop of Algorithm 3.1. Serving the current requests takes at most $\Delta\kappa_l$ time slots, which is approximately the same as becoming leader. As at most another $c_1\Gamma^\zeta$ nodes can become leaders within each broadcasting range (cf. Lemma 3.12) before the old leaders resign after serving the requests, this can increase the transmission probabilities originating from leaders in each broadcasting range by a factor of at most two. This can be handled by decreasing the transmission probability for leader tasks appropriately.

Note that the runtime of the algorithm holds only for "stable" parts of the network in the case of asynchronous wake-up. As nodes that wake up may force other nodes to resign, we cannot guarantee a runtime based only on the wake-up time of the node itself. However, the runtime in Theorem 3.16 holds for v after the last node that can reach v or v's neighbors directly or through a directed path has woken up. This holds as v can only be forced to stop the algorithm or resign from its color by a node that reaches v directly or through a directed path, and v may expect a delay (for example in the request state) only if a neighbor of v is forced to resign. Hence, all time bounds are trivially valid once the last node in the network is awake.

3.5.9 *Maximal Independent Set*

An algorithm for solving MIS can be deduced by simplifying the coloring algorithm. As nodes can either be in the MIS or not, we do only require two colors.

Let 0 be the color that indicates that a node is in the MIS and 1 that it is not. As all nodes in the MIS are independent, we do not require the request state, and nodes in the MIS do not need to serve requests. Also, once a node v receives a $M_C^0(w)$ message, v can instantly transition to the Colored$_1$ algorithm. A factor of Γ^ζ is saved since the Request and Compete$_i$ states are not required anymore. This implies that after $\mathcal{O}((\Delta + \Gamma^\zeta \ell)\Delta\Gamma^\zeta \log n)$ time slots, each node selected a color and thus either is in the MIS or not.

3.6 CONCLUSION

In this chapter we have proven a bound on the interference in networks with arbitrary transmission powers assignments in wireless ad hoc networks. We believe that this generic result will be of use in many algorithms designed for such networks. We have shown that local broadcasting can be transferred to the general case of arbitrary transmission powers with minor efforts due to this result. Local broadcasting can be achieved in $\mathcal{O}(\Delta\Gamma^\zeta \log n)$ time slots if Δ is known, and $\mathcal{O}((\Delta + \log n)\Gamma^{2\zeta} \log n)$ time slots otherwise. We additionally considered the case that allows each node to change its transmission power variably in each time slot and studied the dependence on Γ theoretically and experimentally. To highlight the applicability of our results on communication in networks with arbitrary transmission powers, we presented a distributed node coloring algorithm that is fully adapted to the characteristics of directed communication networks such as unidirectional communication links. Our algorithm computes a $\mathcal{O}(\Gamma^\zeta \Delta)$-coloring in networks with heterogeneous transmission power in $\mathcal{O}((\Delta + \ell\Gamma^{2\zeta})\Delta\Gamma^\zeta \log n)$ time slots. We conclude with two open problems. Can the dependence on Γ be reduced as suggested by our experiments? Is there a fast distributed node coloring algorithm in this setting that does not require the nodes to learn neighborhood first?

4

DISTRIBUTED $(\Delta+1)$-COLORING IN THE SINR MODEL

In wireless ad hoc networks, distributed node coloring is a fundamental problem closely related to establishing efficient communication through TDMA schedules. For networks with maximum degree Δ, a $(\Delta+1)$-coloring is the ultimate goal in the distributed setting as this is always possible. In this chapter we propose a very simple (4Δ)-coloring along with a color reduction technique to achieve $\Delta+1$ colors. All algorithms have a runtime of $\mathcal{O}(\Delta \log n)$ time slots. This improves on previous algorithms for the SINR model either in terms of the number of required colors or the runtime, and matches the runtime of local broadcasting in the SINR model (which can be seen as an asymptotical lower bound).

This chapter is based on joint work with Roman Prutkin. Preliminary results are published in [40, 44] and online available as [43].

4.1 INTRODUCTION

One of the most fundamental problems in wireless ad hoc or sensor networks is efficient communication. Indeed, most distributed algorithms designed for the physical or SINR model consider algorithms to establish initial communication right after the network begins to operate. However, those initial methods of communication are not very efficient, as there are either frequent collisions and reception failures due to interference, or time is wasted in order to provably avoid such collisions and failures. If local broadcasting [55, 65, 133] is used, a multiplicative $\mathcal{O}(\Delta \log n)$ factor is required to execute message-passing algorithms in the SINR model (as before, we use a broadcasting range to define neighborhood, cf. Section 2.2.3). Thus, wireless networks often use a more refined transmission schedule as part of the Medium Access Control (MAC) layer. One of the most popular solutions to the medium access problem are TDMA (time division multiple access, cf. Section 2.1) schedules, which provide efficient communication by assigning nodes to time slots. The main problem in establishing a TDMA[1] schedule can be reduced to node coloring [113]. Given a node coloring, we can establish a transmission schedule by simply associating each color with one time slot. The node coloring considered in this chapter ensures that two nodes capable of communicating directly with each other do not select the same color. A TDMA schedule based on such a coloring

1 This also holds for related techniques such as Frequency-Division-Multiple-Access (FDMA) and Code-Division-Multiple-Access (CDMA) [113].

is usually not yet feasible in the SINR model, however, a feasible TDMA schedule can be computed based on our coloring, for example as shown in Chapter 6.

The problem of distributed node coloring dates back to the early days of distributed computing in the mid-1980s. In contrast to the centralized setting, a $(\Delta+1)$-coloring is considered to be the ultimate goal in distributed node coloring as it is already NP-complete to compute the chromatic number (i.e., the minimum number of colors required to color the graph) in the centralized setting [52]. There is a rich line of research in the distributed setting, however, most of the work has been done for message-passing models like the $\mathcal{LOCAL}$ model. Such models are designed for wired networks and do not fit the specifics of wireless networks.

In this chapter, we use two simple and well-known algorithms (covered for example in [13]) designed for message-passing models, and show that we can efficiently execute the algorithms in the SINR model. This cannot be achieved by a simple simulation of each round of the message passing algorithm by one execution of local broadcasting as this results in a runtime of $\mathcal{O}(\Delta \log^2 n)$ time slots. Instead, we modify both the communication in the SINR model and the algorithms to perfectly fit together. The synergy effect of our careful adjustments is that the coloring algorithm runs in $\mathcal{O}(\Delta \log n)$ time slots, which is asymptotically exactly the runtime of one local broadcast [55]. This matches the runtime of current $\mathcal{O}(\Delta)$-coloring algorithms [30], and improves on current $(\Delta+1)$-coloring algorithms for the SINR model which require $\mathcal{O}(\Delta \log n + \log^2 n)$ or $\mathcal{O}(\Delta \log^2 n)$ time slots (in a different setting) [136]. In the $\mathcal{LOCAL}$ model a lower bound of $\frac{1}{2}\log^* n$ exists [96]. Thus, in contrast to the algorithm proposed in this chapter, at least $\Omega(\log^* n)$ rounds of local broadcasting are required to execute distributed node coloring algorithms designed for the $\mathcal{LOCAL}$ model in the SINR model (unless more efficient communication becomes available).

The communication between nodes in our algorithm is assumed to be as established just after deployment of the sensor network and based on the local broadcasting algorithm proposed by Goussevskaia et al. [55]. Thus, we do not require any pre-computed structure, the only knowledge the nodes require is an upper bound on the maximum number of nodes in the proximity area (or alternatively the maximal degree Δ), an upper bound on the number of nodes in the network, as well as some model-related hardware constants in order to enable initial communication. All our results hold with high probability (w.h.p.). As union bounding a w.h.p. event only decreases the constant c in the probability $1 - 1/n^c$, which can be cancelled by increasing a constant factor in the runtime, we refrain from stating exact w.h.p. bounds in our analysis to simplify notation. Such requirements and assumptions are common in the SINR model. We refer to Section 2.2.3 for more details. We state related work and our contributions in the following.

Related Work

Research on distributed node coloring was started by Linial in [97]. Among other results he shows a lower bound of $\Omega(\log^* n)$ to compute a 3-coloring of an n-cycle. Since then a large body of works considered this distributed coloring problem in

the message-passing models, e.g. [9, 12, 14, 15, 25, 90, 92, 108]. Thus, we highlight only the most efficient algorithms for $(\Delta+1)$-colorings here and refer to a recent monograph by Barenboim and Elkin [13] for a more thorough overview. The fastest deterministic algorithm for general graphs is due to Awerbuch et al. [9] and Panconesi and Srinivasan [108] and runs in $2^{\mathcal{O}(\sqrt{\log n})}$. For moderate values of Δ, Barenboim [12] recently improved the runtime to $O(\Delta^{3/4}\log\Delta+\log^* n)$, from previously $\mathcal{O}(\Delta)+\frac{1}{2}\log^* n$ [14]. The algorithm on which our first algorithm is based was the most efficient $(\Delta+1)$-coloring algorithm in the randomized setting until recently [13, Chapter 10]. The algorithm can be seen as a simple variant of Luby's maximal independent set algorithm [98]. Variants of this algorithm were considered experimentally in [38, 110]. An algorithm of a similar flavor was introduced in by Kothapalli et al. [90]. Using an orientation of the edges they showed that their algorithm computes a $(\Delta+1)$-coloring in $\mathcal{O}(\sqrt{\log n})$ rounds, while transmitting only one bit per node in each round. In recent years the randomized coloring problem received considerable attention which culminated in the currently best randomized algorithm running in $\mathcal{O}(\log\Delta+2^{\mathcal{O}(\sqrt{\log\log n})})$ due to Barenboim et al. [15]. For growth bounded graphs, which generalize unit disk graphs, a deterministic distributed algorithm due to Schneider and Wattenhofer [119] computes a valid $(\Delta+1)$-coloring in $\mathcal{O}(\log^* n)$ rounds. Note that all of these results operate in message-passing models such as $\mathcal{CONGEST}$ and do not consider interference.

In wireless networks, the SINR model received increasing attention first in the electrical engineering community, and was picked up by the algorithms community due to a seminal work by Gupta and Kumar [60]. For a general review of related works in the SINR model we refer to Section 2.2.4 and focus on distributed coloring algorithms in the following. A coloring algorithm due to Moscibroda and Wattenhofer [104] has been adapted to the SINR model by Derbel and Talbi [30], and extended to support arbitrary transmission powers and directed communication in the previous chapter. This algorithm computes an $\mathcal{O}(\Delta)$-coloring in $\mathcal{O}(\Delta\log n)$ time slots in the uniform power setting we consider in this chapter. The algorithm first computes a set of leaders using a maximal independent set algorithm. Then leader nodes assign colors to non-leaders, which again compete for their final color with a restricted number of neighboring nodes that may have received the same assignment. Yu et al. [136] propose two $(\Delta+1)$-coloring algorithms that do not require the knowledge of the maximum node degree Δ. Their first algorithm runs in $\mathcal{O}(\Delta\log n+\log^2 n)$ time slots and assumes that nodes are able to increase their transmission power for the computation. This prevents conflicts between non-leader nodes by allowing the set of leaders to directly communicate to other leaders outside the transmission region and thus coordinating the assignment process. Their second algorithm does not require to tune the transmission power and runs in $\mathcal{O}(\Delta\log^2 n)$ time slots. In a less strict setting, Halldórsson, Wang and Yu [70] recently proposed a coloring algorithm in the SINR model that makes use of multiple channels to achieve a speedup for the aggregation and the coloring problem. Using k channels they compute a $\mathcal{O}(\Delta)$-coloring in $\mathcal{O}(\Delta/k+\log n\log\log n)$ time

slots. They use synchronous execution and require the nodes to measure the SINR or reception power.

Contribution

Our first contribution is RAND4DCOLORING, a very simple and well-known algorithm, which we adapt to the SINR model. In this algorithm each node selects a new color whenever there is a conflict with a neighboring node. We show that after $\mathcal{O}(\Delta \log n)$ time slots there are no conflicts with high probability. This matches the runtime of known $\mathcal{O}(\Delta)$-coloring algorithms and improves the hidden constant while also being significantly simpler.

As part of the analysis of RAND4DCOLORING we introduce a new method, which has the potential of improving the runtime of other randomized algorithms in the SINR model by a $\log n$ factor. This is achieved by carefully balancing the uncertainty of the communication in the SINR model with the uncertainty usually present in randomized algorithms.

Our third contribution is an asynchronous color reduction scheme. Given a d-coloring with $d > \Delta$, we compute a $(\Delta+1)$-coloring in $\mathcal{O}(d \log n)$ time slots. Combined with existing results, for example by Derbel and Talbi [30], our algorithm computes a $(\Delta+1)$-coloring in overall $\mathcal{O}(\Delta \log n)$ time slots. Thus, the algorithm achieves the declared goal of $\Delta+1$ colors, while the runtime of state-of-the-art $\mathcal{O}(\Delta)$-coloring algorithms is matched. Other $(\Delta+1)$-coloring algorithms in the SINR model require at least $\mathcal{O}(\Delta \log n + \log^2 n)$ time slots (under incomparable assumptions). Also, in the synchronous setting, the color reduction algorithm simplifies to an almost trivial color reduction scheme yielding the same results restricted to the synchronous setting.

Outline

In the next section we state the model along with required definitions and introduce different communication techniques in the SINR model. In Section 4.3 we describe and analyze the simple (4Δ)-coloring algorithm. The synchronous variant of our color reduction scheme is presented in Section 4.4, before we propose the asynchronous color reduction in Section 4.5. We conclude this chapter with closing remarks in Section 4.6.

4.2 MODEL AND PRELIMINARIES

Recall that we denote the set of *neighbors* of v by $N_v := \{w \in V \setminus \{v\} \mid \text{dist}(v,w) \leq r^B\}$ and $N_v^+ := N_v \cup \{v\}$ in Chapter 2. We generalize this notation to neighbors of neighbors, denote N_v^+ by N_v^1, and recursively define the *k-neighborhood* of v as $N_v^k := \cup_{w \in N_v^{k-1}} N_w^1$.

k-neighborhood

Note that since $r^B < r^T$, a node v may successfully receive transmissions from nodes that are not its neighbors in the communication graph, although successful transmission from those nodes cannot be guaranteed. This may lead to messages

received by more than Δ "neighbors". To ensure that $\Delta + 1$ colors are sufficient, we assume that messages from outside the broadcasting range are discarded by considering the signal strength of a received message (usually provided by wireless receivers as the Received-Signal-Strength-Indication (RSSI) value [11]). Thus, the maximum degree is defined as for the $(\Delta + 1)$-coloring in [136]. In a more practical setting, one could also define the communication graph based on the actual communication between two nodes.

In the following we introduce the notation required for the analysis of Algorithm 4.1, which consists of several phases and is described in the next section. We say that two neighbors v, w have a *conflict* if $c_v = c_w$ and denote the temporary color of v in phase t by c_v^t. The set of neighbors that are in a conflict with v in phase t is $X^t(v) := \{w \in N_v \mid c_v^t = c_w^t\}$. We call the set $X^t(v)$ the *conflict set* of v in phase t. Let us now define some events. The event that v is in a conflict in phase t is $\mathcal{E}^t_{\text{confl}}(v) := \exists w \in X^t(v)$. Note that it does not matter whether v knows of the conflict or not. The event that a transmission from v to all neighbors N_v of v in phase t is *successful* is $\mathcal{E}^t_{\text{succ}}(v)$. The same transmission is not successful or *fails* if at least one neighbor was unable to receive the message. The corresponding event is $\mathcal{E}^t_{\text{fail}}(v)$. We replace $\mathcal{E}$ by $\mathbb{P}$ to denote the probability of an event, e.g. $\mathbb{P}^t_{\text{succ}}(v)$ for $\mathcal{E}^t_{\text{succ}}(v)$. Note that although the events $\mathcal{E}^t_{\text{succ}}(v)$, and $\mathcal{E}^t_{\text{fail}}(v)$ may not be independent of events happening at other nodes, our bounds on the corresponding probabilities $\mathbb{P}^t_{\text{succ}}(v)$ and $\mathbb{P}^t_{\text{fail}}(v)$ are independent from the node v and possible events at other nodes. Also, our bounds $\mathbb{P}^t_{\text{succ}}(v)$ and $\mathbb{P}^t_{\text{fail}}(v)$ on these events include the event that v reaches some but not all of its neighbors, as $\mathbb{P}^t_{\text{fail}}(v) \leq 1 - \mathbb{P}^t_{\text{succ}}(v) \leq 1/12$ and $11/12 \leq \mathbb{P}^t_{\text{succ}}(v) \leq 1$ (see Lemma 4.1). If $p \geq c$ for probability p and a constant c, we say that p is at least constant, or simply constant. The nodes use two different transmission probabilities in order to adapt to the requirements of the corresponding algorithms. Probability $p_1 := \frac{1}{2\Delta^A}$ is used in Algorithm 4.1, while Algorithm 4.3 uses p_1 and $p_2 := \frac{1}{180}$. Let c be an arbitrary constant with $c > 1$. Throughout this chapter, we use the following definitions: $\kappa_\ell := c\lambda \ln n / p_\ell$ for $\ell = 1, 2$, $\kappa_0 := \lambda \ln 12 / p_1$, and λ a constant (for more details, we refer to Appendix A.2). It holds that $\kappa_0 \in \mathcal{O}(\Delta)$, $\kappa_1 \in \mathcal{O}(\Delta \log n)$, and $\kappa_2 \in \mathcal{O}(\log n)$.

conflict

conflict set $X^t(v)$

$\mathcal{E}^t_{confl}(v)$

$\mathcal{E}^t_{succ}(v)$

$\mathcal{E}^t_{fail}(v)$

$\mathbb{P}^t_{confl}(v)$

$\mathbb{P}^t_{succ}(v)$

$\mathbb{P}^t_{fail}(v)$

p_1, p_2

$\kappa_0, \kappa_1, \kappa_2$

4.2.1 *Extending Local Broadcasting:*

We show that local broadcasting with constant success probability in time inversely proportional to the transmission probability can be achieved. This extends known results regarding local broadcasting, which guarantee local broadcasting with high probability for a fixed number of time slots. Although we are the first to use local broadcasting with constant success probability, the proof of the following lemma is mainly based on standard techniques. Thus, we defer it to Appendix A.2.

Lemma 4.1. *Let v be a node transmitting with probability p_1. Then v successfully transmits to its neighbors with probability $\geq 11/12$ within κ_0 time slots. Transmissions with probability p_ℓ for κ_ℓ time slots are successful w.h.p. for $\ell \in \{1, 2\}$.*

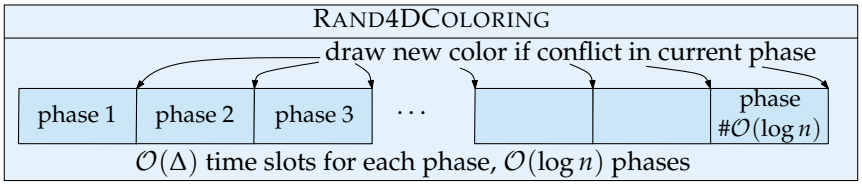


Figure 4.1: Illustration of Algorithm 4.1, which selects a random color at the end of a phase if a conflict is detected. The algorithm computes a (4Δ)-coloring in $\mathcal{O}(\Delta \log n)$ time slots

4.3 SIMPLE (4Δ)-COLORING

The algorithm we propose is at its heart a very simple and well-known randomized coloring algorithm. The underlying approach dates back to an algorithm to compute maximal independent sets by Luby [98], and is covered for example in [13, Chapter 10].

Algorithm 4.1: RAND4DCOLORING for node v

$F_v \leftarrow [4\Delta]$, $c_v^{-1} \leftarrow F_v$.rand()
for $t \leftarrow 0; t \leq 6(c+3)\ln n; t \leftarrow t+1$ **do** // each one phase
  if $c_v^{t-1} \notin F_v$ **then** $c_v^t \leftarrow F_v$.rand() // if conflict, new color
  else $c_v^t \leftarrow c_v^{t-1}$ // otherwise, keep it
  $F_v \leftarrow [4\Delta]$
  Transmit c_v^t with probability p_1 for κ_0 time slots
  foreach *received color* c_w^t *from neighbor* $w \in N_v$ **do** $F_v \leftarrow F_v \setminus \{c_w^t\}$

The algorithm RAND4DCOLORING (Algorithm 4.1 and Figure 4.1), is a simple, distributed and phase-based coloring algorithm. In each phase the node checks whether it knows of a conflict with one of its neighbors. If so, it randomly draws a new color from the set F_v of colors not taken in the previous phase. Finally, the phase is concluded by transmitting the current color. This computes a valid coloring with 4Δ colors in $\mathcal{O}(\log n)$ phases, while each phase takes $\mathcal{O}(\Delta)$ time slots. However, in contrast to previous algorithms of this kind, we do not assume that successful communication is guaranteed by lower layers. Instead we allow the uncertainty in the randomized algorithm to be combined with the uncertainty in the communication in the SINR model, which is jointly handled in the analysis. Thereby we reduce the number of time slots required for each phase from $\mathcal{O}(\Delta \log n)$ to $\mathcal{O}(\Delta)$, making this simple approach competitive in the SINR model. Thus, Algorithm 4.1 solves the node coloring problem using 4Δ colors in $\mathcal{O}(\Delta \log n)$ time slots, which matches the runtime of local broadcasting in the SINR model and improves the state-of-the-art $\mathcal{O}(\Delta)$-coloring in [30]. Let us now state the main results of this section.

Theorem 4.2. *Let all nodes start executing Algorithm 4.1 simultaneously. After $\mathcal{O}(\Delta \log n)$ time slots, all nodes have a valid color $c_v \leq 4\Delta$ w.h.p.*

For the asynchronous setting, the bound on the runtime holds for node v only after all nodes in v's $\mathcal{O}(\log n)$-neighborhood are awake

Theorem 4.3. *Let a node v execute Algorithm 4.1 in the asynchronous setting. Then v has a valid color $c_v \leq 4\Delta$ w.h.p., at most $\mathcal{O}(\Delta \log n)$ time slots after all nodes in its $\mathcal{O}(\log n)$-neighborhood started executing the algorithm.*

In the following section we prove the result for the synchronous setting. In Section 4.3.2 we extending it to the asynchronous setting.

4.3.1 *Analysis of* RAND4DCOLORING

Despite the fact that the underlying coloring algorithm is well-known, our analysis is new and quite involved. The main reason for this is the uncertainty in whether a message is successfully delivered in one phase of Algorithm 4.1. In contrast to guaranteed message delivery, based for example on local broadcasting, message delivery with constant probability can be achieved a logarithmic factor faster, see Lemma 4.1. However, this reduction in runtime comes at a cost: While in the guaranteed message delivery setting, a node v can finalize its color once a phase without a conflict at v happened, this is not possible in our setting. We cannot guarantee the validity of the colors even if a node did not receive a message implying a conflict in one phase, as message transmission is successful only with constant probability. Nevertheless, we show that after $\mathcal{O}(\log n)$ phases of transmitting the selected color and resolving eventual conflicts, the coloring is valid in the entire network w.h.p.

In order to prove correctness of Algorithm 4.1 (RAND4DCOLORING) we shall first bound the probability of a conflict propagating from one phase of the algorithm to the next. This is the foundation for the result that our algorithm computes a valid (4Δ)-coloring in $\mathcal{O}(\Delta \log n)$ time slots w.h.p. for both the synchronous and the asynchronous setting. Assuming that a node v has a conflict in phase t, there are only two cases that may lead to a conflict at v in phase $t+1$:

1. Node v had a conflict in phase t that did not get resolved (either due to being unaware of the conflict or since the new color implies a conflict as well).

2. A neighbor of v had a conflict in phase t and introduced the conflict by randomly selecting v's color.

We shall show that the probability for both cases is at most constant (see Lemma 4.4). Thus, after $\mathcal{O}(\log n)$ phases it holds with high probability that a valid color has been found. Note that we focus on the synchronous setting in this section; however, the main results can be extended to the asynchronous case, cf. Section 4.3.2.

Lemma 4.4. *Let v be an arbitrary node and $\mathbb{P}^t_{confl}(v)$ the probability of a conflict at v in phase t. Then the probability of a conflict at v in phase $t+1$ is at most*

$$\mathbb{P}^{t+1}_{confl}(v) \leq \frac{5}{6} \cdot \max_{w \in N_v} \mathbb{P}^t_{confl}(w).$$

Proof. We shall prove the lemma by considering the two cases that may lead to a conflict at node v in phase $t+1$. The **first** case is that v has a conflict with at least one of its neighbors. Depending on which transmissions are successful there are 3 subcases. Note that $\rightarrow$ denotes $\mathcal{E}^t_{\text{succ}}(v)$, while $\leftarrow$ denotes $\exists w \in X^t(v) : \mathcal{E}^t_{\text{succ}}(w)$—with negations accordingly[2].

(a) $\not\rightarrow, \not\leftarrow$: It is not guaranteed that any of the conflict partners know of the conflict, as the transmissions from v and the nodes in the conflict set $X^t(v) \neq \emptyset$ failed at least partially. There is at least one neighbor $u \in X^t(v)$ that failed to transmit its color successfully to v, which happens with probability $\mathbb{P}^t_{\text{fail}}(u)$. Combined with v's failure to transmit its color successfully, case 1(a) happens with probability at most $\mathbb{P}^t_{\text{confl}}(v)(\mathbb{P}^t_{\text{fail}}(v)\Pr(\not\leftarrow)) \leq \mathbb{P}^t_{\text{confl}}(v)\mathbb{P}^t_{\text{fail}}(v)\mathbb{P}^t_{\text{fail}}(u) \leq \mathbb{P}^t_{\text{confl}}(v)(1/12)^2$. If any conflict partner knows of the conflict, the conflict would be resolved with a certain probability (as in the following cases). However, as this is not guaranteed, we account for the worst case: the conflict is not resolved and propagates to the next phase. Note that since this case happens only with a small probability, it holds that the total probability of case (a) and conflict at v in phase $t+1$ is small.

(b) $\rightarrow, \not\leftarrow$: All nodes in $X^t(v)$ failed to transmit successfully, but v transmitted successfully to all neighbors. Thus, all nodes in $X^t(v)$ know of the conflict, while v might be unaware of it. This case happens with probability at most $\mathbb{P}^t_{\text{confl}}(v) \cdot (\mathbb{P}^t_{\text{succ}}(v) \cdot \Pr(\not\leftarrow))$. The probability that a node $w \in X^t(v)$ selects v's color in phase $t+1$ is at most $\sum_{w \in X^t(v)} 1/|F_w|$ (even if v knows of a conflict and itself selects a new color). This results in an overall probability of at most

$$\begin{aligned}
&\mathbb{P}^t_{\text{confl}}(v) \cdot (\mathbb{P}^t_{\text{succ}}(v) \cdot \Pr(\not\leftarrow)) \cdot \sum_{w \in X^t(v)} \frac{1}{|F_w|} \\
&\leq \mathbb{P}^t_{\text{confl}}(v) \left(\prod_{w \in X^t(v)} \mathbb{P}^t_{\text{fail}}(w) \right) \cdot \sum_{w \in X^t(v)} \frac{1}{|F_w|} \\
&\overset{x:=|X^t(v)|}{\leq} \mathbb{P}^t_{\text{confl}}(v) \left(\mathbb{P}^t_{\text{fail}}\right)^x \cdot \frac{x}{3\Delta} \leq \frac{1}{3\Delta}\mathbb{P}^t_{\text{confl}} \cdot x \left(\frac{1}{12}\right)^x \leq \frac{1}{24}\mathbb{P}^t_{\text{confl}}
\end{aligned}$$

where the first inequality holds since the event $\not\leftarrow$ is equivalent to $\forall w \in X^t(v) : \mathcal{E}^t_{\text{fail}}(w)$ and $\mathbb{P}^t_{\text{succ}}(v) \leq 1$. The second inequality holds since $|F_w| \geq 3\Delta$ and by setting $x = |X^t(v)|$. The last inequality holds since $x(1/12)^x \leq 1/12$ for all $x \in \{1, \ldots, \Delta\}$, and $\Delta \geq 1$.

(c) $\leftarrow$: It holds that v knows of the conflict. Whether v's neighbors know of it or not is not guaranteed. This case happens with probability at most $\mathbb{P}^t_{\text{confl}}(v) \cdot (\Pr(\leftarrow))$. The probability that at least one neighbor of v has or selects the same color as v is at most $\sum_{w \in N_v} \frac{1}{|F_v|} \leq |N_v|\frac{1}{3\Delta} \leq \frac{1}{3}$.

2 A partial success of transmission is often sufficient to trigger dealing with a conflict. We do not consider this in our notation, however, as we evaluate $\mathbb{P}^t_{\text{succ}}(v)$ to be at most 1 for all v and since $\Pr(\text{transmission from } v \text{ to } u \text{ fails}) \leq \mathbb{P}^t_{\text{fail}}(v) \leq 1/12$, our analysis covers this case.

Using $\Pr(\leftarrow) \leq 1$, this results in a probability for a conflict at v in phase $t+1$ of at most $\mathbb{P}^t_{\text{confl}}(v) \cdot (1/144 + 1/24 + 1/3 \cdot \Pr(\leftarrow)) < \left(\frac{1}{2}\right) \cdot \mathbb{P}^t_{\text{confl}}(v)$.

In the **second** case, there was no conflict at v in phase t, but a neighbor w of v selected v's color due to a conflict at w, which happens with probability at most

$$\begin{aligned}
&\sum_{w \in N_v} \underbrace{\Pr(c_v^{t+1} = c_w^{t+1})}_{v\text{'s neighbor } w \text{ selects } v\text{'s color}} \sum_{u \in N_w} \underbrace{\Pr(c_u^t = c_w^t)}_{\substack{u \in N(w) \text{ told } w \\ \text{about their conflict}}} \\
&\qquad \leq \sum_{w \in N_v} \Pr(c_v^{t+1} = c_w^{t+1}) \mathbb{P}^t_{\text{confl}}(w) \\
&\leq \sum_{w \in N_v} \frac{1}{|F_w|} \mathbb{P}^t_{\text{confl}}(w) \quad \leq \left(\frac{1}{3}\right) \max_{w \in N_v} \mathbb{P}^t_{\text{confl}}(w)
\end{aligned}$$

The last inequality holds since $\sum_{w \in N_v} \frac{1}{|F_w|} \leq \sum_{w \in N_v} \frac{1}{3\Delta} \leq \frac{1}{3}$. Combining all events that could lead to a conflict at v in phase $t+1$ it holds that the probability of the union of the events is at most

$$\mathbb{P}^{t+1}_{\text{confl}}(v) \leq \left(\frac{1}{2}\right) \mathbb{P}^t_{\text{confl}}(v) + \left(\frac{1}{3}\right) \max_{w \in N_v} \mathbb{P}^t_{\text{confl}}(w) \qquad \leq \frac{5}{6} \cdot \max_{w \in N_v^+} \mathbb{P}^t_{\text{confl}}(w),$$

which concludes the proof. □

Note that the second case could be avoided if message delivery in each phase would be guaranteed, as a node v that does not have a conflict in phase t, would simply finalize its current color and communicate this. Thus, v could not be forced into a conflict anymore. We shall now show that a set of nodes executing Algorithm 4.1 computes a valid coloring, and hence prove Theorem 4.2.

Proof of Theorem 4.2. Let us consider the probability of a conflict at an arbitrary node $v \in V$ in phase $t = 6(c+3)\ln n$. It holds that

$$\begin{aligned}
\mathbb{P}^t_{\text{confl}}(v) &\leq \left(\frac{5}{6}\right) \max_{w \in N_v} \mathbb{P}^{t-1}_{\text{confl}}(w) \leq \left(\frac{5}{6}\right) \max_{w \in V} \mathbb{P}^{t-1}_{\text{confl}}(w) \\
&\leq \left(\frac{5}{6}\right)^t \max_{w \in V} \mathbb{P}^0_{\text{confl}}(w) \leq \left(1 - \frac{1}{6}\right)^{6(c+3)\ln n} \leq \frac{1}{n^{c+3}},
\end{aligned}$$

where the first inequality is due to Lemma 4.4. The third inequality holds since all nodes are in the same phase due to the synchronous start of the algorithm. Note that the upper bound on the probability that a conflict propagates holds for all nodes. The fourth inequality holds as $\mathbb{P}^0_{\text{confl}}(v) \leq 1$ for all nodes v. The last inequality holds due to a well-known mathematical fact (cf. Fact 2.1 in Section 2.2.3). Thus, the probability for a conflict at an arbitrary node v is small. A union bound over all nodes in the network implies that the coloring is valid w.h.p. The runtime of Algorithm 4.1 is $\mathcal{O}(\Delta \log n)$, as it consists of $6(c+3)\ln n = \mathcal{O}(\log n)$ phases, and each phase takes $\kappa_0 = \mathcal{O}(\Delta)$ time slots according to Lemma 4.1. □

4.3.2 *Asynchronous Simple Coloring*

Let us now consider the asynchronous setting. Recall that we call all nodes that can reach v within $\mathcal{O}(\log n)$ rounds the $\mathcal{O}(\log n)$-neighborhood of v. We say that this neighborhood is *stable* if those nodes are all awake. If the $\mathcal{O}(\log n)$-neighborhood of a node v is stable, Lemma 4.4 holds as well, with only small changes to the proof (mostly caused by overlapping phases, we refer to [42] for the modified version of the proof). Thus, once all nodes in v's $\mathcal{O}(\log n)$-neighborhood are awake, we can bound the probability using said lemma, and prove Theorem 4.3, which we restate below, similar to the proof of Theorem 4.2.

stable

Theorem 4.3. *Let a node v execute Algorithm 4.1 in the asynchronous setting. Then v has a valid color $c_v \leq 4\Delta$ w.h.p., at most $\mathcal{O}(\Delta \log n)$ time slots after all nodes in its $\mathcal{O}(\log n)$-neighborhood started executing the algorithm.*

Proof. In each phase the algorithm transmits its current color and receives the colors of its neighbors with constant probability. Based on the probabilities for a successful transmission and the probability to select a color used by a neighbor after a conflict is detected, it holds that if the neighbors started before v (or at least before the analysis at v begins), we can prove that the probability decreases with each round. In order to decrease the probability with each round also for v's neighbors, and their neighbors, and so on, we require all nodes in a $\mathcal{O}(\log n)$-neighborhood of v to start before v, or postpone the analysis of v to a time t in which the nodes in the $(\log n)$-neighborhood started.

We consider $j := \log_k n$ phases (relative to v) such that all nodes in the $\log_k n$-neighborhood of v execute the algorithm and set $k := 6/5$. After the first phase we apply Lemma 4.4 to the nodes in N_v^{j-1}. This implies that these nodes have a conflict with probability at most $1/k$. Then we apply the lemma to N_v^{j-2}, and so on, decreasing the set of nodes intuitively by one neighborhood in each phase. After $j = \log_k n$ rounds Lemma 4.4 is only applied to v, with the result that the conflict probability of v is at most $\frac{1}{k^{\log_k n}} = 1/n$. □

If afterwards nodes outside of the $\mathcal{O}(\log n)$-neighborhood of v wake-up this does not influence v. As the conflict probability decreases with each phase, and at least $\Omega(\log n)$ phases are required until a new conflict could reach v, the probability of a conflict at v is at most $1/n$.

4.4 SYNCHRONOUS COLOR REDUCTION

In this section we show that SyncColorReduction (Algorithm 4.2 and Figure 4.2) transforms a valid d-coloring into a valid $(\Delta+1)$-coloring with high probability in time $\mathcal{O}(d \log n)$. Let us now describe the algorithm.

Assume the nodes have agreed on a valid coloring with d colors. The algorithm improves an existing coloring by reducing the number of colors to $\Delta+1$. The general scheme is that the nodes in the networks gradually replace their color by a color from $[\Delta]$. In each epoch, the nodes of exactly one color select a new color

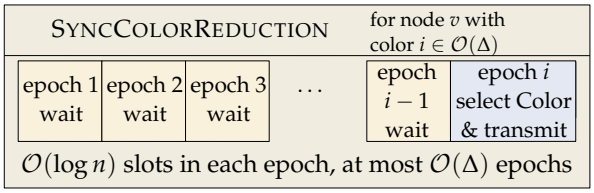


Figure 4.2: Illustration of Algorithm 4.2, the simple synchronous variant of our color reduction algorithm. Given a color $i \in \mathcal{O}(\Delta)$ the node selects and communicates its final color in epoch i. Due to the synchronous start and the valid initial coloring there are no conflicts.

Algorithm 4.2: SyncColorReduction for node v

```
1 F_v ← [Δ]
2 foreach color c_i = i ∈ [4Δ] do
3     if c_v^t ≠ i then                                   // wait
4         listen for κ_2 time slots
5         foreach received color c_w from neighbor w ∈ N_v do  F_v ← F_v \ {c_w}
6     else                                                // otherwise, select and transmit final color
7         c_v ← F_v.random()
8         Transmit c_v with probability p_2 for κ_2 time slots
```

from $[\Delta]$ and broadcast their selection to their neighbors. As each color forms an independent set, those nodes are free to select a color that is not yet taken in their neighborhood and communicate the selected color to the neighbors without introducing a conflict. In order to make the algorithm efficient in the SINR model, we utilize the given coloring to improve the communication efficiency by building a tentative schedule for the transmissions of each node. As nodes of exactly one color attempt to transmit in each epoch, the nodes of this color can successfully transmit their (new) color selection to neighboring nodes with high probability in only $\mathcal{O}(\log n)$ time slots instead of $\mathcal{O}(\Delta \log n)$. As each node that selects a final color from $[\Delta]$ knows of its neighbors colors, it can select and communicate its color without introducing a conflict. Overall this algorithm requires d epochs to reduce a d-coloring to a $(\Delta+1)$-coloring. As each epoch requires $\mathcal{O}(\log n)$ time slots this results in $\mathcal{O}(d \log n)$ time slots.

We shall only sketch the proof in the following, as the general color reduction scheme is already known for the $\mathcal{LOCAL}$ model (cf. for example [13]). In the proof we assume the coloring computed by Rand4DColoring to be given, other colorings can be used equivalently.

Lemma 4.5. *Given a network such that each node has a valid color $c_v \in [4\Delta]$ with probability at least $1 - \frac{1}{n^{c+3}}$. Then a synchronous execution of Algorithm 4.2 computes a valid $(\Delta+1)$-coloring at v with probability at least $\frac{1}{n^{c+1}}$.*

Proof. Given a node v with valid color with probability $1 - \frac{1}{n^{c+3}}$. As the color reduction scheme is a deterministic algorithm (apart from the communication),

only two possibilities for an invalid color at v after the execution exist. The first is that v's color was not valid, and v selected the same color as one of its neighbors. This may happen as two neighbors with the same color selected their final color in the same epoch of Algorithm 4.2. By applying a union bound, this happens with probability at most $\frac{\Delta}{n^{c+3}}$. The second possibility is that v did not receive the color of one of its neighbors, or one of v's neighbors did not receive v's color. It follows from Lemma 4.1 that communication in Algorithm 4.2 is successful with probability $1-\frac{1}{n^{c+3}}$. Thus, applying a union bound, it follows that the overall probability for an invalid color at v is at most $\frac{2\Delta+1}{n^{c+3}} \leq \frac{1}{n^{c+1}}$. □

We observe that a combination of Algorithms 4.1 and 4.2 results in a very simple synchronized $(\Delta+1)$-coloring algorithm with a total runtime of $\mathcal{O}(\Delta \log n)$ time slots.

Corollary 4.6. *Executing Algorithm 4.2 after Algorithm 4.1 in a synchronous setting yields a $(\Delta+1)$-coloring after $\mathcal{O}(\Delta \log n)$ time slots.*

4.5 ASYNCHRONOUS COLOR REDUCTION

In the following section we assume a valid node coloring with $d \in \Theta(\Delta)$ colors to be given and reduce the number of colors to $\Delta+1$ in $\mathcal{O}(d \log n)$ time slots. The algorithm we present in this section circumvents the synchronization problem, essentially, by using two levels of MIS executions. Our algorithm, which we denote by ColorReduction, is illustrated in Figure 4.3, the corresponding pseudocode can be found as Algorithms 4.3 to 4.6. We reference the MIS (Algorithm 4.5) executed with parameter $\ell = 1$ by *first level MIS*, and MIS($\ell = 2$) by *second level MIS*.

first level MIS

second level MIS

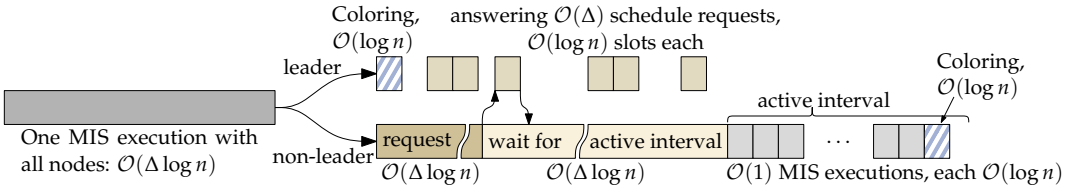


Figure 4.3: Runtime of ColorReduction. Overall $\mathcal{O}(\Delta \log n)$, given a $\mathcal{O}(\Delta)$ coloring.

Let us now describe the algorithm in more detail. The algorithm starts by executing the first level MIS algorithm that determines a set of independent nodes, which we call leaders. Each leader node transitions to Algorithm 4.4, selects and transmits the color 0 it selected and initializes its periodic leader schedule. This schedule assigns each color an *active interval* of length $\mathcal{O}(\log n)$ time slots to allow the nodes of this color to select their final color from $[\Delta]$.

active interval

Each node v_i that is not in the first level MIS selects a leader from its broadcasting range and requests the relative time until it is v_i's turn to be *active*. Upon receipt of its active interval, the node waits until the interval starts and then executes a second level MIS algorithm (which does not interfere with the first level MIS) a constant number of times. In this second level MIS the algorithm benefits from

fewer active nodes, and hence more efficient communication to allow each node to achieve successful transmission of a message to all neighbors in $\mathcal{O}(\log n)$ time slots. Moreover, we can speed up the MIS algorithm by the same factor of Δ to execute it in $\mathcal{O}(\log n)$ time slots, as only a constant number of nodes compete to be in each second level MIS. For each node that wins the second level MIS, there is no other node of the second level MIS in its broadcasting range. Thus, the winning node can select a valid color from F_v, where F_v is the set of colors not selected by neighbors, and transmit its choice to its neighbors without a conflict. If a node does not succeed to be in the second level MIS, it simply executes MIS(2) again. As each node succeeds in one of the MISs within its active interval (cf. Lemma 4.11), each node selects one of the $\Delta + 1$ colors.

Algorithm 4.3: ColorReduction for node v

```
1 F_v ← [Δ]\{0}
2 foreach received col_w do continuously
    F_v ← F_v\{col_w}
4 MIS(1)
```

Algorithm 4.4: Colored(ℓ) for node v

```
1  if ℓ = 1 then                                              // Level 1 leader
2      col_v ← 0, Q ← ∅, c'_v ← 0
3      announce M^1_C(v, col_v) with prob. p_2 for κ_2 slots
4      Define method τ(col, c_v) ≡ col · μ − c_v  mod Δμ neg., max., with
       |τ(col, c_v)| > κ_2
5      while protocol is executed do                          // serve requests
6          c'_v ← c'_v + 1
7          transmit M^1_C(v, col_v) with probability p_1
8          foreach received request from neighbor w: M_R(w, v, col_w^tmp) do
9              Q.push((w, col_w^tmp))
10         if Q not empty then
11             (w, col_w^tmp) ← Q.pop(), t ← τ(col_w^tmp, c'_v)
12             for O(log n) slots do
13                 transmit M^1_C(v, w, t) with probability p_2   // increment c'_v, t
   else                                                       // Level 2 / Non-leader node
15     col_v ← F_v.rand()                                     // valid
16     announce M^2_C(c, col_v) with prob. p_2 for κ_2 slots
17     while protocol is executed do                          // keep color valid
18         transmit col_v with prob. p_1
```

Algorithm 4.5: MIS(ℓ) for node v, based on MW-coloring [30, 120]

$P_v \leftarrow \emptyset$, NEXT $\leftarrow \begin{cases} \text{LEVEL2} & \text{if } \ell = 1 \\ \text{MIS(2)} & \text{otherwise} \end{cases}$
for κ_ℓ *time slots* **do** // Listen first
  foreach $w \in P_v$ **do** $d_v(w) \leftarrow d_v(w) + 1$
  if $M_A^\ell(w, c_w)$ received **then** $P_v \leftarrow P_v \cup \{w\}$; $d_v(w) \leftarrow c_w$
  if $M_C^\ell(w)$ received **then** NEXT(w)
$c_v \leftarrow \Xi(P_v)$ // maximal, non-positive, not conflicting with competing counters in P_v
while *true* **do** // then compete for MIS
  $c_v \leftarrow c_v + 1$
  if $c_v > \kappa_\ell$ **then** COLORED(ℓ) // success
  foreach $w \in P_v$ **do** $d_v(w) \leftarrow d_v(w) + 1$
  if $M_C^\ell(w)$ received **then** NEXT(w)
  transmit $M_A^\ell(v, c_v)$ with probability p_ℓ
  if $M_A^\ell(w, c_w)$ *received* **then** // received competing counter
    $P_v \leftarrow P_v \cup \{w\}$; $d_v(w) \leftarrow c_w$
    if $|c_v - c_w| \leq \kappa_\ell$ **then** $c_v \leftarrow \Xi(P_v)$

4.5.1 *Notation for* COLORREDUCTION *and* MIS

Let us now describe the notation used in the algorithm. We denote the set of available colors by F_v. Note that throughout the algorithm, each node deletes the final colors it receives from F_v. The MIS algorithm (Algorithm 4.5) aims at allowing exactly one node in each neighborhood to succeed to COLORED (Algorithm 4.4), select a color, and announce its success to its competitors. There are minor differences depending on the two levels $\ell = 1$ and $\ell = 2$, however, the algorithm remains the same. We describe the MIS algorithm below. In Algorithm 4.4, v is a leader, col_v denotes the final color from $[\Delta]$, and Q is a queue used to store nodes w

Algorithm 4.6: LEVEL2(w) for node v with leader w

while *true* **do**
  if $M_C^1(w, v, t)$ *received* **then**
    while $t < 0$ **do** // wait for interval
      $t \leftarrow t + 1$ // one time slot each
    while $t < \mu$ **do** // active interval, $\mu = 2k^2\kappa_2$
      // increase t by one in each time slot during MIS(2)
      MIS(2)
  else // transmit request
    transmit $M_R(v, w, \text{col}_v^{\text{tmp}})$ with probability p_1

along with their initial color $\mathrm{col}_w^{\mathrm{tmp}}$ that requested an active interval from v. The remaining time is based on v's periodic schedule, which is defined by its counter value c'_v and w's color. We set $k := 90$, which corresponds to the maximum number of active nodes in a broadcasting range, see Lemma 4.9. The function $\tau(\mathrm{col}_w, c_v)$ intuitively sets t to the start of the next interval corresponding to w's color in v's schedule, so that the starting time of w can be communicated by v w.h.p. before w's active interval starts. During the transmission interval, t is decreased appropriately. The length of each active interval is $\mu := 2k^2\kappa_2 \in \mathcal{O}(\log n)$, and thus length of the schedule is $\Delta\mu \in \mathcal{O}(\Delta \log n)$.

k

active interval μ

schedule length $\Delta\mu$

Let us now consider the messages used throughout the algorithm. To transmit the current counter value c_v to the neighbors, a message $M_A^\ell(v, c_v)$ is used. This message contains the level ℓ, the transmitting node v and its counter value c_v. The message M_C^ℓ indicates that a node succeeded in the MIS of level ℓ. This message has two uses in the algorithm. If it is used to transmit the success in the MIS, it contains the node and the color it selected. If it is used (by a first level leader v) to answer requests for the activity interval, it contains v, the requesting node w and the time remaining until w's active interval begins. The request message $M_R(v, w, \mathrm{col}_v^{\mathrm{tmp}})$ is transmitted by a node v in Algorithm 4.6, if v failed to win the first level MIS algorithm. It contains the node v, a leader w in v's neighborhood and v's color $\mathrm{col}_v^{\mathrm{tmp}}$ from the initial $\mathcal{O}(\Delta)$-coloring.

Adapting the MIS Algorithm

Algorithm 4.5 is a adaptation of the MIS algorithm used in the coloring algorithm in [30, 120]. Apart from constant changes, the lemma follows directly from Theorems 1 and 2 in [30] if $\ell = 1$, and from Lemma 4.1 along with setting Δ to a constant in the proofs of both referenced theorems if $\ell = 2$.

Lemma 4.7. *Algorithm 4.5 computes an MIS among participating nodes within* $\delta_\ell\kappa_2 \in \mathcal{O}(\delta_\ell \log n)$ *time slots, where* $\delta_\ell = \begin{cases} \Delta & \text{if } \ell = 1 \\ k & \text{if } \ell = 2 \end{cases}$ *w.h.p.*

We briefly describe the MIS (Algorithm 4.5) for general ℓ. The algorithm is based on the interplay of counters within a node's neighborhood. Each node v has a counter c_v. We denote the set of neighbors competing in the MIS by P_v. For each node w in this set, the counter value is stored (and increased by one in each time slot) as $d_v(w)$. The MIS algorithm begins with a *listen* phase of κ_ℓ time slots, which ensures that each node participating in the MIS received the counter values of other active neighbors. As nodes joining later execute the listen phase, they know the status of their neighborhood before competing to be in the MIS. Before competing, the counter c_v of v is set to $\Xi(P_v)$. This sets c_v to the maximum value such that $c_v \leq 0$ and $c_v \notin \{d_v(w) - \kappa_\ell, \ldots, d_v(w) + \kappa_\ell\}$ for each $w \in P_v$. Intuitively, c_v is set to a maximal non-positive value that is not within an interval of κ_ℓ of a competing nodes counter value. This ensures that if v reaches the counter threshold, there is sufficient time to inform the competitors $w \in P_v$ of v's success,

or the other way around if one of v's competitors succeeds. The main idea required to prove Lemma 4.7 is that once a node transmits successfully to all its neighbors its counter is not reset anymore. As the counters are reset to at most $(\delta_\ell - 1) \cdot \kappa_\ell$, the runtime bound follows. Note that each node keeps the counter values $d_v(w)$ of neighbors w up-to-date by automatically incrementing them in each time slot.

4.5.2 *Analysis*

Let us first state the main result of this section.

Theorem 4.8. *Given a valid node coloring with $d \geq \Delta$ colors, Algorithm 4.3 computes a valid $(\Delta+1)$-coloring in $\mathcal{O}(d \log n)$.*

As the algorithm is essentially a simple color reduction scheme, each node selects a valid color if the communication can be realized as claimed. To prove this we show that in the second level indeed only a constant number of nodes are active in each broadcasting range (cf. Figure 4.4). We use this to achieve message transmission from active nodes to all their neighbors in $\mathcal{O}(\log n)$ time slots, and show that the second level MIS can be executed in $\mathcal{O}(\log n)$ time slots. Finally, we prove that each non-leader node v succeeds in a second level MIS, and thus colors itself with a color from $[\Delta]$, within the active interval v received by its leader.

We use the next lemma to prove our bounds on the communication in Lemma 4.1. It allows us to increase the transmission probability in the second level MIS by a factor of Δ compared to classical local broadcasting, leading to a decrease in the time required for successful message transmission by the same factor of Δ to $\mathcal{O}(\log n)$.

Lemma 4.9. *In the second level, at most $k = 90$ nodes are active in each broadcasting region.*

Proof. The lemma follows from a geometric argument regarding the number of first level MIS nodes within a certain distance of each node. Let us consider an arbitrary node v. Nodes in the broadcasting range of v must select their leader from the set of (first level) MIS nodes within the radius of two broadcasting ranges from v. Geometrically at most 18 independent nodes can be in a disk of radius $2r^B$, thus each node from the broadcasting range of v selects one of these 18 nodes as leader. Each leader may be selected as leader by at most 5 nodes of each color, since these nodes must be pairwise independent. Thus, overall the nodes in v's broadcasting range may follow up to 18 schedules, and for each schedule at most 5 nodes are active at the same time, which implies the upper bound of 90 active nodes in the broadcasting range of v. The proof is illustrated in Figure 4.4. □

Based on this result we can bound the runtime of our algorithm, starting with Algorithm 4.6.

Lemma 4.10. *Let v execute Algorithm 4.6 with leader w. Then a) v transmits the request message successfully within κ_1 time slots w.h.p.; b) v receives its active interval after at*

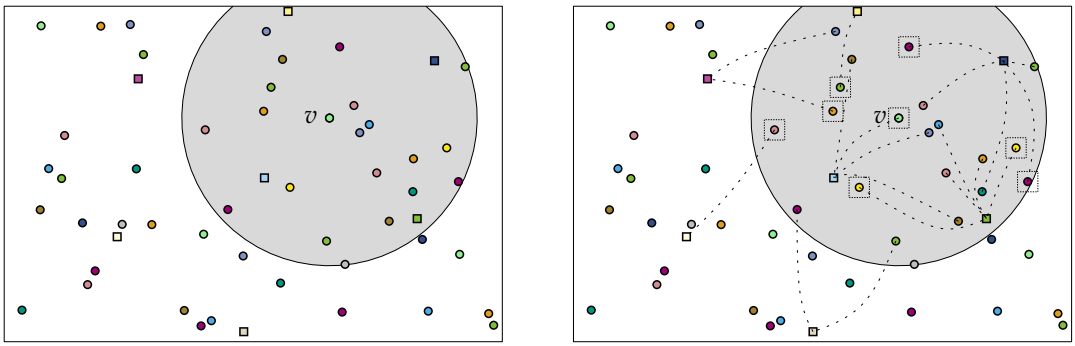


Figure 4.4: Left: Node v with its broadcasting region in a network with valid coloring; Nodes in the first level MIS are squares. Right: Nodes in v's broadcasting range are connected to their selected leader by a dashed line. Nodes currently active in the second level are surrounded by a square. At most $k = 90$ nodes are active in each broadcasting region.

most another κ_1 time slots w.h.p.; and c) the wait-time t until v's active interval starts is at most $\Delta\mu \in \mathcal{O}(\Delta \log n)$.

Proof. Part a) is implied directly by Lemma 4.1. Regarding Part b), it holds that v's leader node w must answer at most Δ requests. For each request w transmits the node's answer for κ_2 time slots. Thus v receives its active interval at most $\kappa_1 + \Delta\kappa_2$ time slots after it started transmitting its request. Part c) holds by definition of $\tau(\cdot, \cdot)$, as the length of the schedule is $\Delta\mu$. □

We shall now argue that each non-leader node succeeds to win a second level MIS in its active interval.

Lemma 4.11. *Given a node v executing Algorithm 4.6. Once $t = 0$, v wins a second level MIS set within $\mu = 2k^2\kappa_2 \in \mathcal{O}(\log n)$ time slots.*

Proof. The active interval of v is from $t_s = 0$ to $t_e = 2k^2\kappa_2$. According to Lemma 4.9, at most k nodes are active in v's broadcasting range at any time (including v). Due to the possible overlap of the active intervals, at most $2k - 2$ neighbors of v may be active during the interval $\{t_s, \ldots, t_e\}$. With each second level MIS execution of length at most $k\kappa_\ell$ time slots at least one neighbor is admitted to be in an independent set. This node selects a valid color, transmits this color to its neighbors and does not participate in following MIS executions. Thus, after at most $2k - 2$ MIS executions v either succeeded in a previous MIS execution, or all active neighbors of v succeeded and thus v must succeed in the next second level MIS execution. Finally, as $(2k - 1) \cdot k\kappa_\ell \leq 2k^2\kappa_\ell$ the lemma holds. □

As a final step we show that the final color selected by each node is valid w.h.p.

Lemma 4.12. *Given a node v entering Algorithm 4.4. It holds that a) while v transmits its final color no neighbor of v succeeds in a second level MIS w.h.p.; and b) the color v selects is not selected by one of v's neighbors w.h.p.*

Proof. Part a) holds since all neighbors of v that participated in the MIS after v succeeded in a second level MIS know the counter value of v w.h.p. (due to the offset between the counters ensured by $\Xi()$ and Line 14 in Algorithm 4.5). Hence they do not enter the MIS before v finished transmitting its final color. As the nodes are not synchronized, a node w may have just entered the active interval. In this case, however, the listening phase of κ_2 time slots prevents w from succeeding before it knows the status of all active neighbors. Thus, while v transmits its color, no neighbors of v select their final color (except for first level leaders taking color 0, which does not conflict with the second level coloring). For Part b) observe that the algorithm removes each final color it receives from the set of available colors according to Algorithm 4.3. Due to Part a) it holds that the colors of all neighbors of v that selected a final color before v were able to transmit the color to v successfully. Thus v selected a color which was not selected by one of its neighbors. □

We are now able to prove the main theorem. Note that runtime bounds hold for each node once the node starts executing the algorithm. This allows fully asynchronous wake-up of nodes.

Proof of Theorem 4.8. It follows from Lemma 4.12 and the fact that each node succeeds in an MIS (and hence enters Algorithm 4.4 and selects a final color), that the final color of each node is valid w.h.p. and a union bound over all nodes implies that overall the coloring is valid w.h.p. Only $\Delta+1$ final colors are used due to the initialization of the set F_v.

The runtime of the first level MIS is $\mathcal{O}(\Delta \log n)$ time slots according to Lemma 4.7. Algorithm 4.6 requires another $\mathcal{O}(\Delta \log n)$ slots until starting the active interval, which is of length $\mathcal{O}(\log n)$, overall resulting in $\mathcal{O}(\Delta \log n)$ time slots. □

This directly implies the following corollary. As the asynchronous runtime bound of RAND4DCOLORING holds only after the last node (in the nodes neighborhood woke up), we use the MWColoring algorithm that computes a valid $\mathcal{O}(\Delta)$-coloring in $\mathcal{O}(\Delta \log n)$ time slots for the corollary.

Corollary 4.13. *Let each node in the asynchronous network execute the MW-coloring algorithm [30], followed by Algorithm 4.3. Then $\mathcal{O}(\Delta \log n)$ time slots after a node started executing the algorithms it selected a valid color from $[\Delta]$.*

4.5.3 *Discussion*

Before concluding the chapter, we briefly discuss some of the assumptions or requirements of the algorithms.

Local Synchronization

We do not require synchronization, however, if synchronization is available we can use very simple algorithms to compute a $(\Delta+1)$-coloring (as described in Sections 4.3 and 4.4. A wireless network can be synchronized either by using

network-wide broadcasts, or by applying synchronization methods such as, for example, Timing-sync Protocol for Sensor Networks (TPSN) [50]. Assuming a node with a valid clock within few hops of each node, the network can be synchronized within a few rounds of local broadcasting. Then, the simple synchronous coloring algorithms presented in Sections 4.3 and 4.4 can be used to compute a $(\Delta + 1)$-coloring. For a survey on on synchronization methods for sensor networks we refer to [122].

Concurrent Execution of Algorithms

In this chapter, we consider the algorithms isolated from other algorithms that may be executed simultaneously by other nodes (which just woke up, for example). The additional interference introduced by a constant number c_0 of different algorithms that are executed simultaneously in the network can be handled simply by reducing the transmission probabilities used in the algorithms by c_0. This does not change the asymptotic complexity of the algorithms, as long as the number c_0 of concurrently executed algorithms is constant.

Without knowledge of Δ

We assume an upper bound on the maximum degree in the network to be known by the nodes. Regarding initial communication, there are algorithms that do not require this assumption. However, if this upper bound on the local density is not known, one must begin with a very low transmission probability and use a so-called slow-start technique to find the correct transmission probability. Current slow-start techniques for local broadcasting require $\mathcal{O}(\Delta \log n + \log^2 n)$ time slots [65, 133]. Unfortunately, we cannot adapt the slow-start technique to achieve successful message transmission with constant probability as shown for the case with knowledge of Δ in Section 4.2.1. For the algorithm by Halldórsson and Mitra [65], for example, reducing the runtime by a $\log n$ factor by allowing success with only constant probability (instead of w.h.p.) fails as we cannot guarantee that the so-called fallback occurs with high probability, which is required to keep the sum of local transmission probabilities low.

4.6 CONCLUSION

In this chapter we considered algorithms for distributed node coloring in the SINR model. We proposed two algorithms, which are both inspired by very simple algorithms well-known for decades in the message-passing models $\mathcal{LOCAL}$ and $\mathcal{CONGEST}$. Our first algorithm, RAND4DCOLORING, computes a valid (4Δ)-coloring in $\mathcal{O}(\Delta \log n)$ time slots. The algorithm is very simple, and achieves its runtime bound (in the case of asynchronous wake-up) once its $\mathcal{O}(\log n)$-neighborhood is awake.

Our second algorithm, COLORREDUCTION, computes a valid $(\Delta + 1)$-coloring in $\mathcal{O}(\Delta \log n)$ time slots and assumes a $\mathcal{O}(\Delta)$-coloring to be given. The algorithm

uses the initial coloring to coordinate the color selection, and is fully resistant to asynchronous wake-up of nodes. Combined with RAND4DCOLORING or the MW-Coloring algorithm this results in a $(\Delta + 1)$-coloring algorithm in $\mathcal{O}(\Delta \log n)$ time slots. As the best local broadcasting algorithms in this setting require $\mathcal{O}(\Delta \log n)$ time slots, this is best-possible unless faster local broadcast algorithms are found.

We conclude with open problems. It would be interesting to find an algorithm that achieves message transmission with constant probability without the knowledge of Δ that improves the runtime bound of $\mathcal{O}(\Delta \log n + \log^2 n)$. Such an algorithm could generalize the algorithms presented in this chapter to operate without prior knowledge of Δ. And if not, can we find fast distributed node coloring algorithms without the knowledge of Δ (without increasing the transmission power)?

5

EXPERIMENTAL EVALUATION OF DISTRIBUTED NODE COLORING ALGORITHMS

In this chapter we evaluate the distributed node coloring algorithms proposed in the previous chapter using the network simulator Sinalgo [33]. We compare the described coloring algorithms to the MW-Coloring algorithm introduced by Moscibroda and Wattenhofer [104] and transferred to the SINR model by Derbel and Talbi [30], and a variant of the coloring algorithm of Yu et al. [136]. We additionally consider several practical improvements to the algorithms and evaluate their performance in both static and dynamic scenarios.

Our experiments show that Rand4DColoring is very fast, computing a valid (4Δ)-coloring in less than one third of the time slots required for local broadcasting. Regarding other $\mathcal{O}(\Delta)$-coloring algorithms our algorithm is at least 4 to 5 times faster. Additionally, the algorithm is robust even in networks with mobile nodes and an additional listening phase at the start of the algorithm makes Rand4DColoring robust against the late wake-up of large parts of the network.

Regarding our $(\Delta + 1)$-coloring algorithm ColorReduction we observe that it is significantly slower than Rand4DColoring, however, still faster than the considered variant of the Yu et al. coloring algorithm, which is the only other $(\Delta + 1)$-coloring algorithm for the SINR model. Further improvement can be made with an error-correcting variant that increases the runtime by allowing some uncertainty in the communication and afterwards correcting the introduced conflicts.

The results of this chapter are published as [41].

5.1 INTRODUCTION

Compared to other models of interference often used in distributed computing, the SINR model of interference is a very realistic model of wireless communication. However, to analytically prove guarantees on the runtime and show an algorithms correctness becomes relatively complex. Thus, over the past years techniques were developed to tackle the complexity of the model. This, however, led to the introduction of many constant factors in different parts of the algorithms. In this chapter we study the two distributed node coloring algorithms we presented in Chapter 4 and two existing coloring algorithms in a more practical setting. We use the network simulator Sinalgo [33] to execute the algorithms in a variety of deployment scenarios in the static and the dynamic setting.

Let us briefly consider the algorithms we evaluate in this chapter. The first algorithm, which we denote by MWColoring in this chapter, was proposed by Moscibroda and Wattenhofer in [104] for the protocol model and transferred to the SINR model by Derbel and Talbi [30]. MWColoring computes an $\mathcal{O}(\Delta)$-coloring in $\mathcal{O}(\Delta \log n)$ time slots by first selecting leader nodes, which coordinate the color selection of other nodes so that only few nodes in each neighborhood compete for the same color. We implement and evaluate a variant of the distributed $(\Delta + 1)$-coloring algorithm proposed by Yu et al. [136]. Their algorithm computes leaders, which then increase their transmission power to block nodes that could hinder the leaders (original) neighbors from selecting a valid color. We use a variant of their original algorithm, as their algorithm operates in a slightly different setting, however, the ideas are applicable in our setting as well. We denote the variant by YuColoring and introduce it in Section 5.2.4. Finally, we consider two algorithms proposed in Chapter 4. Rand4DColoring is a very simple randomized coloring algorithm that computes a (4Δ)-coloring by simply selecting a new random color whenever a conflict is detected. Our second algorithm, ColorReduction, uses an existing coloring to coordinate the color selection process to compute a $(\Delta + 1)$-coloring. The nodes executing the algorithm first select a set of leaders, which compute a medium access schedule based on the existing colors. Due to this schedule only few nodes are active at a time, which enables the active nodes to quickly win the competition for their final colors. All considered algorithms require at most $\mathcal{O}(\Delta \log n)$ time slots. For more details, we refer to Section 5.2.

Related Work

Although distributed node colorings are widely applicable and their study was initiated more than 25 years ago, only few experimental evaluations are concerned with distributed node coloring algorithms. To the best of our knowledge the first experimental evaluation on distributed node coloring algorithms is due to Finocchi, Panconesi, and Silvestri [38]. They study very simple node coloring algorithms, which are similar to our Rand4DColoring coloring algorithm. However, their algorithms and the evaluation are based on the simpler message-passing models, which do not consider interference. They observed that such simple algorithms are very fast, and proposed some practical improvements to the coloring algorithms. In particular, they found that allowing each node to participate in the conflict resolution in each round yields a faster runtime than participating with some probability. In an earlier experimental study Marathe, Panconesi and Risinger [100] considered simple edge colorings of the same randomized trial-and-error flavor as the later considered vertex coloring algorithms. They found that such algorithms performed "extremely good" in their experiments. Pindiproli and Kothapalli [110] extend the study on distributed node coloring algorithms by Finocchi, Panconesi and Silvestri by considering the same randomized algorithms and compare it to a similar algorithm that requires only $\mathcal{O}(\sqrt{\log n})$ rounds of transmitting a single bit by exploiting a given orientation of the edges. They study oriented cycles, com-

plete and random graphs and find that exploiting a given orientation reduces the runtime in most cases.

A study on distributed node coloring algorithms was compiled under my guidance by Schlegel [118]. It considers RAND4DCOLORING, MWCOLORING, and YUCOLORING in the synchronous setting and can be seen as tentative to this chapter.

Contribution

First, we confirm that the very simple RAND4DCOLORING coloring algorithm is indeed very fast, achieving a runtime an order of magnitude faster than its direct competitor, the MW-coloring algorithm. Interestingly, the algorithm computes a valid (4Δ)-coloring in less time slots than required for one round of local broadcasting.

Second we show that our COLORREDUCTION algorithm is significantly faster than YUCOLORING, our variant of the $(\Delta + 1)$-coloring algorithm by Yu et al.

Third, we propose heuristic improvements for COLORREDUCTION, MWCOLORING, and YUCOLORING. These heuristic improvements are inspired by RAND4DCOLORING and allow the nodes to decrease the number of time slots accounted for the transmission of a message. This may lead to conflicts, which the algorithms try to resolve afterwards. We refer to these heuristics as correcting variants and show that they considerably improve the runtime while keeping the number of conflicts in the network close to zero.

Fourth, we study the performance of the algorithms in a network with mobile nodes and in a network in which a large fraction of the nodes start the algorithm after the remaining network has computed a valid coloring. We observe that our correcting variants of MWCOLORING and YUCOLORING, and especially RAND4DCOLORING are robust against mobility of nodes. Regarding the late wake-up of some nodes, we combine a listening phase added to the start of the algorithm with a strategy to select a new color that respects colors currently taken by neighbors. Our experiments show that this is sufficient to make RAND4DCOLORING robust against asynchronous wake-up of nodes.

Finally, a minor theoretical contribution of this chapter is the variant of YUCOLORING we transfer to our setting of known maximum node degree Δ. The distributed $(\Delta + 1)$-coloring algorithm achieves a runtime of $\mathcal{O}(\Delta \log n)$ time slots (in this simpler setting).

Outline

The remainder of this chapter is structured as follows. In the next section we describe and introduce the algorithms we consider in our experiments. In Section 5.3 we describe the simulator and the setting of our experiments before evaluating the algorithms. We conclude this chapter in Section 5.4.

5.2 CONSIDERED ALGORITHMS

We consider the algorithms proposed and described in Chapter 4, along with the competing algorithms MWColoring and YuColoring, and some heuristic improvements to the various algorithms. Recall that the color of a node is *valid* if no neighbor selected the same color. Also, we sometimes call a node that selected the same color as one of its neighbors to have a *conflict* with this neighbor, cf. Section 2.1 and Section 4.2. To execute the algorithms we experimentally determine the parameters for the algorithms. Most parameters are required in the process of enabling successful communication. As those parameters are the same as for local broadcasting, which we considered already in Section 2.3.1, we reuse the results obtained there (cf. Section 5.3 and Table 5.2). To compute the transmission probability used by the nodes we divide a parameter txConst by the maximum degree Δ. Other parameters are the duration, which is set to the number of time slots required for reliable communication in the network, and factor, which describes the multiplicative factor between regular and fast local broadcasting. We show the relation of the parameters in calculating the values used for the simulation in Table 5.1 Note that we use factor slightly different for Rand4DColoring,

Table 5.1: Transmission probabilities and durations based on the simulation parameters

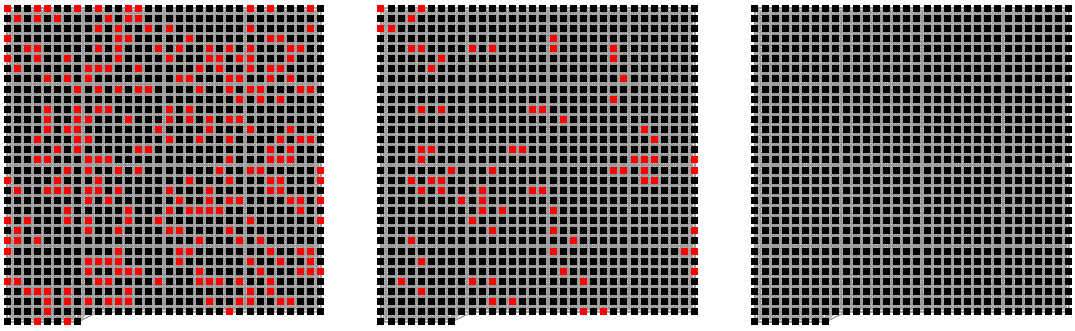
Simulation parameter	Value
Local broadcast	
transmission probability	$\frac{\text{txConst}}{\Delta}$
transmission duration	duration
Fast local broadcast	
transmission probability	$\frac{\text{txConst}}{\Delta} \times \Delta \times \text{factor} = \text{txConst} \times \text{factor}$
transmission duration	$\frac{\text{duration}}{\Delta \times \text{factor}}$

as this algorithm does not rely on fast local broadcast transmissions. Let us now describe the algorithms.

5.2.1 Rand4DColoring

We introduced and analyzed the phase-based algorithm Rand4DColoring in Section 4.3. Let us briefly recall the algorithm. During each phase, the node may receive the colors of some of its neighbors. If, at the end of the phase, a conflict between its current color and the color received by a neighbor is detected, the node selects a new color at random. The duration of a phase is set to $\mathcal{O}(\Delta)$ in our analysis, however, to determine the optimal phase length experimentally we use the parameter factor (as Rand4DColoring makes no explicit use of fast local broad-

casting this parameter is otherwise unused for RAND4DCOLORING). We illustrate the execution of the algorithm in Figure 5.1 on the grid deployment to increase the readability. We use the parameters as described in Section 5.3 and refer to Section 2.3 for a more detailed visualization of a network using the grid deployment.



(a) After deployment (b) Resolving conflicts (c) All colors are valid

Figure 5.1: Illustration of an execution of RAND4DCOLORING. Black nodes have a valid color, red nodes are in conflict with one of their neighbors. We observe that even directly after deployment not too many conflicts exist (left). Furthermore, they get gradually eliminated (center) until all nodes have a valid color (right)

Apart from RAND4DCOLORING we consider some variants of the algorithm. The first variant aims at respecting colors already selected in the neighborhood of a node. This is done by storing the latest color received by each neighbor and selecting a new random color so that the stored colors of the neighbors are not selected. Note that the neighbor colors may not reflect the current color of the neighbors. We add a waiting period before starting the color selection phases of the algorithm to receive the neighbors' colors. Thereby, a node waking-up in a validly colored network does not introduce a conflict to any of the nodes that already selected a valid color. We denote this variant by RAND4DRESPECTCOLORING and expect it to be robust against the late wake-up of nodes.

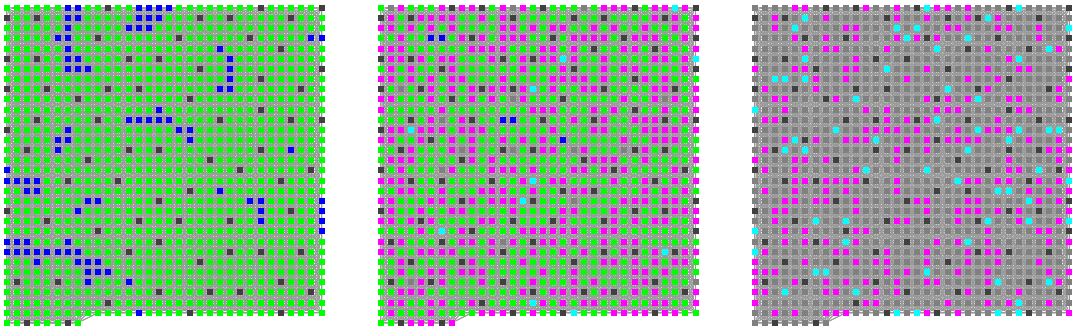
We do also consider a variant that finalizes the selected color after it did not receive a color conflict for at least DURATION time slots. This enables each node of the algorithm itself to decide when the coloring algorithm is completed. We call this variant RAND4DFINALCOLORING.

In another variant we consider the number of available colors as a parameter c. Thereby we can reduce or increase the number of colors used by the algorithm arbitrarily, even despite the fact that the theoretical analysis holds only for $c > 4\Delta$ colors. We refer to the variant with $c = \Delta + 1$ as RAND1DCOLORING.

5.2.2 COLORREDUCTION

Our color reduction algorithm COLORREDUCTION is described in Section 4.5 for the asynchronous case, which we consider here. The algorithm first computes a (first level) MIS to determine independent leaders. Then all non-leaders request an

active interval from the leaders, during which the non-leader nodes then repeatedly execute the faster (second level) MIS algorithm. Once a non-leader node v is in the independent set, it selects a color from the set of free colors F_v, transmits the color to all its neighbors and resigns from the independent set. The initial color is used to get a slot in one of the leaders schedules. These schedules achieve that few nodes compete in the second level MIS, which allows making these MISs very fast. We experimentally determine the parameter FACTOR to speed up, for example, the second level MIS executions in Section 5.3.2. We visualize an execution of COLORREDUCTION in a network with 1000 nodes deployed according to the grid deployment strategy in Figure 5.2.



(a) After leader election (b) Second level starts (c) Most nodes finished

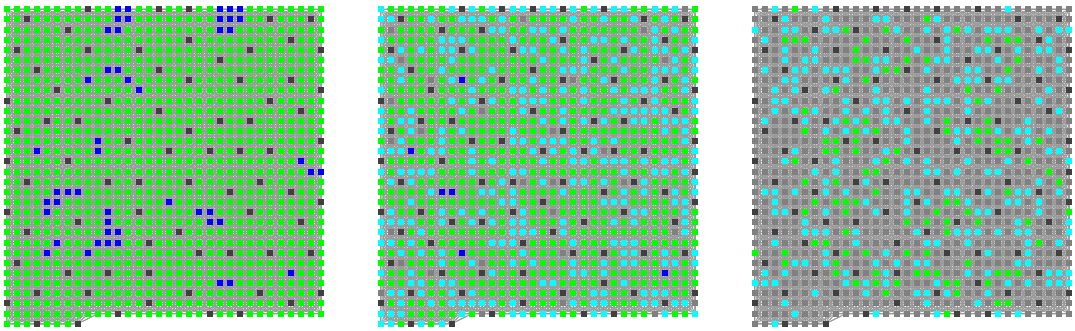
Figure 5.2: The flow of COLORREDUCTION. On the left most leaders (black) are computed. Green nodes are dominated and blue nodes still execute the first level MIS. In the center many dominated nodes received their active interval (magenta nodes), some cyan nodes compete in the second level MIS for a color. Very few gray nodes have already finalized their color by winning a second level MIS. On the right, most nodes selected their final color (gray), while others wait for their active interval or actively compete for selecting a final color.

We consider two variants of COLORREDUCTION in the following. The first variant does not use a valid color but each node simply selects a random number from the set of available colors. We denote this variant by CRRANDCOLOR. Another variant aims at improving the runtime by combining COLORREDUCTION with ideas for RAND4DCOLORING. Namely, we decrease the time accounted for successful transmission of messages, and resolve the introduced conflicts afterwards. Resolving a conflict can be done by resetting a non-leader node to the state in which the node waits for its active interval and compute a new valid active interval based on its previous active interval and the schedule length. For leader nodes we use the simpler strategy of reseting to a random color from F_v, to prevent issues that arise once leaders may resign from their duties. We denote this heuristic as CRRCORRECTING, and shall only use it as an extension to CRRANDCOLOR.

5.2.3 MWCOLORING

We implement the MW-Coloring algorithm as described by Derbel and Talbi [30] for the SINR model and denote it by MWCOLORING. As in the other algorithms

we experimentally determine the parameter FACTOR to ensure a good tradeoff between running time and average number of conflicts in the network. We replaced probabilities, constants and timers accordingly. The algorithm proceeds as follows: First the nodes compete to be in an MIS. The nodes in the MIS become leaders and select color 0, while the remaining nodes request a continuous interval of colors from a selected leader. Once this color interval is received, the node competes in another MIS against at most constantly many neighbors for a color. If the MIS is won, it selects the color, otherwise it moves on to the next color in its color interval and competes again. A notable difference between COLORREDUCTION and MWCOLORING is that the number of time slots required for the color-competing MIS in MWCOLORING is significantly higher than the second level MIS in COLORREDUCTION, however, a lot less of these slower MISs are executed. We illustrate the flow of the algorithm MWCOLORING in Figure 5.3.



(a) After leader election (b) Compete for color (c) Nodes finalize color

Figure 5.3: One execution of MWCOLORING. On the left almost all leaders (black) are computed. Green nodes are dominated and request a color block, while blue nodes still compete in the MIS to become leaders. In the center many nodes received their color blocks to compete for a color (cyan) and some nodes already selected their final color (gray). On the right all leaders are computed and more nodes selected their final color.

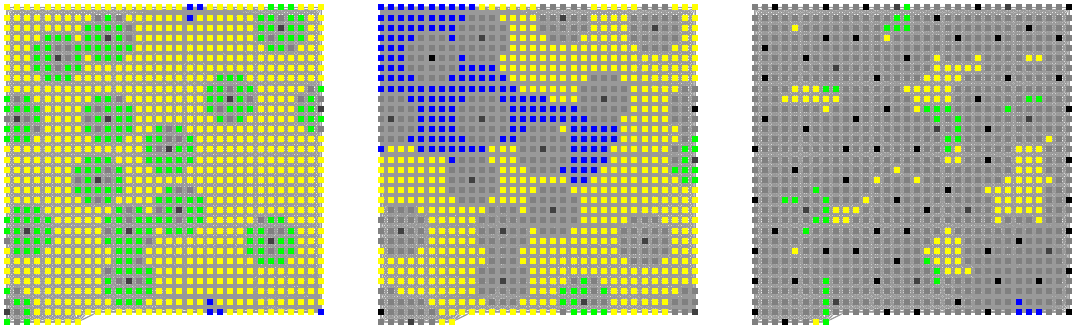
Similarly to our color reduction algorithm, the MW-Coloring algorithm does not account for faults as the correctness holds with high probability. Again, to potentially increase the performance of the algorithm we consider a heuristic that corrects occurring errors once they are detected. To correct the errors, a non-leader node that detected a conflict is reset to compete for the first color block it received. Leader nodes that detect a conflict select a new random color from $[\Delta]$. We refer to this variant as MWCORRECTING.

5.2.4 YUCOLORING

The coloring algorithm by Yu et. al [136] computes a $(\Delta+1)$-coloring in $\mathcal{O}(\Delta \log n + \log^2 n)$ time slots. The main idea behind achieving the optimal number of $\Delta+1$ colors in their algorithm is to increase the transmission power in order to coordinate the color selection process within several hops. Therefore, the nodes in the network

have two transmission powers, which results in two transmission ranges, r_1 and r_2, where r_1 is essentially equivalent to the regular broadcasting range defined in Section 2.2.3. As $r_2 = 3r_1$, nodes with the increased transmission power can reach all nodes within three regular broadcasting ranges.

The algorithm itself works as follows: First, the nodes compute an MIS with respect to r_2. All nodes in the MIS transmit a so-called DoNotTransmit-message to all nodes within r_2. Thereby the nodes within the range r_2 enter a blocked state S, which they only leave once they receive a StartTransmit-message or a StartColoring-message. The nodes in the MIS transmit a StartColoring-message, however, only to the nodes within the smaller range r_1. These nodes start with the color selection process by transmitting an AskColor-message to their MIS node. The MIS node coordinates the requests and allows one after the other to select the smallest color not taken by a neighbor. The colors can be selected without a conflict, as all close-by nodes are either coordinated by the MIS node or are in the blocked state S. Naturally, once a color is selected by a node, the node informs all its neighbors about its color selection. We show the status of nodes during the execution of the algorithm in Figure 5.4.



(a) First leaders (b) Second MIS starts (c) Many nodes colored

Figure 5.4: In the illustration of YuColoring the MIS is regarding a larger range. Thus, on the left only few leaders (black) dominate green nodes that request permission to select a color. Some nodes already selected their final color (gray). Most nodes are blocked (yellow), while some blue nodes still compete to enter the MIS. In the center, the first MIS nodes start to resign as leaders, allowing formerly blocked node to compete in the MIS again (blue nodes). On the right, only few nodes remain to be colored, some wait to receive the permission by their leader, most others are currently blocked.

In the setting Yu et. al designed the algorithm for, the nodes are not given a linear estimate of the maximum degree Δ. Thus, for the algorithm to ensure successful communication, a more complicated slow-start mechanism is used for each transmission. In order to circumvent this for our experiments, we adapt the algorithm to the case of known Δ in this section. This results in a $(\Delta+1)$-coloring algorithm with runtime $\mathcal{O}(\Delta \log n)$ time slots. Compared to ColorReduction, this variant runs under the same assumptions but additionally requires the nodes to increase their transmission power. We shall call the variant described in the following YuColoring.

Theorem 5.1. YuColoring *computes a* $(\Delta+1)$*-coloring in* $\mathcal{O}(\Delta \log n)$ *time slots if all nodes know the maximum degree* Δ*.*

Algorithm 5.1: YuColoring for node v

Continuously:
if *Received DoNotTransmit*$_u$ **then** $F_v \leftarrow F_v \cup \{u\}$ and transit to *state* $\leftarrow$ blocked
if *Received Color*$_u(c)$ *from node* u **then** $C_v \leftarrow C_v \cup \{c\}$
switch *state* **do**
  case *start*
    wait for $\mathcal{O}(\Delta \log n)$ time slots
    transit to *state* $\leftarrow$ MIS
  case *blocked*
    if *Received StartColoring*$_u$ **then** transit to *state* $\leftarrow$ C1
  case *MIS*
    We use MIS($\ell = 1$) from ColorReduction (Algorithm 4.5).
    Successful nodes transit to *state* $\leftarrow$ leader
  case *leader*
    Transmit DoNotTransmit$_v$ with range r_2 and prob. p_{high} for $\mathcal{O}(\log n)$ slots
    select color 0
    Transmit StartColoring$_v$ with range r_1 and prob. p_{high} for $\mathcal{O}(\log n)$ time slots
    if Q *not empty* **then**
      $u \leftarrow$ Q.pop()
      Transmit Grant$_u$ with range r_1 and prob. p_{high} for $\mathcal{O}(\log n)$ time slots
    else
      Transmit StartColoring$_v$ with range r_1 and prob. p_{high} for $\mathcal{O}(\log n)$ slots
  case *C1*
    Transmit AskColor$_v$ with range r_1 and prob. p_{low} for $\mathcal{O}(\Delta \log n)$ time slots
    if *Received Grant*$_u$ **then** transit to *state* $\leftarrow$ C2
  case *C2*
    select smallest color c not in C_v
    Transmit Color$_v(c)$ with range r_1 and prob. p_{high} for $\mathcal{O}(\log n)$ time slots

The correctness essentially follows from the correctness of the original algorithm. For the argument to be more concise, we elaborate on the main points in the following and give a pseudocode of YuColoring in Algorithm 5.1.

The coloring is valid: Let us consider a node v and assume its coloring is not valid due to a conflict with its neighbor u. If v has color 0, it is a leader node and the conflicting node u must have been one of the nodes v dominated. Thus,

with high probability, u received the DoNotTransmit and the StartColoring, afterwards transmitted AskColoring itself and received a Grant message - leading to the selection of another color $c \neq 0$. If v's color is not 0, it selected its color during such a color selection process itself. As both v and u successfully transmit their color after selection with high probability and neighbors respect this selection, it must be the case that v and u selected the color simultaneously. Since the leader nodes wait long enough between transmitting the Grant message to the two nodes, this can only happen if v and u listen to two different leaders. This, however, is not possible as all nodes within at least two broadcasting ranges of v received the DoNotTransmit message of v's leader with high probability.

All nodes get colored: Essentially this holds as each node v is either in the MIS (with respect to r_2) at some point or one of its neighbors is in the MIS and allows v to select a color.

The runtime of the algorithm is $\mathcal{O}(\Delta \log n)$ time slots: Let us consider the maximum time until a node v or one of its neighbors is in the MIS. Remember that the MIS is computed with respect to the range r_2, while the neighborhood relation we consider for the coloring is relative to r_1. As each r_1-neighborhood of a MIS node is colored after $\mathcal{O}(\Delta \log n)$, this is also the asymptotic time that passes between the MIS node transmits DoNotTransmit and StartTransmit. As at most 36 nodes can be independent regarding the range r_1 in the r_2-range of a node v, after at most 36 rounds of the MIS (and potentially the following blocked/coloring state) all nodes in the r_2-range of v must have either been in the MIS or are neighbors of an MIS-node. Thus, either v or one of its neighbors wins the MIS competition and starts the coloring routine afterwards. Overall, this results in a runtime of $\mathcal{O}(\Delta \log n)$ time slots.

Regarding our experiments, we experimentally determine the parameter FACTOR to achieve the best runtime for this algorithm. The maximum node degree, which is used to calculate the transmission probability, is determined with respect to the regular transmission range r_1.

The algorithm heavily relies on the blocking of nearby nodes to prevent neighboring nodes to select an invalid color as neighbors are not allowed to select colors simultaneously. Note, however, that it cannot be guaranteed in our simulation (or in practice) that transmissions are successfully received. Thus, the blocking of other nodes can be error-prone. If a node receives the DoNotTransmit message but fails to receive the StartTransmit message, it will not leave the blocked state. For the algorithm to finish properly, we consider such nodes as finished with color 0. This and other problems that come with potentially unreliable message transmission may lead to color conflicts. To reduce the number of color conflicts, we additionally consider a variant YUCORRECTING. If a color conflict is detected after a final color was selected, the node revokes its color and restarts the algorithm by competing in the MIS.

Table 5.2: Parameters that achieve successful local broadcasting in the different distributions. R=Random, G=Grid, PG=PerturbedGrid, C=Cluster

Distribution	R	G	PG	C	C&R	C&G	C&PG
txConst	0.15	0.15	0.10	0.30	0.25	0.20	0.20
duration	4600	3400	4900	12 900	8100	8200	8100

5.3 EXPERIMENTS

We conduct several experiments to evaluate the performance of the different algorithms. We measure the time required to compute a valid coloring and the number of nodes that were not able to select a valid color. As described in Section 2.3 we use one time slot as the time required for one transmission. To measure the runtime of our algorithms, we deploy the nodes simultaneously in the area and start the algorithms asynchronously after a waiting period that is chosen uniformly at random between 0 and 10 time slots (using real numbers) for each node. The runtime measurement starts with the deployment of the nodes and ends once all algorithms are in a finished state or all nodes in the network have selected a valid color. Note that the time slots of the nodes are not synchronized and may overlap partially. To measure the number of nodes with a valid and an invalid color, each node notifies the simulator whenever it selects a new color, which is then checked for validity with colors selected by the neighbors. This is done within the simulation framework, thus we do neither use messages nor tell the nodes about the result of this color inspection.

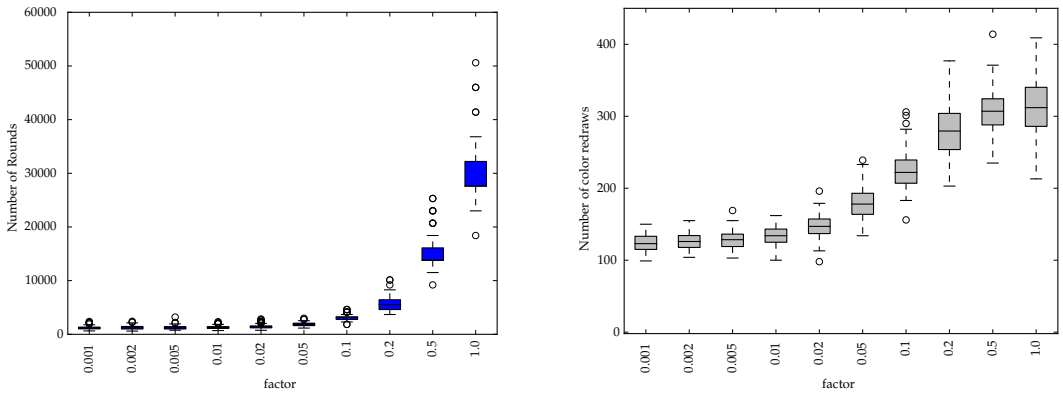
To implement the message transmission in the algorithms, we use local broadcasting, and therefore set the transmission constant txConst and the broadcast duration duration as determined in Section 2.3.1. Although Rand4DColoring does not use local broadcasting we use the same txConst, to increase comparability of the algorithms. We restate the main parameters in Table 5.2

Let us now briefly remind us of the setting and the parameters used in the simulation, for more details we refer to Section 2.3. We use the network simulator Sinalgo [33] to (usually) simulate 1000 nodes that are deployed on an area of 1000 m $\times$ 1000 m according to some deployment strategy. We use Random, Grid, PerturbedGrid (PGrid), and Cluster deployment, as well as mixed variants thereof such as Cluster&Random, Cluster&Grid, and Cluster&PGrid, in which 50 % of the nodes are deployed according to each of the combined deployment schemes. For each experiments we use 100 runs on the same pre-computed deployments. To model interference we use the standard geometric SINR model, which considers the interference of all concurrent transmissions in the network when deciding whether a certain transmission can successfully be received or not. By the SINR constraints we get a maximum transmission range of 100 m (cf. Section 2.2.3). As common in the SINR model, we set the broadcasting range, which defines the neighborhood relation, as a fraction of the transmission range to 84 m.

In the following sections, we determine optimal parameters for the algorithms and compare the algorithms. In the next section we determine the parameter FACTOR for RAND4DCOLORING and evaluate the algorithm and its variants in the described setting. Afterwards, we determine optimal parameters for COLORREDUCTION, MWCOLORING, and YUCOLORING in Sections 5.3.2 to 5.3.4, before considering heuristic improvements to these algorithms in Section 5.3.5. We compare the best results obtained for each variant in Section 5.3.6. Afterwards we study the performance of the algorithms if mobility of nodes is allowed in Section 5.3.7 and if large parts of the network wake up after the remaining part finished the algorithm in Section 5.3.8.

5.3.1 RAND4DCOLORING

In contrast to the other coloring algorithms that we evaluate, RAND4DCOLORING does not rely on any structure that has to be built or maintained. Instead, it is a simple phase-based algorithm that selects a new random color at the end of each phase if a conflict is detected during the phase. In the theoretical analysis, the length of each phase is set to $\mathcal{O}(\Delta)$ to show that the probability for a conflict decreases in each phase. Instead of setting the length of a phase to the maximum degree, we use the parameters DURATION and FACTOR to experimentally determine the optimal phase-length. Therefore, we set the DURATION according to the runtime of one local broadcast as observed in Section 2.3.1 and use DURATION $\times$ FACTOR as the length of each phase. Recall that the nodes do not finalize their color while executing this algorithm. Thus, we measure the number of nodes with a valid color and stop the simulation once all nodes have a valid color. Let us study the parameter FACTOR using the random deployment in our first experiment. We use values ranging from 0.001 to 1, corresponding to phase-lengths between 5 and 4600 time slots. Our results are depicted in Figure 5.5.



(a) Runtime

(b) Total number of color redraws

Figure 5.5: Determining the parameter FACTOR for RANDCDELTACOLORING, which influences the length of each phase. Both the runtime (left) and the number of color redraws (right) increase with increasing FACTOR.

We observe that both the runtime and the number of color redraws increases with the phase-length. Especially for the runtime this was expected, as our theoretical analysis guarantees the runtime of $\mathcal{O}(\Delta \log n)$ time slots only for a phase-length of $\mathcal{O}(\Delta)$, while FACTOR $= 1$ sets the phase-length to one round of local broadcasting, which is asymptotically in $\mathcal{O}(\Delta \log n)$. However, there is also a less formal intuition justifying the decreasing runtime for the decreasing phase-length. Once a node has detected a conflict (by receiving a message from a neighbor), it is not beneficial if the node must wait for the end of the phase before changing its color. Assume the node waits until the phase ends. It may happen that the node transmits its color to its neighbors, which may become aware of a conflict. If this happens, the respective neighbors also reset their color at the end of their phase, although this conflict would be resolved without the neighbors intervention with significant probability at the end of the phase. On the other hand, dealing with the conflict directly does not introduce any penalty, as it is already determined that the detecting node resets its color. Hence, the shorter the phases are, the lower the runtime of the coloring algorithm.

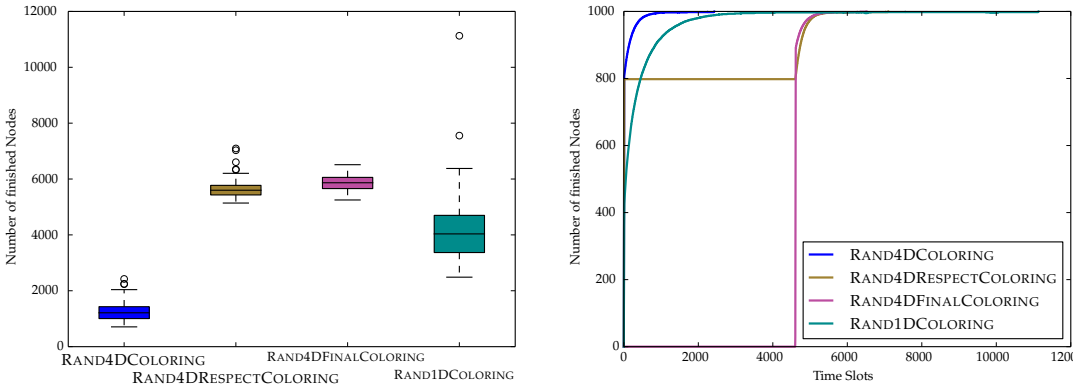
Regarding the number of color redraws, we can observe something interesting. The longer the phases are, the more redraws are required, cf. Figure 5.5b. This corresponds to the fact that the longer the phases are, the higher the probability that all conflicts are detected by the nodes. If a conflict is only detected by one of the conflicting nodes, the probability that it is resolved is already significant. Therefore, using phases of minimal length intuitively reduces the number of color redraws by a factor of two compared to phases of length DURATION. As each color redraw leads to some possibility of selecting the color of a neighbor we observe a factor of even slightly more than two in the total number of color redraws for long phases. Note that even longer phases do not lead to a further increase, as almost all conflicts are detected after phase-lengths that correspond to FACTOR $= 1$.

Let us now compare the different variants of RAND4DCOLORING we described in Section 5.2.1. We use FACTOR $= 0.001$ as determined in the previous experiment. The result of this comparison is given as Table 5.3 and visualized in Figure 5.6a for the random distribution. We defer the results for the remaining deployments to Table A.1 in Appendix A.3. We can clearly see that the basic

Table 5.3: Comparison of average runtime and average number of conflicts of our RAND4DCOLORING variants

	runtime	conflicts
RAND4DCOLORING	1256	0.00
RAND4DRESPECTCOLORING	5668	0.00
RAND4DFINALCOLORING	5865	0.00
RAND1DCOLORING	4174	0.00

RAND4DCOLORING algorithm is the fastest with a runtime of only 1256 time slots. This was expected as the variants either improve the resulting coloring or make the



(a) Runtime comparison

(b) Number of finished nodes over time

Figure 5.6: Comparison of our RANDCOLORING variants regarding runtime (left) and progress (right).

algorithm more robust regarding a specific setting. To explain the runtime of the variants RAND4DRESPECTCOLORING and RAND4DFINALCOLORING, let us consider Figure 5.6b, in which the average progress of the variants is illustrated by showing the average number of finished nodes. For these algorithms the number of finished nodes corresponds to the number of nodes with a valid color, as these variants do never finalize their color. Only for RAND4DFINALCOLORING the value corresponds to the number of nodes that finalized their color.

We observe in Figure 5.6b that while RAND4DCOLORING and RAND1DCOLORING are able to make fast progress after the algorithms started, the two remaining variants do not resolve a conflict or finalize a color for a little more than 4000 time slots before finishing within the next 1000-2000 time slots. This behavior is due to two reasons. For RAND4DRESPECTCOLORING, which is expected to be more robust in the case of heterogenous wake-up patterns, this is caused by a listening period, during which the colors of neighbors are received and stored. Afterwards, this variant selects its color so it does not coincide with the latest received color from any of the neighbors. For RAND4DFINALCOLORING it is due to a waiting period before finalizing the current color. This waiting period of a node is reset with each conflict it detects and allows the node to decide when a selected color can be finalized. In both algorithms the waiting or listening period is set to exactly DURATION time slots. Finally, RAND1DCOLORING decreases the number of used colors to $\Delta + 1$, resulting in a runtime of 4174 time slots. Note that reducing the number of colors to $\Delta + 1$ results in a higher variance in the runtime, as the probability to select a color of one of the neighbors increases.

In the following three sections, we consider the basic variants of the remaining algorithms, before considering the corresponding heuristic improvements in Section 5.3.5, and finally comparing the results of the algorithms in Section 5.3.6.

5.3.2 ColorReduction

Let us now consider the algorithm ColorReduction, which assumes a given $\mathcal{O}(\Delta)$-coloring to compute a coloring with $\Delta + 1$ colors. To achieve its runtime of $\mathcal{O}(\Delta \log n)$ it uses two levels of MIS that achieve a coordinated color selection. The nodes in the first level MIS are the leaders, which independently decide on a schedule and coordinate the activity of the dominated nodes according to this schedule. The dominated nodes select a neighboring leader to request an active interval based on its initial color, during which they compete in the second level MIS for the authorization to select a final color. For more details on the algorithm we refer to Section 4.5. We set the length of an active interval to DURATION $\times$ FACTOR, which is exactly the time required for a fast local broadcast. The length of the schedule is based on the maximum degree in the network and the length of each active interval and set to $\Delta \cdot$ (DURATION $\times$ FACTOR). Each leader sets the start of the schedule independently and selects the active interval of the dominated nodes based only on their initial color and schedule.

In the following we determine the parameter FACTOR. For this and the remaining algorithms, FACTOR specifies the factor between local broadcasting with all nodes and fast local broadcasting with a small subset of the nodes. For fast local broadcasting the transmission probability is increased by a multiplicative factor of $\Delta \times$FACTOR, while the number of time slots accounted for it is divided by $\Delta \times$FACTOR, cf. Table 5.1. As the optimal value of FACTOR may depend on the size of the sets, how often this mode of transmission is used, and possibly other factors, we determine this parameter for each of the algorithms separately. The results for ColorReduction using 1000 nodes and the random deployment scheme are given in Table 5.4 and illustrated in Figure 5.7. The values for the other deployment strategies can be found as Table A.2 in Appendix A.3.

Table 5.4: Average number of conflicts and average runtime for ColorReduction using different parameters FACTOR.

FACTOR	0.05	0.1	0.2	0.3	0.4	0.6	0.8
ColorReduction							
conflicts	0.00	0.04	0.10	0.00	0.12	**0.51**	2.47
runtime	339 013	171 099	87 924	59 995	46 266	**32224**	25 384

We observe that the number of conflicts increases slowly with the parameter FACTOR. For FACTOR $= 0.6$ the average number of conflicts is only 3.1. On the other hand, the number of time slots decreases significantly for an increasing FACTOR, as the fast local broadcasting becomes faster. For FACTOR $= 0.6$ the number of nodes without valid color is only 0.51 on average while achieving a competitive runtime. Thus, we use this value in the following. This leads to an average runtime of 32 224 time slots, with 0.51 conflicts in the computed $(\Delta + 1)$-coloring for 1000 nodes on average.

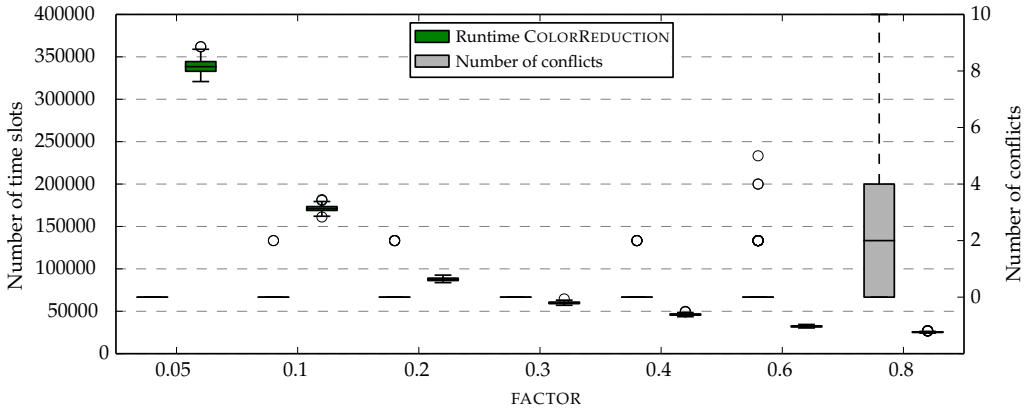
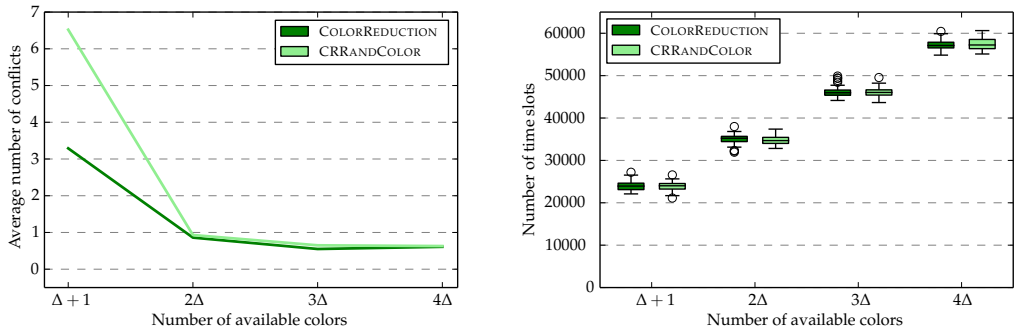


Figure 5.7: Runtime and number of conflicts of ColorReduction for different values of the parameter FACTOR.

Let us now consider our first variant CRRandColor, which in contrast to its basic algorithm ColorReduction simply draws a random number from the a set of colors. For the basic algorithm on the other hand the number must correspond to a valid node coloring. To compare the algorithms we consider various sets of colors. Note that the valid color is pre-computed at no cost for the algorithm for ColorReduction, while in CRRandColor each node draws a random color, regardless of whether neighbors select the same color or not. We measure the average runtime and number of conflicts for the sets $\{0, 1 \ldots, c \cdot \Delta\}$, with $c = 1, \ldots, 4$[1]. We depict the results in Figure 5.8 and Table 5.5 for the random deployment. The results for the remaining deployments are similar and deferred to Table A.3 in Appendix A.3.



(a) Number of conflicts

(b) Runtime

Figure 5.8: Comparing ColorReduction with its variant CRRandColor. CRRandColor does not use a valid coloring but simply random number from the specified set, but performs almost as good as ColorReduction.

The results indicate that ColorReduction does not require a valid coloring to perform well in practice. We see a significant increase in the number of conflicts

1 For $c > 1$ we write $c\Delta$ instead of $c(\Delta + 1)$ for brevity.

Table 5.5: Comparing ColorReduction and CRRandColor for a varying number of colors in the used coloring.

Number of colors	$\Delta+1$	2Δ	3Δ	4Δ
ColorReduction				
conflicts	3.29	0.86	0.55	0.61
runtime	18 638	24 824	32 197	39 737
CRRandColor				
conflicts	6.52	**0.93**	0.65	0.63
runtime	19 766	**24758**	32 287	39 696

Table 5.6: Average number of conflicts and average runtime for MWColoring using different parameters factor.

factor		0.05	0.1	0.2	0.3	0.4	0.6
MWColoring	conflicts	0.1	0.1	**0.4**	1.0	1.5	2.7
	runtime	81 195	44 700	**27982**	23 995	22 807	21 870

for CRRandColor only for colorings of cardinality $\Delta+1$, however, assuming a valid $\Delta+1$ coloring to be given renders executing the algorithm unnecessary. For colorings of size larger than 2Δ the difference in the number of conflicts and the runtime of the algorithms is negligible. Also, we observe that the smaller the color set the faster the algorithm. As computing a valid coloring additionally requires some effort, we focus on the variant CRRandColor with a random (2Δ)-coloring in the following. We consider the correcting variants of CRRandColor, MWColoring, and YuColoring combined in Section 5.3.5, thus we first determine the parameters of MWColoring and YuColoring in the next sections.

5.3.3 MWColoring

The MWColoring algorithm was first proposed for the protocol model before being adapted to the SINR model. Nevertheless it was the first distributed node coloring algorithm proven to be feasible in the SINR model. It computes an $\mathcal{O}(\Delta)$ coloring in $\mathcal{O}(\Delta \log n)$ time slots.

To achieve an optimal runtime for the algorithm, we experimentally determine the parameter factor in the following. This parameter is used, in addition to txConst and duration to enable the nodes to successfully transmit messages to their neighbors. The average number of conflicts and the average runtime of MWColoring in the random deployment setting is given in Table 5.6, results for the other deployments are deferred to Table A.4. We notice that the increase in the number of conflicts introduced with the increase of factor is moderate. Even with

Table 5.7: Average number of conflicts and average runtime for YuColoring using different parameters factor.

factor		0.05	0.1	0.2	0.3	0.4	0.6
YuColoring	conflicts	0.6	0.7	**1.4**	2.9	6.4	22.4
	runtime	286 167	160 088	**99946**	82 707	72 660	67 131

factor $= 0.6$ only about 3 of the 1000 nodes do not have a valid color on average. The average number of time slots required for the algorithm to finish decreases by almost a factor of three from factor $= 0.05$ to factor $= 0.2$, while introducing an average number of conflicts of only 0.4. Thus, we use factor $= 0.2$ for this algorithm, resulting in a runtime of 27 982 time slots to compute an $\mathcal{O}(\Delta)$ coloring with 0.4 conflicts on average.

5.3.4 YuColoring

Let us now consider the YuColoring algorithm. As for MWColoring, we determine the optimal parameter factor of the algorithm. Therefore we test parameter values between 0.05 and 0.6 using the random deployment. The results are shown in Table 5.7 and for the remaining deployments in Table A.5. As for ColorReduction and MWColoring, the runtime decreases with an increasing factor, while the number of conflicts increases. Therefore, we again set factor $= 0.2$, as afterwards the decrease in runtime becomes slower while the increase in the number of conflicts seems to accelerate. With this choice the algorithm achieves a runtime of 99 946 time slots with an average of 1.4 conflicts in a network of 1000 nodes deployed uniformly at random.

5.3.5 *Correcting Variants*

After determining optimal parameters for all algorithms we now consider heuristic improvements to CRRandColor, MWColoring, and YuColoring, namely CRRCorrecting, MWCorrecting, and YuCorrecting. In these heuristics, we aim at making the algorithms both faster and more robust towards failures in the communication. Therefore, instead of ignoring detected color conflicts as it is mostly done in the basic algorithms, we actively deal with them and try to resolve the conflicts. All nodes use selected leaders to coordinate the color selection process of the remaining nodes. Therefore, if a color conflict is detected for a non-leader node, we simply reset the node to a state of the algorithm in which it can select a valid node again. For CRRCorrecting we reset a node that detected a conflict to wait for its active interval, while computing its next valid active interval based on its previous active interval and the schedule length. For MWCorrecting we go back to the first non-leader MIS, and for YuColoring we reset to the start of the algorithm as the original leaders may not be serving as leaders anymore. To reset

a leader node is more involved, as there may be non-leader nodes depending on it. Thus, we combine the algorithms with ideas of RAND4DCOLORING and allow the leader node to simply select a new color. As the colors of neighbors are recorded in COLORREDUCTION, we try to select a color that is not on the record for CRRCORRECTING. In MWCORRECTING we select a random color from the set $[\Delta]$, as those colors are not used for non-leaders. We do the same for YUCORRECTING, although the colors are also used by non-leaders in this algorithm.

It is obvious that, although the number of conflicts may be reduced, the runtime of the algorithm increases for these variants. However, as the algorithms are able to detect and resolve conflicts, we reduce the time accounted for a successful transmission to decrease the runtime while introducing some (hopefully temporary) conflicts. We do this by reducing the parameter DURATION to a fraction of the value determined using local broadcasting (cf. Table 5.2). We report the results for the random deployment in Table 5.8 and Figure 5.9. We denote the fraction of DURATION used in the experiment as DURATION′. Recall that DURATION = 4600 for the random deployment strategy.

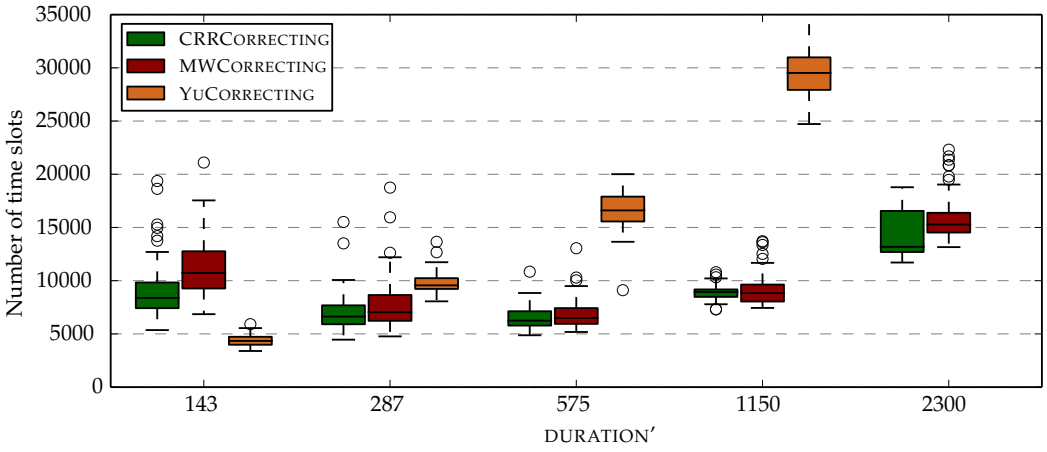


Figure 5.9: Runtime of the correcting variants CRRCORRECTING, MWCORRECTING, and YUCORRECTING. We omit DURATION′ = DURATION = 4600 and limit the number of time slots to 35 000 for better readability.

For the standard case of unchanged DURATION we observe that the average number of conflicts decreases from 0.93, 0.4 and 1.4 (cf. Tables 5.5 to 5.7) to 0.00 for CRRCORRECTING, MWCORRECTING and YUCORRECTING, while the runtime increases as expected. However, using a smaller parameter DURATION′ the correcting variants are able to compute a coloring with less conflicts in time less than the respective basic algorithms. For CRRANDCOLOR the correcting variant achieves to compute a coloring without conflicts even for very small values of DURATION′. The best runtime is obtained using DURATION′ = 575, for which the algorithm computes a $(\Delta + 1)$-coloring in 6489 time slots. Similarly, MWCORRECTING achieves to $\mathcal{O}(\Delta)$-color the network even for the smallest considered DURATION′ values essentially without a conflict. The (very) small number of average conflicts we observed is probably due to conflicts that were not yet detected. As CRRCORRECTING prevents

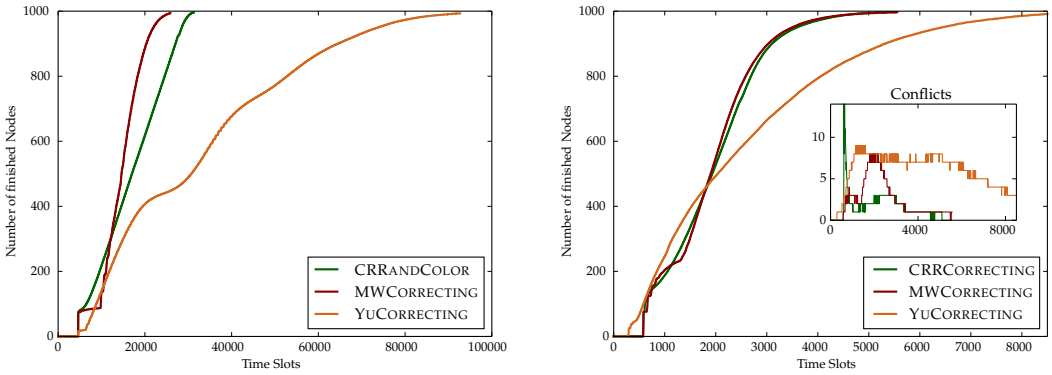
Table 5.8: Average runtime and conflicts by the correcting variants. We used varying fractions of the DURATION parameter (denoted by DURATION′) and the random deployment.

Fraction of DURATION		1/32	1/16	1/8	1/4	1/2	1
Resulting DURATION′		143	287	575	1150	2300	4600
CRRCORRECTING	conflicts	0.00	0.00	**0.00**	0.00	0.00	0.00
	runtime	8965	6984	**6489**	8883	14 348	25 218
MWCORRECTING	conflicts	0.13	0.23	**0.10**	0.02	0.02	0.00
	runtime	11 065	7688	**6834**	9105	15 762	31 027
YUCORRECTING	conflicts	4.62	**1.23**	0.28	0.09	0.00	0.00
	runtime	4370	**9807**	16 652	29 635	58 189	116 646

creating new conflicts by storing the neighbors current colors, such conflicts occur more frequently in MWCORRECTING. The best runtime of MWCORRECTING is also achieved for DURATION′ = 575, resulting in 6834 time slots. YUCORRECTING, on the other hand, does not achieve a coloring without conflict for the smaller DURATION′ values. This is due to the blocking of nodes implemented by the leaders that transmit DoNotTransmit messages. For very small DURATION′ values, some nodes are not able to receive the StartTransmit message and remain blocked for the execution of the algorithm. The best runtime is achieved for DURATION′ = 287 with only 4370 time slots, however, leading to an average number of conflicts of 4.62. Thus we select DURATION′ = 287, resulting in an $(\Delta + 1)$-coloring algorithm with an average number of 1.23 conflicts after 9807 time slots (which is still an order of magnitude faster than the basic algorithm).

5.3.6 *Performance Comparison of Coloring Algorithms*

Before comparing the runtime of all algorithms, we briefly study the progress of the algorithms CRRANDCOLOR, MWCOLORING, and YUCOLORING and their correcting variants. The average progress (i.e., the average number of finished nodes over time) of all algorithms is illustrated in Figure 5.10. Note that the average progress of RAND4DCOLORING and its variants is already discussed in Section 5.3.1. The first, sudden increase that can be observed for all algorithms is exactly after DURATION (or DURATION′) time slots. This is when the first nodes enter the MIS, become leader and select their color. As the MIS is computed regarding a three times larger broadcasting range in YUCOLORING, a lot less nodes enter the MIS in this algorithm. Some time after the MIS computation, the nodes around the leaders select their colors. While this allows both CRRANDCOLOR and MWCOLORING to gradually color all nodes in the network, several plateaus can be observed for YUCOLORING. This is due to the fact that in YUCOLORING not all nodes are able to select a color after the first MIS execution, as the MIS is computed regarding



(a) Progress of basic algorithms

(b) Progress of correcting variants

Figure 5.10: Progress of the algorithms ColorReduction, MWColoring, YuColoring (left) and their correcting variants (right). For the correcting variants we additionally depict the average number of conflicts.

a larger broadcasting range than the neighborhood around the leaders which is allowed to select a color. Thus, we see at least two more MIS executions leaving their traces in this progress around time slots 20 000 and 40 000 for YuColoring in Figure 5.10a. Although this happens also in YuCorrecting, we cannot clearly observe the moment it happens in the average progress depicted in Figure 5.10.

For CRRandColor and MWColoring, the leaders dominate the whole network, resulting in each node requesting an active interval or a color interval from its respective leader. Let us compare the progress of CRRandColor and MWColoring. Recall that leaders in CRRandColor coordinate the time interval in which dominated nodes compete to select a final color, while in MWColoring the leaders coordinate which colors the dominated nodes compete for. This fundamental difference allows CRRandColor to use fast local broadcasting for the second level MIS (which can be seen as a competition for being allowed to select a valid color). The coordination of active nodes based on the nodes' initial color leads to an almost perfectly linear increase of the number of finished nodes over time. In MWColoring, the nodes compete in fewer but slower MIS-executions for the colors, with leads to the rapid increase once the threshold is reached (in Figure 5.10a at around 9000 time slots). Note that the rapid increase once the threshold is met is due to the requirement of achieving a valid node coloring with very few conflicts, however, it also hints that faster progress is possible, as shown by our correcting variants.

Regarding our correcting variants, we observe that the progress of CRRCorrecting and MWCorrecting is very similar. One reason for this is that both use the same duration' value of 575 time slots. The second reason is that by reducing the time accounted for local broadcasting, all the slack is removed from the algorithms. As both algorithms elect leaders, which then allow a dominated node to either be active or compete for certain colors (depending on the algorithm), the remaining progress essentially shows this similarity.

Inside Figure 5.10b we additionally show the number of conflicts occurring in the respective algorithms over time. We observe that in general, the number of simultaneous conflicts is relatively low with well below 15 conflicts at each time. For CRRCorrecting there is a peak at or around the leader election phase, indicating that too many nodes entered the MIS. A smaller peak is also visible for MWColoring, however, here more nodes fail to select a valid color in the following color competition. For YuCorrecting, the number of conflicting nodes is relatively stable at around 8 simultaneous conflicts. Although the number of conflicts decreases, not all can be corrected, as some conflicts are due to blocked nodes[2].

To compare the performance of the algorithms on the different deployment strategies, we show the runtime and the number of conflicts of the algorithms on the different deployments in Table 5.9. For the random deployment, the values are the best values from Tables 5.3 and 5.5 to 5.8. For the remaining distributions, we select the values from the corresponding Tables A.1 and A.3 to A.7, which we deferred to Appendix A.3.

Considering the conflicts, we aimed at selecting the parameters so that the number of conflicts is low. Only for YuColoring and YuCorrecting we allowed a slightly higher number of average conflicts to reduce the runtime, if possible. Apart from this, the number of conflicts is comparable for the algorithms, thus, we focus on the runtime in the following. The results of the algorithms are very consistent throughout the different deployments. This indicates that the communication parameters are sufficiently well chosen to allow the algorithms to deliver their performance without being constrained by, for example, congestion problems.

We do not depict the basic variant ColorReduction in Table 5.9 as the performance is essentially the same as CRRandColor (cf. Table A.3 in Appendix A.3). We consider CRRandColor, as this variant uses random colors from a set of 2Δ colors instead of a valid coloring. Hence, precomputing a valid coloring is not required. As CRRandColor and CRRCorrecting compute $(\Delta + 1)$-colorings, YuColoring and YuCorrecting are its main competitors. A valid coloring of the same size is also computed by our variant Rand1DColoring that heuristically reduces the number of available colors in Rand4DColoring to $(\Delta + 1)$, however, we discuss this variant later. CRRandColor computes a $(\Delta + 1)$-coloring using between 20 367 and 67 881 time slots. The correcting variant CRRCorrecting reduces the runtime to values between 6489 and 17 802 time slots. This is at par with MWColoring and MWCorrecting, and significantly less than YuColoring and YuCorrecting. The basic algorithm YuColoring requires between 99 946 and 164 839 time slots, while YuCorrecting reduces the runtime to values between 8654 and 20 849 time slots.

Our other algorithm, Rand4DColoring, computes a (4Δ)-coloring, hence MWColoring and its variant MWCorrecting are its main competitors. Depending on the deployment strategy, Rand4DColoring achieves a runtime between 974 and 3321 time slots. This is by far superior to the runtime achieved by both MW-

2 Recall that we set the leader to a quit-state and select color 0 once the leader of a blocked node resigns from its leader functionality (cf. Section 5.2.4).

Table 5.9: Comparison of the average runtime and number of conflicts for all considered algorithms in all considered deployment strategies. R=Random, G=Grid, PG=PerturbedGrid, C=Cluster

Distribution	R	G	PG	C	C&R	C&G	C&PG
Rand4DColoring							
conflicts	0.00	0.00	0.00	0.00	0.00	0.00	0.00
runtime	1256	974	1372	3321	2316	2186	2016
Rand1DColoring							
conflicts	0.00	0.00	0.00	0.00	0.00	0.00	0.00
runtime	4174	3358	4548	11 822	8153	8211	6627
CRRandColor							
conflicts	0.93	0.82	0.10	1.05	1.83	1.03	3.23
runtime	24 758	20 367	27 817	67 881	42 034	42 692	42 385
CRRCorrecting							
conflicts	0.00	0.00	0.00	0.00	0.00	0.00	0.02
runtime	6489	7017	7017	17 802	11 884	11 333	10 896
MWColoring							
conflicts	0.42	0.26	0.28	0.56	0.44	0.34	1.06
runtime	27 982	25 456	32 812	73 670	46 363	47 041	46 142
MWCorrecting							
conflicts	0.10	0.21	0.12	0.45	0.44	0.36	0.42
runtime	6834	5741	7567	23 531	12 780	13 535	13 353
YuColoring							
conflicts	1.39	2.01	1.12	0.9	0.88	0.94	1.30
runtime	99 946	113 105	129 054	164 839	122 267	141 003	126 488
YuCorrecting							
conflicts	1.23	1.77	2.15	0.74	0.81	1.23	0.64
runtime	9807	8654	11 479	20 849	15 060	15 835	14 620

Coloring and MWCorrecting. MWCorrecting requires between 5741 and 23 531 time slots, which is between 4 and 5 times the runtime of Rand4DColoring. MWColoring achieves a runtime between 25 456 and 73 670 time slots. Note that Rand4DColoring does not finalize the colors, however, even using a duration for finalizing the color, the variant Rand4DFinalColoring which finalizes colors, achieves a runtime between 4354 and 16 110 time slots (cf. Table 5.3 in Appendix A.3).

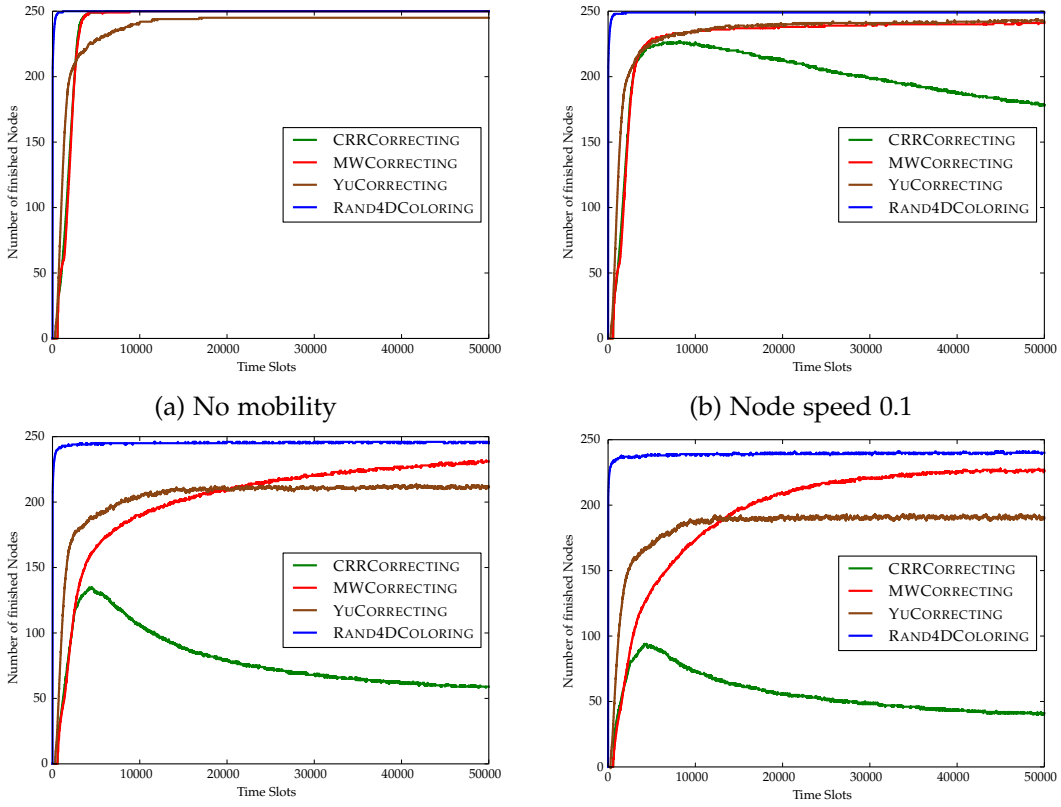
Reducing the number of available colors in Rand4DColoring to only $(\Delta + 1)$ colors yields a $(\Delta + 1)$-coloring heuristic, which we denote by Rand1DColoring. The heuristic selects a random color from $[\Delta]$ by resolving conflicts once detected and requires only between 3358 and 11 822 time slots. Thus it achieves the lowest runtimes of all considered $(\Delta + 1)$-coloring algorithms. As mentioned, however, Rand4DColoring and its variants do not finalize their colors. Hence, if finalization of the colors is required CRRCorrecting achieves the best results for $(\Delta + 1)$-coloring algorithms.

We conclude from our comparison that the best performance for $\mathcal{O}(\Delta)$-colorings is achieved by Rand4DColoring and for $(\Delta + 1)$-colorings by Rand1DColoring and CRRCorrecting, depending on the exact setting.

5.3.7 *Coloring in Dynamic Networks*

After studying the performance of the algorithms regarding runtime and number of conflicts, let us now consider more elaborate scenarios, which require the nodes to react on changes in the network. We consider two scenarios, in the first one we allow the nodes to move, which forces the algorithms to maintain the validity of the coloring in a dynamically changing network. Obviously the algorithms cannot maintain the coloring valid at each node, as the neighborhood changes in an unforeseen manner. Thus, we study how large the fraction of nodes is that maintains a valid color despite the dynamic changes. In the second scenario, considered in the next section, we shall allow a fraction of the nodes to wake up after the remaining part of the network has already selected a color. Thereby we evaluate how well the algorithms can cope with highly asynchronous wake-up schemes. We restrict ourselves to the random deployment for these experiments and do not use pre-computed position files for the wake-up scenario in the next section.

For the dynamic setting we consider only 250 nodes on a deployment area of 500 m $\times$ 500 m, which results in a similar density as in the previously considered settings. We reduce the number of nodes, as mobility increases the time required for the simulation significantly. Due to the high complexity of updating the relevant positions for each single event the used simulation framework Sinalgo additionally requires the synchronous mode, in which time slots of the different nodes are perfectly synchronized. Thus, it is sufficient to update the positions once per time slot. The nodes are deployed uniformly at random and move according to the RandomDirection model, cf. Section 2.3. The nodes alternate between moving and waiting time, both times are drawn randomly and follow a Gaussian distribution with mean value of 100 time slots and a variance of 50. The speed of a node is also drawn at random according to a Gaussian distribution with a mean between 0 and 1 meter per time slot and a variance of 2. For simplicity we refer to the mean values as the *node speed*. We show the number of finished nodes over time for Rand4DColoring, CRRCorrecting, MWCorrecting, and YuCorrecting for node speed values of 0, 0.1, 0.5, and 1 in Figure 5.11.



(a) No mobility

(b) Node speed 0.1

(c) Node speed 0.5

(d) Node speed 1.0

Figure 5.11: The progress of the coloring for the different algorithms. We depict the average number of finished nodes over time for different nodes speed values drawn according to a Gaussian distribution with mean values between 0 and 1.

Some algorithms can clearly cope with mobility better than others. Our algorithm Rand4DColoring, MWCorrecting, and YuCorrecting maintain a higher fraction of validly colored nodes than CRRCorrecting.

There are three main reasons for the poor performance of CRRCorrecting. First, nodes that selected a leader to request an active interval from this leader keep the selected leader throughout the simulation. Due to mobility such a final selection is not helpful as the node may be separated from the leader it selected before receiving the active interval. Second, the algorithm uses only $\Delta + 1$ colors, thus the the probability of a conflict is higher than in Rand4DColoring and MWColoring. The third and major reason is that between detecting the conflict and selecting a new color a relatively long time passes, compared to the other algorithms. Nodes that detected a conflict compute the next valid active interval based on the previous active interval and the schedule length, and reset to waiting for this next active interval. Due to mobility, however, this does not guarantee that few nodes are active in each interval. Instead it mainly leads to many nodes waiting to compete for a color, while the nodes directly start to compete for a color in the other algorithms. Thus, while the schedule helps coordinating the color selection scheme in the static setting, this is not the case for the dynamic setting.

The second $(\Delta+1)$-coloring algorithm, YuCorrecting[3], maintains a significantly higher fraction of valid colors. Non-leader nodes may get blocked and separated from the blocking node, which could also lead to a low performance. However, the algorithm prevents such a case. Even if a node is blocked by one leader it may receive the StartColoring message by another leader and continue to request a color by this other leader. Another source of error in the dynamic setting is, as in CRRCorrecting, the request state, in which the nodes depend on the one leader they selected. However, less nodes are affected, as the nodes are allowed to leave the request state if they get blocked by another node. Finally, the algorithm may benefit from the smaller deployment area more than the other algorithms due to its increased transmission range, however, we expect this to be not significant. Overall, the algorithm achieves a solid performance in the mobile setting.

MWColoring performs even a little better than YuColoring, which is probably mainly due to the increased number of colors and thereby the lower probability for a conflict. As before, some nodes may get stuck in the request state, as dominated nodes select one leader and keep trying to contact this leader, however, resetting the nodes to their first color competition state leads to significantly less time required to re-color the nodes than in CRRCorrecting.

The best performance in the mobile setting is achieved by Rand4DColoring, which maintains a very high fraction of validly colored nodes throughout the execution. If a node moves closer to another node with the same color and thereby a conflict is introduced, one of the nodes simply selects a new color. Thus, a very high percentage of validly colored nodes is maintained. Even for a node speed of 1.0 less than 4 % of the nodes have an invalid color. This performance can be achieved as no structure needs to be build or maintained and detected conflicts are treated immediately by selecting a new random color.

We conclude from this experiment that it is quite complex for node coloring algorithms to maintain a valid coloring. Especially, the more complex the algorithms and the structures required to build the coloring, the worse they performed in the dynamic setting. This is due to the overhead required for building the structures, as well as the potential faults at different states of the algorithms. Our simple Rand4DColoring coloring algorithm performed best and maintained a valid coloring at almost all nodes even for relatively high node speeds.

5.3.8 *Highly Asynchronous Wake-up*

In this section we examine the robustness of the algorithm with respect to some nodes in the network waking-up later than large parts of the network. We consider how many of the 500 already colored nodes are disturbed by another 100 to 500 nodes waking up in the network and executing the algorithm. We use the random deployment on an area of 1000 m $\times$ 1000 m as in most parts of this chapter. We con-

3 In this setting we do not select color 0 for blocked nodes, as nodes may leave the blocked state. As this does not happen without mobility, there is a small fraction of nodes not finishing the algorithm in Figure 5.11a

sider the algorithms Rand4DColoring, Rand4DRespectColoring and the correcting variants CRRCorrecting and MWCorrecting and measure the number of already colored nodes that detect a conflict. We do not consider YuCorrecting, as YuColoring does explicitly not support the setting of nodes waking up in late stages of the algorithm (which also resulted in worse results in preliminary experiments). The results of the experiment are shown in Figure 5.12 and Table 5.10. Note that the basic algorithms ColorReduction and MWColoring are expected to produce no or a lot less disturbance as they support late wake-up of nodes and the reliable communication ensures that the nodes are aware of colors selected by neighbors. However, we consider the performance of the variants we optimized towards achieving a high performance regarding the runtime and the number of conflicts, thus, already colored nodes may be disturbed as we do not have reliable local broadcasting.

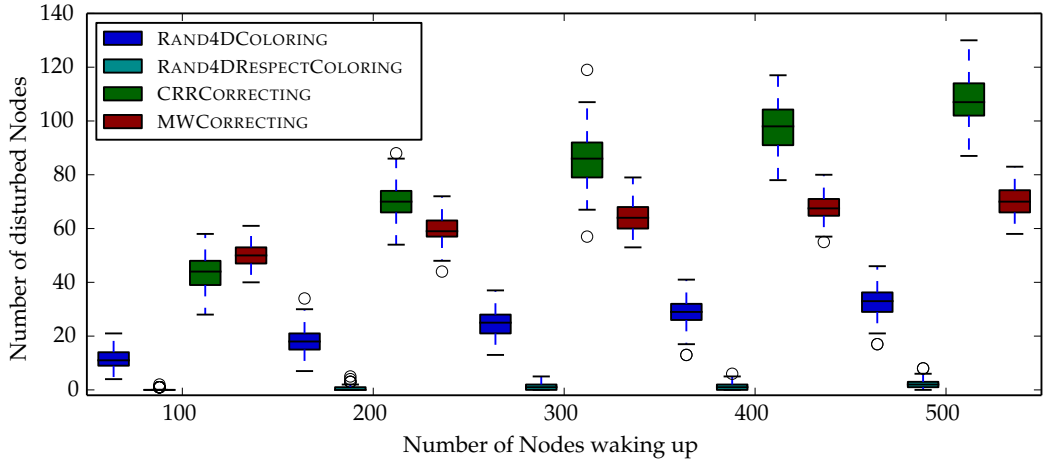


Figure 5.12: After 500 nodes finished and agreed on a valid coloring a varying number of additional nodes wake up (x-axis). We count the number of previously finished nodes that are disturbed, i.e. they detected a conflict after the additional nodes woke up.

Table 5.10: The average number of disturbed nodes for a varying number of nodes waking up after the 500 pre-deployed nodes selected a valid color.

Number of nodes waking up late	100	200	300	400	500
Rand4DColoring	11.24	18.05	24.97	28.51	32.39
Rand4DRespectColoring	0.14	0.72	1.12	1.54	2.34
CRRCorrecting	43.79	69.84	85.89	98.05	107.54
MWCorrecting	50.12	59.89	64.55	67.93	70.15

For RAND4DCOLORING the results indicate that even if additional 500 nodes wake up, only 32 already colored nodes detect a conflict. If we reduce the number of nodes starting that late, the number of disturbed nodes decreases further to 10.9, which corresponds to about 2 % of the pre-colored nodes. Thus, although theoretical considerations in Section 4.3 indicate that late wake-up of few nodes may introduce many conflicts we do not observe this here. The RAND4DRESPECTCOLORING variant of the algorithm, which additionally requires the nodes to listen for DURATION time slots before transmitting color messages, achieves essentially optimal results with only 0.2 to 2.3 disturbed nodes on average. The remaining correcting variants perform a lot worse, with roughly between 40 and 180 disturbed nodes. However, adding an additional listening phase and preventing the nodes from selecting colors taken by neighbors should significantly decrease the number of disturbed nodes, similar to how RAND4DRESPECTCOLORING improved upon RAND4DCOLORING.

5.4 CONCLUSION

In this chapter we experimentally evaluated several distributed node coloring algorithms designed for the SINR model. Apart from the algorithms RAND4DCOLORING and COLORREDUCTION, which we proposed in Chapter 4, we considered their competitors MWCOLORING and YUCOLORING, and variants of all algorithms. We used the network simulator Sinalgo [33] to study the runtime and the number of conflicts in the computed colorings on several deployment scenarios.

We conclude that our simple (4Δ)-coloring algorithm RAND4DCOLORING is very fast, as it requires significantly less time slots to compute a valid coloring than MWCOLORING or its variant MWCORRECTING. The coloring algorithm is even able to validly color the network in less time slots than required to finish one round of local broadcasting. RAND4DCOLORING can be made robust against highly asynchronous wake-up by adding a listening phase (we considered this variant as RAND4DRESPECTCOLORING). Additionally RAND4DCOLORING achieves the highest fraction of valid colors among the considered algorithms if nodes are mobile.

Regarding COLORREDUCTION we observe that the algorithm is significantly faster than our variant[4] of YUCOLORING while also introducing slightly less conflicts. Additionally, we studied a variant CRRANDCOLOR of COLORREDUCTION, which replaced the valid color by a random color without resulting in a significant difference in the algorithms performance. The fastest $(\Delta + 1)$-coloring algorithm in our experiments was RAND1DCOLORING, a variant of RAND4DCOLORING. Although RAND1DCOLORING does not finalize its color, which gives a slight advantage, this is surprising and hints the effectiveness of this simple family of algorithms.

The correcting variants are a combination of the ideas in RAND4DCOLORING and the remaining coloring algorithms, as the communication becomes unreliable and occurring conflicts are corrected. The variants decrease both the runtime and the

4 Originally, the coloring algorithm by Yu et al. [136] assumes a different setting, however, its ideas can be transferred to our setting, cf. Section 5.2.4.

number of conflicts significantly, however, without reaching the performance of RAND4DCOLORING or RAND1DCOLORING.

Overall the best performance for $\mathcal{O}(\Delta)$-colorings is achieved by RAND4DCOLORING and for $(\Delta + 1)$-colorings by RAND1DCOLORING and CRRCORRECTING, depending on the setting.

We conclude this chapter with open problems. Can the correcting heuristics CRRCORRECTING, MWCORRECTING, and YUCORRECTING be improved to achieve a better performance in the dynamic setting? Additionally, to the best of our knowledge this is the first work considering the experimental evaluation of distributed node coloring algorithms in the setting of wireless communication. It is interesting to see how the algorithms perform in a real-world sensor network.

Part II

TOWARDS CONGEST ALGORITHMS

6

SIMULATING CONGEST ALGORITHMS IN THE SINR MODEL

In this chapter we consider the distributed construction of a deterministic local broadcasting schedule in the SINR model of interference. During the execution of such a schedule each node should be able to transmit one message to its neighbors successfully. This allows the execution of higher-level algorithms in the SINR model for wireless networks. Our schedule can be computed in only $\mathcal{O}(\Delta \log n)$ time slots and we prove that the length of the constructed schedule is asymptotically optimal, i.e. of length $\mathcal{O}(\Delta)$. Considering the simulation of $\mathcal{CONGEST}$ algorithms in the SINR model, our deterministic schedule achieves a runtime of $\mathcal{O}(\tau\Delta + \Delta \log n)$ time slots, where τ is the original runtime in the (vertex-)$\mathcal{CONGEST}$ model. As a lower bound of $\Omega(\Delta)$ time slots per round exists, this is optimal apart from the logarithmic factor. For the edge-$\mathcal{CONGEST}$ model we show a lower bound of $\Omega(\Delta^2)$ per round, which we essentially match with a simulation of edge-$\mathcal{CONGEST}$ algorithms with runtime τ in $\mathcal{O}(\tau\Delta^2 + \Delta \log n)$ time slots.

This chapter is based on joint work with Dorothea Wagner. Preliminary results are published in [47].

6.1 INTRODUCTION

Local broadcasting is one of the most fundamental task in wireless ad hoc and sensor networks after deployment. In contrast to global broadcasting, where one message must be spread over the whole network, in the problem of local broadcasting each node must transmit one message only to all direct neighbors. In wireless networks usually only a fraction of all nodes can transmit simultaneously due to the signal interference of multiple transmissions. Hence, local broadcasts must be coordinated in order to avoid too high interference. Since interference is modeled relatively realistic in the SINR model (Signal-to-Interference-and-Noise-Ratio model, cf. Section 2.2.3), finding a local broadcasting schedule is non-trivial in this model. For many other models, such as the message-passing based $\mathcal{CONGEST}$ or $\mathcal{LOCAL}$ models, the problem of computing a local broadcasting schedule does not occur as interference-free communication is assumed (cf. Section 2.2.1) and message reception is guaranteed regardless of other transmissions.

As wireless technology is becoming more and more ubiquitous, distributed computing in a wireless context—along with the SINR model—received increasing attention in recent research. Local broadcasting is a fundamental problem in the

SINR model that can be used as a building block to solve higher-level problems. Hence it is quite well studied and can be solved in $\mathcal{O}(\Delta \log n)$ time slots [55] if Δ is known (other results are discussed in Section 6.1). Due to the vast amount of algorithms designed for message-passing models, one particularly interesting application of local broadcasting is to simulate algorithms designed for message-passing models in the SINR model.

For complex algorithms it may be more effective to invest some time in a preprocessing step in order to achieve faster local broadcasting. In fact, this can be beneficial and Derbel and Talbi [30], Jurdzinski and Kowalski [78], and Barenboim and Peleg [16] achieve—using different methods and assumptions—local broadcasting in $\mathcal{O}(\Delta)$ time slots, which is optimal due to a trivial lower bound (cf. Section 6.4). Derbel and Talbi use as increased transmission power to coordinate the schedule and require $\mathcal{O}(\Delta \log n)$ time slots for their preprocessing. Jurdzinski and Kowalski assume position information to be given and require $\mathcal{O}(\Delta \log^3 n)$ slots, while Barenboim and Peleg use so-called Steiner-nodes, which are additionally deployed in the network, and assume free feedback of the nodes to compute the schedule in $\mathcal{O}(\log^* n(\Delta + \log n \log\log n))$ time slots. Inspired by the approaches of Derbel and Talbi, and Jurdzinski and Kowalski we describe how to construct a deterministic local broadcasting schedule with optimal length $\mathcal{O}(\Delta)$ and preprocessing time of $\mathcal{O}(\Delta \log n)$ time slots. We use distributed node coloring to construct an infeasible local broadcasting schedule and combine it with the concept of dilution by Jurdzinski and Kowalski, which enables us to achieve feasibility of the schedule while increasing the length of the schedule by only a constant multiplicative factor. We require the nodes to know an upper bound on the number of nodes n, the maximum node degree Δ in the network, their own ID, and location information. We do not require carrier sensing and restrict ourselves to uniform and non-adjustable transmission powers.

Our algorithm differs from the previously mentioned algorithms in various ways. In contrast to the distributed node coloring by Derbel and Talbi we do not require the nodes to tune their transmission power. With regard to the backbone structure constructed by Jurdzinski and Kowalski the method described in this chapter is faster by a polylogarithmic factor. Compared to the approach of Barenboim and Peleg we neither require feedback nor additional Steiner-nodes.

Related Work

Using local broadcasting to emulate a round-based message-passing environment even complex distributed algorithms designed for the $\mathcal{CONGEST}$ model can be made available in wireless sensor networks. Relieved from the complicated analysis in the SINR model, such algorithms can tackle other complex problems, such as for example all-pairs shortest paths [72], graph partition [29], or the backbone construction algorithm in Chapter 7.

The simulation of message-passing algorithms in radio networks (in which a message is successfully received if the receiver is silent and only one of its neighbors is transmitting) has first been studied by Alon et al. in [4]. They propose a

separate simulation of each round of the message-passing algorithm. Among other results they proved a bound of $\Theta(\Delta^2)$ for the case that each node transmits a different message to each of its neighbors. The lower bound translates to the SINR model with a slightly modified proof (see Section 6.4.1), while the upper bound has not yet been reached (if we account for the preprocessing). Kuhn et al. [91] proposed an abstract interface—an abstract MAC layer—that enables abstract models to be executed in more realistic models for wireless communication. However, they describe an implementation of the abstract MAC layer by local broadcasting in the radio network model, which does not account for global interference. Recently such an abstract MAC layer for the SINR model has been proposed by Halldórsson, Holzer and Lynch [63]. They focus on global problems such as broadcast or global consensus and provide two important and general bounds. The first is f_{ack}, which bounds the time required to finish and acknowledge a broadcast. The second is f_{approg}, which intuitively bounds the time required for a node to receive a "current" message if at least one of its neighbors is transmitting. Compared to this work, their work allows to solve global problems much faster, however, it does not provide an efficient local broadcast schedule that would, for example, enable the nodes to directly execute $\mathcal{CONGEST}$ algorithms.

Local broadcasting in the SINR model has first been studied by Goussevskaia et al. in [55]. They considered local broadcasting with known and unknown competition (which is the number of nodes within the proximity range around the node) in asynchronous networks, and propose two randomized algorithms for the SINR model with runtimes of $\mathcal{O}(\Delta \log n)$ and $\mathcal{O}(\Delta \log^3 n)$ for known and unknown competition, respectively. Yu et al. [137] improve the approximation ratio for the unknown competition by a logarithmic factor to $\mathcal{O}(\Delta \log^2 n)$ and propose two algorithms for the synchronized model (with synchronous and asynchronous wake-up) that make use of carrier sensing and thereby achieve local broadcasting in $\mathcal{O}(\Delta \log n)$ time slots. In [133] Yu et al. improve the algorithm for asynchronous time slots and unknown competition further to $\mathcal{O}(\Delta \log n + \log^2 n)$ and provide a lower bound of $\Omega(\Delta + \log n)$ for randomized algorithms in this model. Halldórsson and Mitra [65] provide an algorithm with the same running time of $\mathcal{O}(\Delta \log n + \log^2 n)$ in the same model, that is slightly simpler and more robust towards node failure. They also provide an algorithm that achieves a running time of $\mathcal{O}(\Delta + \log^2 n)$ per round of local broadcasting with the assumption that acknowledgments are received at no cost. Barenboim and Peleg recently improved this result to $\mathcal{O}(\Delta + \log n \log\log n)$ [16].

The first result in the SINR model that achieves local broadcasting in $\mathcal{O}(\Delta)$ time slots after a preprocessing stage of $\mathcal{O}(\Delta \log n)$ slots is by Derbel and Talbi [30]. They transfer a distributed node coloring algorithm proposed by Moscibroda and Wattenhofer [104] to the SINR model and, by tuning the transmission power during the coloring step, achieve a deterministic local broadcasting schedule of length $\mathcal{O}(\Delta)$ that is feasible in the SINR model. Jurdzinski and Kowalski [78] assume the location to be known to the nodes and achieve the optimal runtime of $\mathcal{O}(\Delta)$ for local broadcasting without requiring the capability of nodes to tune their trans-

Table 6.1: Local broadcasting results for the SINR model. Ordered chronologically by appearance with separation in algorithms with and without preprocessing; (abbreviations: (a)sync: (a)synchronous model, cs: carrier sense, ACKs: free acknowledgements, tpt: transmission power tuning, steiner: additional Steiner nodes, S: schedule length, P: preprocessing time.)

Publication	Assumptions	Runtime
Goussevskaia et al. [55]	async, Δ	$\mathcal{O}(\Delta \log n)$
Goussevskaia et al. [55]	async	$\mathcal{O}(\Delta \log^3 n)$
Yu et al. [137]	async	$\mathcal{O}(\Delta \log^2 n)$
Yu et al. [137]	sync, cs	$\mathcal{O}(\Delta \log n)$
Yu et al. [133]	async	$\mathcal{O}(\Delta \log n + \log^2 n)$
Halldórsson & Mitra [65]	async, ACKs	$\mathcal{O}(\Delta + \log^2 n)$
Barenboim & Peleg [16]	async, ACKs	$\mathcal{O}(\Delta + \log n \log\log n)$
Derbel & Talbi [30]	sync, Δ, tpt	S: $\mathcal{O}(\Delta)$, P: $\mathcal{O}(\Delta \log n)$
Jurdzinski & Kowalski [78]	sync, Δ, location, deterministic	S: $\mathcal{O}(\Delta)$, P: $\mathcal{O}(\Delta \log^3 n)$
Barenboim & Peleg [16]	sync, ACKs, steiner	S: $\mathcal{O}(\Delta)$, P: $\mathcal{O}(\log^* n(\Delta + \log n \log\log n))$
This work	sync, Δ, location	S: $\mathcal{O}(\Delta)$, P: $\mathcal{O}(\Delta \log n)$

mission power. Their algorithms is deterministic and the preprocessing stage requires $\mathcal{O}(\Delta \log^3 n)$ time slots. The authors introduce the concept of dilution (cf. Section 6.2) and build a deterministic backbone structure that enables communication to the backbone in $\mathcal{O}(\Delta)$ and local broadcasts from within the backbone in constant time. Using additional Steiner nodes Barenboim and Peleg construct a local broadcasting schedule in $\mathcal{O}(\log^* n(\Delta + \log n \log\log n))$ [16]. We give an overview on results regarding local broadcasting in Table 6.1.

Contribution

Our main contribution is the construction of a local broadcasting schedule that achieves local broadcasting in asymptotically optimal time $\mathcal{O}(\Delta)$ using a preprocessing of only $\mathcal{O}(\Delta \log n)$ time slots. To achieve this we use a standard node coloring as basis of the schedule, which is usually not feasible. To achieve feasibility in the SINR model we use the concept of dilution introduced by Jurdzinski and Kowalski [78].

Our local broadcasting schedule enables us to simulate algorithms (with original runtime τ) designed for the $\mathcal{CONGEST}$ and the edge-$\mathcal{CONGEST}$ model in the SINR model. We achieve a runtime of $\mathcal{O}(\tau\Delta + \Delta \log n)$ and $\mathcal{O}(\tau\Delta^2 + \Delta \log n)$ time slots in the SINR model, respectively.

Finally, we show that both results are optimal up to the logarithmic factor by showing a lower bound of $\Omega(\Delta^2)$ to simulate one round of edge-$\mathcal{CONGEST}$. A lower bound of $\Omega(\Delta)$ for a round of $\mathcal{CONGEST}$ is already known.

Outline

The remainder of this chapter is structured as follows. In the next section, we describe required models and state some basic definitions. In Section 6.3, the construction of the deterministic local broadcasting schedule is described and we show its feasibility in the SINR model. Afterwards we consider the simulation of $\mathcal{CONGEST}$ algorithms in the SINR model in Section 6.4 and prove our lower bound. We conclude this chapter with some final remarks in Section 6.5.

6.2 MODELS AND PRELIMINARIES

Let us first recall the essential notation used for the geometric SINR model of interference. In this model a transmission from a node v to a node u is *successful*, or *feasible*, iff the SINR constraint holds. This constraint is based on the uniform transmission power P, the Euclidean distance $\text{dist}(\cdot,\cdot)$, the set of simultaneous transmissions I, the attenuation parameter $\alpha \in [2,6]$, a hardware constant $\beta \geq 1$ and the environmental noise $N > 0$, and evaluates to $\frac{\frac{P}{\text{dist}(u,v)^\alpha}}{\sum_{w\in I}\frac{P}{\text{dist}(w,v)^\alpha}+\text{N}} \geq \beta$. The *broadcasting range* is based on the constraint and denoted by r^B. The nodes closer to v than r^B are its neighbors, the degree of v the number of its neighbors, and Δ is the maximum degree of any node in the network. As we do not require more specifics about the communication in the SINR model, we refer to Section 2.2.3, where we introduce the SINR model more thoroughly.

For the simulation of higher-level algorithms in the SINR model we require synchronous execution of the computed local broadcasting schedule. We do not require synchronous execution for the computation of the schedule. Many current positioning systems, such as GPS, position information can be used for clock synchronization [84]. For other synchronization options we refer to [122].

Simulating $\mathcal{CONGEST}$ Algorithms in the SINR Model

We introduced the $\mathcal{CONGEST}$ model of distributed computation in Section 2.2.1. Recall that algorithms designed for this model use a messages of size at most $\mathcal{O}(\log n)$, which enables the transmission of at most a constant number of node IDs in the range $[0,\dots,n]$. The $\mathcal{CONGEST}$ model is sometimes also denoted as vertex-$\mathcal{CONGEST}$, since the congestion is realized at the vertices of the communication graph. We refer to the corresponding model, in which congestion is realized at the edges, as edge-$\mathcal{CONGEST}$. For a simulation of algorithms designed for the $\mathcal{CONGEST}$ model of distributed computation in the geometric SINR model we require the following properties to hold:

- **Locality:** The neighbors of each node v must be reachable in our model, i.e. within the nodes broadcasting range r^B.
- **Synchronization:** Two neighbors are not allowed to be in different rounds of the vertex-/edge-$\mathcal{CONGEST}$ algorithm.

For the simulation of each round of the vertex-$\mathcal{CONGEST}$ (edge-$\mathcal{CONGEST}$) algorithm to be successful we require that one (Δ) transmission(s) per node must be feasible in the SINR model of interference w.h.p. We assume the network to be connected, hence synchronization in combination with connectivity already implies that all nodes must be in the same round of the $\mathcal{CONGEST}$ algorithm.

Dilution and Backbone Structure

pivotal grid

In accordance with [78] we call a partition of the 2-dimensional plane in boxes of size $\gamma \times \gamma$ the *pivotal grid* G_γ, where $\gamma = r^T/\sqrt{2}$. Note that the dimensions of the box are such that all nodes within the same box are within each others transmission radius. Formally each box includes its bottom and left side but does not include its top and right side. We assume *box* $C(i,j)$ to be the box with lower left coordinates $(i,j) \in \mathbb{R}^2$. A node with position (x,y) is in box $C(i,j)$ iff $\lfloor \frac{x}{\gamma} \rfloor = i$ and $\lfloor \frac{y}{\gamma} \rfloor = j$.

box $C(i,j)$

local broadcasting schedule

A *local broadcasting schedule* can be seen as an assignment of 0/1-bit-strings to nodes indicating in which time slots the node is allowed to broadcast. In the deterministic schedule constructed in this chapter, however, each node transmits only once throughout an execution of the schedule. Hence we can simply store the number of the time slot instead of a 0/1 bit-string[1].

dilution

In order to combine geometric information with local broadcast schedules, we use the concept of *dilution* as introduced in [78]. For a constant δ, which determines the distance between two active transmissions and will be defined later, we assign each node v local coordinates $(l_x^v, l_y^v) = (\lfloor \frac{x}{\gamma} \rfloor \mod \delta, \lfloor \frac{y}{\gamma} \rfloor \mod \delta) = (i \mod \delta, j \mod \delta)$. This ensures that nodes in the same box of G_γ share the same local coordinates. Now, we can *dilute* a local broadcast schedule by a factor of δ^2 by allowing each node v with local coordinates (l_x^v, l_y^v) to send in time slot $t\delta^2 + l_x^v\delta + l_y^v$ iff v was allowed to send in time slot t in the original schedule.

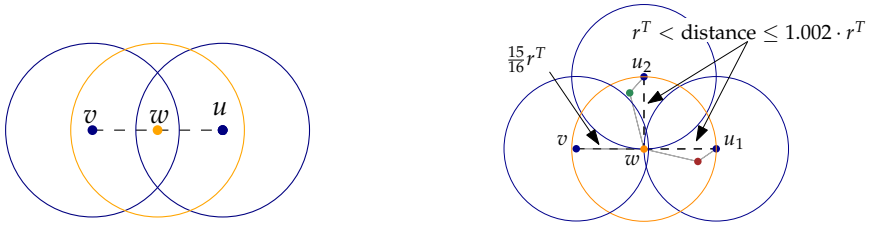
6.3 DETERMINISTIC LOCAL BROADCASTING SCHEDULE

An often considered approach to establish efficient communication in a wireless network is to compute a graph coloring and use this coloring to decide when and for how long each node is allowed to transmit a message. This can be done by simply associating each color with a time slot, and thereby creating a TDMA schedule. As a first step, let us consider the simpler protocol model, in which a transmission is successful iff in the interference range (which usually is at least the transmission

1 A variant of such a schedule has already been used for the asynchronous color reduction algorithm in Section 4.5. There each node received an interval of time slots in which to be active.

range) of the receiver only one node is transmitting at a given time. Even in this simpler model a node coloring that ensures that two nodes are assigned different colors if they are within each others transmission range is not sufficient to directly build a feasible transmission schedule as depicted in Figure 6.1. However, if the transmission range equals the interference range this can be overcome by using a distance-2-coloring (i.e., a coloring which ensures unique colors within each transmission region).

Due to the global nature of interference in the SINR model, finding some sort of agreement about transmission schedules (i.e., medium access) is required for deterministic local broadcasting schedules. In the case of coloring in the SINR model, even a distance-2-coloring that achieves unique colors within each transmission region is not sufficient as shown in Figure 6.1b.



(a) Unique colors within distance r^T

(b) Unique colors within each transmission region (distance-2-coloring)

Figure 6.1: A coloring as depicted on the left does not yield a feasible local broadcasting schedule in the protocol model as simultaneous transmissions from v and u to w are not feasible. Although the coloring on the right corresponds to a feasible local broadcasting schedule in the protocol model, it is not feasible in the SINR model as the SINR constraint is violated. Note that w is not directly connected to u_1 and u_2 but through the red/green node.

However, schedules can be made feasible if the node coloring ensures unique colors in an area larger than the transmission region. Unfortunately coordinating such a coloring without communicating with nodes outside the transmission region seems hard. In current solutions this is realized by tuning the nodes transmission power to reach a larger transmission region, as done in [30], by adding additional so-called Steiner-nodes, which achieve short communication paths to other close-by nodes [16] or using position information [78]. Yet another option would be to traverse through the network to coordinate the schedule, however, this requires $\Omega(D)$ time slots, where D is the diameter of the network, cf. Section 2.1. We use position information to compute a grid structure which helps in coordinating the local broadcasting schedule. In the following theorem we show that we can distributively construct a feasible local broadcasting schedule based on the location information and a given node coloring, even if the coloring does not ensure unique colors within each transmission region (i.e. it is a standard distance-1-coloring). Recall that such a coloring can be computed within $\mathcal{O}(\Delta \log n)$ time slots in the SINR model, as presented in Chapter 4. If not noted otherwise we assume such a coloring.

Theorem 6.1. *Given a network of nodes in which each node knows its location, the color assigned by a coloring using at most $c_{max} = \mathcal{O}(\Delta)$ colors, and c_{max} itself. Then we can distributively compute a local broadcasting schedule that is feasible under the SINR model of interference with length in $\mathcal{O}(\Delta)$.*

In order to prove the theorem we first show that such a coloring is a local broadcasting schedule in which at most one node sends in each box of the pivotal grid G_γ (Lemma 6.2), and then prove that we can achieve a feasible schedule by applying dilution to this schedule (Lemma 6.3).

Lemma 6.2. *Given a network in which each node has a unique color within distance r^T. This coloring corresponds to a local broadcasting schedule in which at most one node per box of the pivotal grid G_γ is transmitting in each slot.*

Proof. As each node knows the number c of its color and a shared upper bound $c_{\max}$ on the number of colors assigned to the nodes in the network we can assign each color to one of $c_{\max}$ time slot. Consider a node v within box $C(i,j)$ and color c. Since the diameter of each box is exactly r^T, the coloring ensures that there is no other node within box $C(i,j)$ that has color c. □

We extend Proposition 1 of [78] in the following by explicitly giving a formula to compute the constant δ (depending only on α), which enables us to prove feasibility of a δ-diluted schedule in the SINR model of interference for $\alpha > 2$. For $\alpha = 2$ we do also achieve feasibility, however in this case $\delta \in \mathcal{O}(\log n)$ is additionally dependent on n. This leads to an increase in the schedule length of a multiplicative factor of $\delta^2 \in \mathcal{O}(\log^2 n)$. We consider this case after proving Lemma 6.3.

Lemma 6.3. *Let $\alpha > 2$ and $\delta = \left(\frac{8P\sum_{k=1}^{\infty}\frac{1}{k^{\alpha-1}}}{N\gamma^\alpha}\right)^{1/\alpha} + 3$. Then a local broadcasting schedule $\mathcal{S}$ in which at most one node in each box of the pivotal grid G_γ transmits in each time slot can be made feasible in the SINR model of interference with a constant increase in the schedule length.*

Proof, partially based the proof of Proposition 1 in [78]. Let $\text{len}(\mathcal{S})$ be the length of the local broadcasting schedule $\mathcal{S}$. In order to achieve a feasible schedule, we dilute the schedule $\mathcal{S}$ by a constant δ^2 and obtain a feasible schedule $\mathcal{S}'$ with $\text{len}(\mathcal{S}') = \mathcal{O}(\text{len}(\mathcal{S}) \cdot \delta^2) = \mathcal{O}(\text{len}(\mathcal{S}))$. In this schedule $\mathcal{S}'$, a node v with local coordinates (l_x^v, l_y^v) sends in time slot $t\delta^2 + l_x^v\delta + l_y^v$ if and only if the node would have sent in time slot t of schedule $\mathcal{S}$.

Let us begin by showing that δ is indeed a constant. Clearly, this holds for the influence of the constants P, N, γ. The sum $\sum_{k=1}^{\infty}\frac{1}{k^{\alpha-1}}$, which is the generalized harmonic number of order $(\alpha-1)$, is in $\mathcal{O}(1)$ for any $\alpha > 2$ [89]. Let us now consider an arbitrary time slot of schedule $\mathcal{S}'$, a node v that transmits a message in this time slot, and another node w that is within the broadcasting region of v. Let $C(i,j)$ be the box in which v is located, hence $(l_x^v, l_y^v) = (i \mod \delta, j \mod \delta)$ are the local coordinates of v. We claim that w can successfully receive the message sent by v and hence—as we considered an arbitrary sender, receiver and time slot—this

schedule is feasible in the SINR model. To show this claim we bound the interference received by w simultaneously transmitting nodes by first upper bounding the number of simultaneously transmitting nodes within certain distances and then computing an upper bound on the interference of all those nodes on w.

The application of δ-dilution ensures that the only nodes u that transmit simultaneously with v have local coordinates $(l_x^u, l_y^u) = (i \mod \delta, j \mod \delta) = (l_x^v, l_y^v)$. Note that local coordinates are shared by all nodes in the same box. Hence we call boxes that have nodes with the same local coordinates as v, i.e. boxes that are also allowed to send in the considered time slot, *active*. Due to the cyclicity of the modulo operator, δ-dilution results in a grid of active boxes with distance $\xi := (\delta - 1)\gamma$ between each two active boxes, as depicted in Figure 6.2. According to Lemma 6.2 at most one node in each active box transmits in each time slot.

active

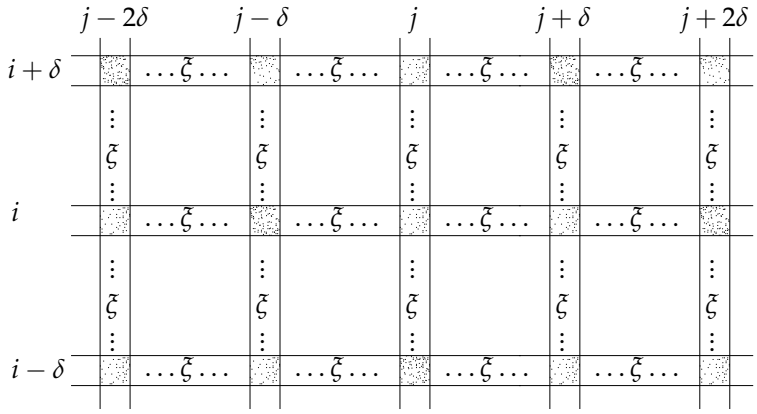


Figure 6.2: Grid cells of G_γ that are active simultaneously to a transmission originating from box $C(i,j)$. Note that in order to increase readability $\xi := (\delta - 1)\gamma$.

Let us now examine how many active boxes are at specified distances. We consider the boxes in so-called rings, which actually are the border layer of active boxes of a square centered at the box $C(i,j)$. In the situation of Figure 6.2 all depicted nodes in columns $j-\delta$, j and $j+\delta$ except for $C(i,j)$ itself are in boxes of ring level 1 from $C(i,j)$. It can be observed that in each ring of level $k \geq 1$, exactly $8k$ active boxes can be accommodated. Also, each node in level k has distance at least $k((\delta-3)\gamma)$ from w ($\delta - 3$ since w can be at most 2 boxes away from v).

Using this relation we can now upper bound the interference received by w from all nodes sending simultaneously with v, which are at most $8k$ nodes from each ring level k. Hence the interference at w is at most

$$\sum_{\substack{u \in V\setminus\{v\},\\ u \text{ sending simultaneously with } v}} \frac{P}{\mathrm{dist}(u,w)^\alpha} \leq \sum_{k=1}^{\infty} (8k) \frac{P}{(k(\delta-3)\gamma)^\alpha} \tag{6.1}$$

$$\leq \sum_{k=1}^{\infty} \frac{8Pk}{k^\alpha(\delta-3)^\alpha\gamma^\alpha} \leq \frac{8P}{(\delta-3)^\alpha\gamma^\alpha}\left(\sum_{k=1}^{\infty}\frac{1}{k^{\alpha-1}}\right) \leq N \tag{6.2}$$

where the first equation follows from applying the considerations about the ring levels and the last equation follows by insertion of δ, which we already showed to be constant. Evaluating the SINR at node w yields

$$\mathrm{SINR}_w = \frac{\frac{P}{\mathrm{dist}(v,w)^\alpha}}{\sum_{\substack{u \in V \setminus \{v\}, \\ u \text{ sending simultaneously with } v}} \frac{P}{\mathrm{dist}(u,w)^\alpha} + N} \geq \frac{\frac{P}{(r^B)^\alpha}}{2N} \geq \beta$$

where the first inequality follows from $\mathrm{dist}(v,w) \leq r^T$ and Equation 6.2 and the last inequality follows from the definition of the broadcasting range $r^B = (\frac{P}{2N\beta})^{1/\alpha}$. This concludes the proof for $\alpha > 2$. □

We will now briefly consider the case of $\alpha = 2$.

Corollary 6.4. *Let $\alpha = 2$ and $\delta = \left(\frac{8P\sum_{k=1}^{n}\frac{1}{k^{\alpha-1}}}{N\gamma^\alpha}\right)^{1/\alpha} + 3$. Then a local broadcasting schedule $\mathcal{S}$ in which at most one node in each box of the pivotal grid G_γ transmits in each time slot can be made feasible in the SINR model of interference with a factor $\delta^2 \in \mathcal{O}(\log^2 n)$ increase in the schedule length.*

Proof. Since at most n non-empty ring levels exist and the distance of the levels increases it holds that

$$\sum_{\substack{\text{non-empty} \\ \text{ring levels } k}} \frac{1}{k^{\alpha-1}} \leq \sum_{k=1}^{n} \frac{1}{k^{\alpha-1}} \tag{6.3}$$

and hence the resulting sum $\sum_{k=1}^{n} \frac{1}{k}$ and Equation 6.1 in general can be evaluated to $\mathcal{O}(\log n)$ [89]. This implies $\delta \in \mathcal{O}(\log n)$ and finally $\mathrm{len}(\mathcal{S}') = \mathcal{O}(\mathrm{len}(\mathcal{S}) \cdot \delta^2) = \mathcal{O}(\mathrm{len}(\mathcal{S}) \cdot \log^2 n)$ as claimed in the corollary. □

A pseudo code description of the procedure described above is given in Algorithm 6.1. First an initial schedule is computed by distributed node coloring, then this schedule is diluted in order to obtain a schedule that is feasible in the SINR model. We can see that the algorithm itself is very simple. For a definition of the parameters cf. Section 6.2 or Section 2.2.2. Note that regarding δ neither the ceiling nor limiting the sum at n affects our theoretic results.

6.4 SIMULATING $\mathcal{CONGEST}$ ALGORITHMS IN SINR

Using the deterministic local broadcasting schedule constructed in Section 6.3, $\mathcal{CONGEST}$ algorithms with a runtime in $\mathcal{O}(\tau)$ can be simulated in $\mathcal{O}(\tau\Delta + \Delta \log n)$ for $\alpha > 2$. This can be done by first computing the local broadcasting schedule in $\mathcal{O}(\Delta \log n)$ and then simulating the algorithm using so-called single-round-simulation as introduced by Alon et al. [4]. This requires one execution of the local broadcasting schedule for each round of the message-passing algorithm. There is a simple and well-known lower bound of $\Omega(\Delta)$ for one round of local broadcasting. As only one transmission can be received in a time slot, Δ nodes in a broadcasting region require $\Omega(\Delta)$ time slots to transmit to one shared neighbor.

Algorithm 6.1: Distributed computation of a feasible local broadcasting schedule at node v

Input: location information (x^v, y^v); SINR parameters α, N, β, P; network parameters Δ, n

$c, c_{\max} \leftarrow$ color assigned by distributedNodeColoring(Δ, n, α, N, β, P)

$\delta \leftarrow \left\lceil \left(\frac{8\mathrm{P}\sum_{k=1}^{n} \frac{1}{k^{\alpha-1}}}{\mathrm{N}\gamma^{\alpha}} \right)^{1/\alpha} \right\rceil + 3$ // dilution constant

$(l_x^v, l_y^v) \leftarrow (\lfloor \frac{x^v}{\gamma} \rfloor \mod \delta, \lfloor \frac{y^v}{\gamma} \rfloor \mod \delta)$ // local coordinates

active_slot $\leftarrow \delta^2 c + \delta l_x^v + l_y^v$

while *true* **do** // Executing local broadcasting schedule

  slot $\leftarrow$slot+1

  if *slot - active_slot* $\equiv 0$ *(mod* $\delta^2 \cdot c_{max}$*)* **then** // $c_{\max}$ is max. color

    Transmit next message

We restricted ourselves to the simulation of general (vertex-)$\mathcal{CONGEST}$ algorithms in most parts of our chapter. In this model a node can send one message of size $\mathcal{O}(\log n)$ to its neighbors in each round, cf. Section 6.2. However the methods transfer to the simulation of algorithms designed for similar models, for example if a different message is transmitted to each neighbor or if differently-sized messages are used. In particular for messages of arbitrary size s in a message-passing algorithm, either the message size during simulation in the SINR model is set to $O(s + \log n)$, or the messages are split to $\mathcal{O}(\log n)$ sized messages, leading to an increase in runtime of $s / \log n$ if $s \in \Omega(\log n)$. If, as in the edge-$\mathcal{CONGEST}$ model, a different message is sent to each neighbor the runtime of the simulation increases to $\mathcal{O}(\tau\Delta^2 + \Delta \log n)$, as our local broadcasting schedule must be executed Δ times per round of the original edge-$\mathcal{CONGEST}$ algorithm. We shall show in the following that this is also tight up to logarithmic factor.

6.4.1 *Lower Bound for Edge-$\mathcal{CONGEST}$ Algorithms*

In this section we prove that simulating one round of an edge-$\mathcal{CONGEST}$ algorithm in the SINR model requires $\Omega(\Delta^2)$ time slots. Such a lower bound has already been stated by Alon et al. in [4] for the radio network model. As it does not directly transfer to the SINR model, we prove the lower bound in the following lemma.

Lemma 6.5. *Executing one round of an edge-$\mathcal{CONGEST}$ algorithm at each node cannot be simulated in less than $\Omega(\Delta^2)$ time slots in the SINR model, where Δ is the maximum node degree of all nodes in the network.*

Proof. Assume a graph in which all nodes are within one broadcasting radius r^B and let this graph consist of two clusters S_l, S_r of the same geometric diameter $d = 1$. Let those clusters be at least $\eta = 20$ times their diameter apart from each other. Such clusters are shown in Figure 6.3.

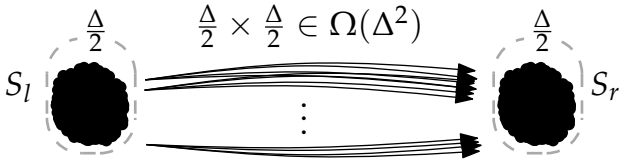


Figure 6.3: Two clusters of the same diameter within one broadcasting region. The distance between the clusters is more than η times the diameter of the cluster.

To execute one round of the edge-$\mathcal{CONGEST}$ algorithm at each of the nodes, each node in the network may transmit a message to each of its neighbors. Let us only consider the transmission from the left cluster to the right cluster. Each node in the left cluster must transmit one different message to each node in the right cluster. This yields $\frac{\Delta}{2} \times \frac{\Delta}{2} \in \Omega(\Delta^2)$ inter-cluster-transmissions.

We will now show that no more than two inter-cluster transmissions (from the left to the right cluster) can occur during one time slot in the SINR model. Let $v \in S_l$ be in the left cluster and $w \in S_r$ be in the right cluster. Assume v transmits to w in time slot t and assume two other nodes u, u' transmit to some other nodes in the same time slot. We show that w cannot successfully receive v's message due to a SINR of less than 1. Let $u, u' \in S_l$ transmit, then the SINR constraint (cf. Section 6.2) evaluates to

$$\frac{\frac{P}{\text{dist}(v,w)^\alpha}}{\frac{P}{\text{dist}(u,w)^\alpha} + \frac{P}{\text{dist}(u',w)^\alpha} + N} \leq \frac{\frac{P}{(\eta)^\alpha}}{2 \cdot \frac{P}{(\eta+2)^\alpha}} = \frac{\frac{1}{(\eta)^\alpha}}{\frac{2}{(\eta+2)^\alpha}} = \frac{(\eta+2)^\alpha}{2\eta^\alpha} \overset{\alpha \in [2,6], \eta = 20}{<} \beta,$$

where the first inequality holds as $\text{dist}(v,w) \geq \eta$, $\text{dist}(u,w) \leq (\eta + 2d)$, and $N > 0$, the strict inequality follows from the selection of α and η. Hence w cannot receive v's message and at most two transmissions from the left to the right cluster can happen in one time slot. This shows that $\frac{1}{2} \cdot \left(\frac{\Delta}{2} \times \frac{\Delta}{2}\right) \in \Omega(\Delta^2)$ time slots are needed to simulate one round of a message-passing algorithm. □

6.4.2 *On the Size of Messages*

Similar to the $\mathcal{CONGEST}$ model, the SINR model usually allows messages of size $\mathcal{O}(\log n)$. If larger messages are used, for example in algorithms for the $\mathcal{LOCAL}$ model of computation, this can be achieved by splitting the messages. In the local broadcasting schedule proposed by Derbel and Talbi (their algorithm is based on distributed node coloring with tuned transmission radius), the message size is described as $\mathcal{O}(s \cdot \log n)$, where s is the original message size. For the simulation of $\mathcal{CONGEST}$ algorithms this results in messages of size $\mathcal{O}(\log^2 n)$ instead of $\mathcal{O}(\log n)$. However, we argue that messages of size $\mathcal{O}(\log n)$ are possible in their schedule as well, and hence this additional logarithmic factor is not necessary. Their algorithm consists of two parts. In the first part a distributed node coloring is computed. For this only the node ID and the number of the color must be transmitted. Hence messages of size $\mathcal{O}(\log n)$ are sufficient. In the second part the actual simulation takes place. Therefore the original message of size s along with a node

ID (in case the sender/receiver node must be identified) must be transmitted. This requires messages of size $\mathcal{O}(s + \log n)$. For $\mathcal{CONGEST}$ algorithms this results in messages of size $\mathcal{O}(\log n)$, since $s \in \mathcal{O}(\log n)$. For the algorithm by Jurdzinski and Kowalski and the algorithm recently proposed by Barenboim and Peleg a message size of $\mathcal{O}(\log n)$ is also sufficient.

Hence, for all these algorithms, using either tuned transmission powers, additional Steiner nodes or location information messages of size $\mathcal{O}(\log n)$ are sufficient to simulate a $\mathcal{CONGEST}$ algorithms in the SINR model.

6.5 CONCLUSION

In this chapter we introduced a new algorithm to compute a deterministic local broadcasting schedule of optimal length $\mathcal{O}(\Delta)$ that is feasible in the SINR model of interference. The construction of the schedule requires $\mathcal{O}(\Delta \log n)$ time slots, which is optimal up to the logarithmic factor.

Our local broadcasting schedule enables the simulation of algorithms designed for the $\mathcal{CONGEST}$ model in the realistic SINR models of interference in wireless networks. An algorithm with original runtime of τ rounds in the $\mathcal{CONGEST}$ model can be simulated in in the SINR model, which is optimal apart from the logarithmic factor. For simulating algorithms designed for the edge-$\mathcal{CONGEST}$ model we require $\mathcal{O}(\tau\Delta^2 + \Delta \log n)$ time slots, which we argue to be tight up to the logarithmic factor as well.

Several open problems are immediate. Current techniques require to tune their transmission power, add Steiner-nodes or have position information. Can we compute an optimal local broadcasting schedule without additional knowledge? And if not, which other forms of knowledge empower nodes to compute such a schedule?

7

IMPROVED DISTRIBUTED CONSTRUCTION OF FCDS PACKINGS

In this chapter we consider the problem of creating a backbone structure that achieves high throughput in wireless networks in the $\mathcal{CONGEST}$ model. For networks with node connectivity k, Fractional Connected Dominating Set (FCDS) Packings can achieve a broadcast throughput of $\Theta(k/\log n)$ messages per round, which is optimal for routing-based broadcasting. FCDS were recently proposed by Censor-Hillel et al. [20, 21] and are a natural generalization of Connected Dominating Sets (CDS), allowing each node to participate fractionally in multiple FCDS. Thereby, $\Omega(k)$ co-existing transmission backbones are established, taking full advantage of the network connectivity. In this chapter we propose a modified distributed algorithm to compute FCDSs, which improves upon previous algorithms for $k\Delta \in o(\min\{\frac{n \log n}{k}, D, \sqrt{n \log n} \log^* n\} \log n)$,where Δ is the maximum node degree, D the diameter and n the number of nodes in the network. We achieve this by explicitly computing connections between tentative dominating sets.

This chapter is based on joint work with Matthias Wolf. Preliminary results are currently under submission [48] and in parts based on work done during Matthias Wolfs bachelor thesis [130].

7.1 INTRODUCTION

Wireless ad hoc and sensor networks are used to monitor the environment, industry processes and even large parts of infrastructure. In order to cope with the growing networks size and its demand for efficient communication throughout the network, we need algorithms and protocols that utilize the capacity available in the network optimally. One of the standard methods to manage high throughput in the network is to compute a backbone structure. Recently, Censor-Hillel et al. [20, 21] proposed an algorithm that allows to build a network topology based on Fractional Connected Dominating Set (FCDS, see Section 7.2 for a definition), which can be seen as a generalized Connected Dominating Set (CDS). Such fractional connected dominating sets can be used to achieve a broadcast throughput of $\Theta(k/\log n)$ messages in networks with n nodes and vertex-connectivity k, which is optimal for routing-based algorithms [20]. This improves on the standard method of using one backbone by, intuitively, replacing it with as many fractional backbones as the net-

work can fit due to its connectivity. To give further intuition, we show an example network that admits multiple FCDS (a so-called FCDS packing) in Figure 7.1.

In this chapter we propose a modified version of the distributed algorithm originally proposed in [21]. Our modified algorithm is expected to be beneficial for future large-scale wireless sensor networks, as such networks are likely to consist of a huge number of small wireless sensors deployed on a relatively large area. Additionally, those sensors operate on small transmission ranges to conserve energy while staying operational, which potentially results in networks with large diameter. We present a distributed algorithm that computes a FCDS packing by explicitly computing the connector paths between not-yet connected components of the respective FCDS. Our algorithm runs in the message-passing model $\mathcal{CONGEST}$ and has a round complexity of $\mathcal{O}(\log^2 n(\min\{\frac{n\log n}{k}, D+\sqrt{n\log n}\log^* n\}+k\Delta))$. This improves the runtime of previously $\mathcal{O}(\log^3 n\cdot\min\{\frac{n\log n}{k}, D+\sqrt{n\log n}\log^* n\})$ for large and relatively sparse networks with moderate connectivity k, or more precisely if $k\Delta \in o(\min\{\frac{n\log n}{k}, D, \sqrt{n\log n}\log^* n\}\log n)$. This is for example true if the connectivity and the degree are less than $\sqrt{D}$. One example for a family of networks that benefit from our improvement is described in Section 7.2. Intuitively the achieved complexity is beneficial for networks with large diameter and moderate density, and generally for large but sparse networks.

Our algorithm is based on the virtual graph structure Censor-Hillel et al. [20, 21] use to compute a FCDS packing. In their distributed implementation they do not explicitly compute the connector paths between the components but rely only on the existence of a sufficient number of paths, which requires additional coordination within tentatively established dominating sets (so-called components). In [21], the approach of explicitly computing the paths is rendered as probably too expensive. However, our algorithm improves the runtime while explicitly computing possible connector paths between tentative components. Let us briefly introduce the layered approach used in both their algorithm as well as ours. For each node in the network we introduce a set of $\mathcal{O}(\log n)$ virtual nodes, each virtual node shall be assigned to one FCDS, resulting in a FCDS packing in which each FCDS has at least weight $1/\mathcal{O}(\log n)$. The virtual nodes are assembled to a virtual graph $\mathcal{G}$, and partitioned in layers such that there are 1 to 3 copies of a node in each layer. A nodes copy is connected to all copies of the nodes original neighbors in G (cf. Section 7.2 for details). Now, layer for layer, the virtual nodes are assigned different classes, each of which shall result in a FCDS once the algorithm finishes. Using the first half of the layers, dominating sets are formed, which are then connected by selecting so-called connector paths between components using nodes from the remaining layers. Both our algorithm and the algorithms of [20, 21] follow this general scheme.

The improvement in our algorithm is achieved by improving the process of how connector paths are matched to existing components. In [21], paths are matched to components by building a so-called bridging graph. In the bridging graph, whole components (which may span large parts of the network) are simulated by a virtual node and participate in a matching. Thus, the matching algorithm

must communicate through the component, which may require time in the order of $\Omega(D)$ in each step of the matching. In our algorithm we construct a matching graph that can directly be executed by transmitting one message to each neighbor in each round. As we determine the connector paths explicitly, this routine must be executed separately for each FCDS that is computed, resulting in a factor of k in our runtime. Overall, finding the connector paths requires $\mathcal{O}(k\Delta \log n)$ rounds for each one of the $\mathcal{O}(\log n)$ layers. Once the connector paths are found, they can be used to connect the components in order to create multiple CDS in the virtual graph. By translating each CDS to a FCDS in the network, $\Theta(k)$ communication backbones are established in the network, each with weight $1/\log n$.

Related Work

Research on FCDS was started by Censor-Hillel et al. in [20]. They propose a centralized algorithm to compute CDS partitions of size $\Omega(k/\log^5 n)$, as well as FCDS packings of size $\Omega(k/\log n)$, where k is the vertex-connectivity of the network. FCDS packings are the natural generalization of CDS partitions, which allows each node to participate in multiple CDS with a fractional weight between 0 and 1 per CDS such that the sum of the weights is at most 1. Additional to computing FCDS packings, they show that the broadcast throughput using a FCDS packing of size $\Omega(k/\log n)$ is $\Omega(k/\log n)$ messages per round, which is optimal if restricted to routing-based approaches. In contrast to network coding [39], which can achieve a throughput of $\Theta(k)$ broadcast messages [20], routing-based approaches consider messages as atomic tokens and use simple store-and-forward methods to route the message. Ene, Korula and Vakilian [34] consider FCDS packings under the constraint that each node has a capacity and compute packings of size $\Omega(k)$ for planar and minor-closed families of graphs, and $\Omega(k/\log n)$ for the general case using centralized algorithms. The first distributed implementation is again due to Censor-Hillel, Ghaffari and Kuhn [21]. In this work they consider both vertex- and edge-connectivity. For vertex-connectivity they compute a FCDS packing (or a fractionally disjoint weighted dominating tree, which is a similar concept) of size $\Omega(k/\log n)$, building on the initial approach in [20] as we do in this chapter.

Other related topics are connected dominating sets [27, 59], as well as dominating set partitions (see e.g. [37]). Regarding the application of FCDS to increase the throughput, related works are multi-message broadcasting in wireless networks [17] and network coding [39].

Contribution

Our main contribution is that we improve the distributed computation of FCDS packings of size $\Theta(k/\log n)$ for large but relatively sparse networks to a runtime of $\mathcal{O}(\log^2 n(\min\{\frac{n\log n}{k}, D+\sqrt{n\log n}\log^* n\}+k\Delta))$ rounds in the $\mathcal{CONGEST}$ model.

To achieve this result we show how to explicitly compute the connector paths between components, which is interesting on its own. We show how to construct and distributively compute a bipartite helper graph, which is then used in a maximal

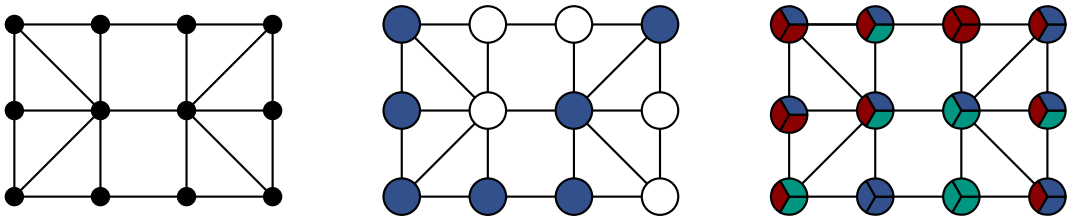


Figure 7.1: From left to right: A 3-vertex connected graph G, a (not optimal) connected dominating set in G marked by blue nodes, and a FCDS packing in G establishing 3 fractional connected dominating sets.

matching to select a sufficient fraction of the available connector paths. Interestingly, we can construct the helper graph and compute the connector paths without communication through the components.

Outline

In the next section we introduce the notation used in the construction of the FCDS packing. In Section 7.3 we describe how the FCDS can be computed distributively. Our main contribution is described in Section 7.4, where we show how to explicitly compute the connector paths between components. We conclude this chapter in Section 7.5.

7.2 PRELIMINARIES

Our algorithms operate on the communication graph $G = (V, E)$ of a wireless ad hoc or sensor network. As described in Section 2.1, an edge $e = (u, v) \in E$ is in the communication graph if $v, u \in V$ can communicate with each other directly in the network. We assume the communication to be bidirectional, and hence the communication graph to be undirected. We use the standard $\mathcal{CONGEST}$ message-passing model, which we introduced in Section 2.2.1. Communication is based on synchronous rounds, in which each node can receive the messages of its neighbors as well as transmit one identical message to all neighbors itself. In contrast to the edge-$\mathcal{CONGEST}$ model, in which the nodes may transmit different messages to their neighbors, the congestion is on the nodes instead of the edges of the corresponding communication graph. Note that this fits the broadcast nature of wireless networks.

A dominating set $S \subseteq V$ is a set of nodes in the network such that for each node $v \in V$ it holds that either $v \in S$ or a neighbor of v is in S. If such a set is connected we denote it by connected dominating set (CDS). *Fractional CDS* (FCDS) packings are the natural generalization of CDS. In an FCDS packing, each node can participate in multiple FCDSs with a weight $x_i \in [0, 1]$ for each FCDS, such that $\sum_i x_i \leq 1$ for each node. We illustrate a CDS and a FCDS in Figure 7.1. The virtual graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ used in the construction of the FCDS was first introduced in [20]. For each node $v \in V$ we introduce $\mathcal{O}(\log n)$ copies in the set of virtual nodes $\mathcal{V}$. Each copy of v is connected to all other copies of v in $\mathcal{V}$ and to each

FCDS

virtual graph $\mathcal{G}$

copy of a neighbor $w \in V$ of v in G. We denote the neighbors of v in G by N_v, and in $\mathcal{G}$ by $\mathcal{N}_v$. In contrast to the original description, which was also used for the first distributed implementation in [21], we use $3L$ copies of each node instead of $4L$ for $L \in \mathcal{O}(\log n)$, however, this is a minor technical detail. In total the virtual graph has $3L$ copies of the original graph, plus some additional edges. We subdivide the virtual graph in layers and call the first L copies of V in $\mathcal{V}$ the *lower layers*. Each so-called *upper layer* consists of two copies of V. We call the nodes of the first copy *type-1 nodes*, and the nodes of the second copy *type-2 nodes*. For each layer l we denote the nodes of layers 1 to l by $\mathcal{V}_l$ and the subgraph induced by these nodes by $\mathcal{G}_l$.

lower layers
upper layer
type-1 nodes
type-2 nodes

As we compute multiple FCDS simultaneously in the virtual graph and distinguish each FCDS by a class $i \leq t \in \Theta(k)$. We denote the subset of nodes of class i up to layer l by $\mathcal{V}_l^i$, and the induced subgraph by $\mathcal{G}_l^i$. We use $\Psi(v_l) = v$ to project from nodes (or a set of nodes) of the virtual graph to the corresponding real node(s). Throughout the rest of this chapter we shall use the term *node* to refer to virtual nodes in $\mathcal{G}$, and *real node* to refer to a node in G. During the execution of the algorithm, we aim at connecting not-yet-connected components of the dominating set of a class i to other components of the same class. Given a connected component $\mathcal{C}$ we use so-called connector paths to identify vertices that could connect $\mathcal{C}$ to another component $\mathcal{C}'$ with $\Psi(\mathcal{C}) \cap \Psi(\mathcal{C}') = \emptyset$, both of the same class. In compliance with [20] we call a path P a *connector path* for component $\mathcal{C}$ if it satisfies the following conditions:

$\Psi(v_l)$

connector path

a) P has one endpoint in $\mathcal{C}$ and the other endpoint in $\mathcal{C}'$.

b) P has at most two internal vertices.

c) P cannot be shortened, i.e., for $P = (s, v, w, u)$ with $s \in \mathcal{C}$ and $u \in \mathcal{C}'$, u does not have a neighbor in $\mathcal{C}'$ and v does not have a neighbor in $\mathcal{C}$.

d) if $P = (s, v, w, u)$ with $s \in \mathcal{C}$ and $u \in \mathcal{C}'$ has two internal nodes, v is of type-2 and w of type-1.

e) if $P = (s, v, u)$ with $s \in \mathcal{C}$ and $u \in \mathcal{C}'$ has one internal vertices, v is of type-1.

Connector paths can have at most one or two internal nodes, as the components of each class are already dominating. We call connector paths with one internal node *short* and those with two internal nodes *long*. For a path $(v_1, v_2, \ldots, v_{i-1}, v_i)$, we call the set of nodes $\{v_2, \ldots, v_{i-1}\}$ the *internal nodes* or *internal vertices*. We call a set of paths $\{P_1, \ldots, P_j\}$ *internally vertex-disjoint* if the internal nodes of $\{P_1, \ldots, P_j\}$ are mutually disjoint.

short
long
internally vertex-disjoint

The following lemma states that we always find at least k connector paths for each component in a k-connected graph.

Lemma 7.1 (Lemma 4.3 in [21])**.** *For each component $\mathcal{C}$ of an arbitrary class i at an arbitrary level l it holds that $\mathcal{C}$ has at least k internally vertex disjoint connector paths*

Note that the algorithms proof of correctness requires the connector paths of one component to be internally vertex-disjoint. We assume our connector paths to have this property in the following section. It is easy to verify (based on Menger's Theorem [102] or [3, Theorem 10.4]) that enough short connector paths are available. For long connector paths, we shall explain how a sufficient number of internally vertex-disjoint long connector paths can be found in Section 7.4. In the virtual graph $\mathcal{G}$, our algorithm computes t CDSs, as each node selects exactly one class i. Let us now briefly consider how this translates to a FCDS packing in G. Let $v \in G$ be a node of the network and $v_1, \ldots, v_{3L}$ the corresponding virtual nodes in $\mathcal{G}$. Given $t = \Theta(k)$ connected dominating sets in $\mathcal{G}$, we can construct a FCDS in G by weighting the class of each virtual node v_i by $1/3L$ at the real node v. As there are $3L$ virtual copies of v, the weight constraint is satisfied, and the CDS partition translates to a FCDS packing. Before describing the distributed FCDS algorithm, let us consider an example network that benefits from our runtime improvement.

Example network

Given a network with n nodes and a parameter d. We distribute the nodes of the network on d lines with n/d nodes on each line. We iterate over the nodes on each line and connect the ith node on a line to the node before and after i on the line and to the ith nodes on the other lines. Thus, d nodes, one from each of the lines, form a clique. For $d = 4$, an example of such a network is displayed in Figure 7.2. We can see that such a network is d connected, and has maximum degree $d + 1$.

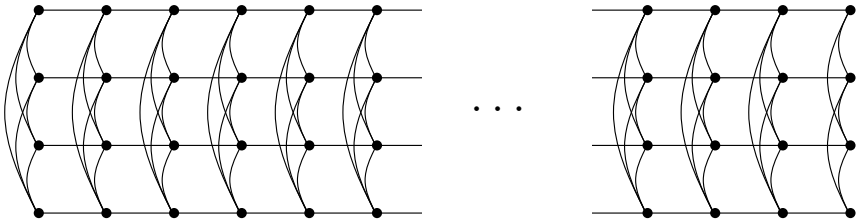


Figure 7.2: A $d = 4$-vertex connected network, with diameter in $\Omega(n/d)$.

The diameter of the network, however, is in $\Omega(n/d)$. Thus, even for instances with very high connectivity $d = \sqrt[3]{n}$, our algorithms results in an improvement over [21] as $k\Delta = D = \mathcal{O}(n^{2/3})$. For sparser variants our algorithm is faster as well.

7.3 DISTRIBUTED FCDS COMPUTATION

Computing the $t = \Theta(k)$ connected dominating sets in $\mathcal{G}$ consists of two main components. Recall that we assign each virtual node to one of t classes, which shall form the CDSs after the execution of the algorithm. The first $\mathcal{O}(\log n)$ layers of virtual nodes establishes that each class dominates the whole graph with high probability, cf. Lemma 7.2. This is surprisingly simple and can be achieved by having each virtual node select one of the classes at random. For the second $\mathcal{O}(\log n)$ layers we aim at connecting a constant fraction of the connected compo-

nents in each layer with constant probability. This leads to connectivity of each class with high probability after the last layer, yielding the desired CDSs. The existing distributed algorithm to compute FCDS packings uses the same approach for the lower layers, and essentially matches existing components in each of the upper layers without computing the connector paths. Our approach on the other hand explicitly computes the long connector paths by constructing a helper graph in which a matching algorithm finds $\Omega(k)$ such paths. Thus we do not require communication through existing components to compute the paths, which is beneficial for many networks, especially if they are large with respect to the diameter. Our algorithm consists of the following steps. Note that the overall design of the algorithm is similar to that of [21], however, we use a different method to connect the components of each class, which is one of the key parts of the algorithm.

A) Each virtual node in the lower layers randomly selects one of the t classes. This leads to domination of each class w.h.p., cf. Lemma 7.2.

B) For each upper layer l from L to $2L$ we try to connect existing connected components of each class in the nodes of layers 1 to $l-1$ using nodes of layer l. We call the nodes of the previous layers 1 to $l-1$ *old nodes* and the nodes of layer l *new nodes*. For each layer we execute steps B.1 to B.4.

old nodes

new nodes

B.1) Identify connected components of old nodes. We use the protocol of [21], which is based on an algorithm by Thurimella [124] to identify connected components. To be self-contained, we describe the protocol in Appendix A.4.

B.2) Let nodes of type-1 select a random class.

B.3) For each class i: If the nodes component is not yet connected by short a connector path, find $\Omega(k)$ internally vertex-disjoint long connector paths. We construct a helper graph $\mathcal{H}_i$ and run a simple matching algorithm to find the long connector paths. For details on this step we refer to Section 7.4.

B.4) If the nodes type-2 node is on long connector paths, the node discards the paths for which the type-1 node selected a wrong class, and selects the class of one of the remaining paths at random. If no path remains, the node selects a random class.

After executing this algorithm each virtual node in $\mathcal{G}$ has assigned itself to one of the t classes. Each class dominates the whole graph (Step A) and is connected (Step B). Thus, the nodes computed $t = \Theta(k)$ CDSs in the virtual graph $\mathcal{G}$. The CDSs can be converted to one FCDS of size $\Omega(k/\log n)$ by assigning each CDS a weight of $1/3L$, cf. Section 7.2. Note that the matching algorithm in Step B.2 matches type-1 with type-2 nodes, thus it does not require communication and coordination within the components. Let us now briefly reference the result that achieves dominance in the lower layers.

Lemma 7.2 (Lemma 4.1 in [21]). *For each class i, $\mathcal{V}_l^i$ is a dominating set in $\mathcal{G}$ w.h.p.*

The proof idea is based on the fact that, for class i and a node $v \in V$, the probability that v's virtual copy on layer l selects i is at least $1/t = 1/\mathcal{O}(k)$. As each node has at least k neighbors on l, this yields constant probability per layer, and w.h.p. over all $\log n$ layers.

7.4 FINDING CONNECTOR PATHS

In this section we show how our algorithm computes internally vertex-disjoint connector paths for each component in order to connect a constant fraction of the components in each upper layer. We begin this section by giving a high-level proof showing that we can indeed connect a constant fraction of the components with each new layer. In the next sections we introduce the necessary tools and prove the remaining results. In Section 7.4.1 we introduce the graph $\mathcal{H}_i$, which helps to reduce the problem of finding long connector paths for each component to a matching problem. The matching problem is discussed in more detail in Section 7.4.3.

As introduced in Section 7.2, connector paths can have one or two internal nodes, we call them short and long connector paths, respectively. To prove correctness for the algorithm, it must hold for each one of the upper layers that at least a constant fraction of the components (formed by old nodes) of each class are connected to another component of the same class using nodes from the current layer with at least constant probability. We shall now state the overall result of this section, which was first obtained and proven in [21]. We give a brief sketch of the proof to be self-contained. Note that there is a minor technical flaw in the original proof regarding the number of missing connections M_l in layer l, see [130, p. 21] for details and a corrected proof.

Lemma 7.3 (Lemma 4.4 in [21])**.** *Let $l \in [L, 3L]$. Then $M_{l+1} \leq (1-\delta)M_l$ with probability at least ρ.*

Sketch of proof, based on [21, 130]. To prove the theorem we consider each component and show that the component is connected to another component by layer $l+1$ with constant probability. This implies that a constant fraction of the existing components are connected by layer $l+1$ with constant probability. Given a component $\mathcal{C}$ of class i on layer l and assume class i has at least two components. It holds that $\mathcal{C}$ has at least k connector paths, connecting $\mathcal{C}$ to another component of class i, according to Lemma 7.1. There are two cases: Either at least $k/2$ of the paths are short connector paths, or at least $k/2$ of the paths are long connector paths.

Let us first consider the case of at least $k/2$ short connector paths. This is intuitively the easier case, as only one node separates $\mathcal{C}$ from another component on $\Omega(k)$ paths. Recall that each layer has two copies of each real node: a type-1 and a type-2 node. Let us consider only the type-1 node for now. According to Lemma A.4 (in Appendix A.4, from [20]) it is sufficient that all type-1 nodes select a random class to connect $\mathcal{C}$ to a neighboring component with constant probability in this case. Intuitively, this holds as $\Omega(k)$ nodes can connect $\mathcal{C}$ with another component, and each of these nodes selects one of t classes.

For the case of less than $k/2$ short connector paths, it holds that there are $\Omega(k)$ long connector paths as in total k connector paths exist according to Lemma 7.1. We use the remainder of this section to prove that a component $\mathcal{C}$ selects a long connector path with constant probability. The theorem follows with the result stated in Lemma 7.13: At least one of the connector paths selects the required class with at least constant probability. □

7.4.1 *Helper Graph $\mathcal{H}_i$*

Finding internally vertex-disjoint long connector paths is only relevant if a component has less than $k/2$ internally vertex-disjoint short connector paths. We consider this case here as short connector do not need to be found explicitly if sufficiently many exist (Step B.2) is sufficient). Since each component has at least k internally vertex disjoint connector paths, the component must have at least $k/2$ long connector paths in this case. We introduce a helper graph in this section, which is defined such that a maximum matching in this graph corresponds to finding a maximum number of internally vertex-disjoint long connector paths.

helper graph $\mathcal{H}_i$

For each class i on an upper layer l we define the *helper graph* $\mathcal{H}_i^l$ as the union of the helper graphs $\mathcal{H}_i^l[\mathcal{C}]$ constructed for each component $\mathcal{C}$ of class i on layer l. Note that although the helper graphs are constructed for each layer, we omit the layer l in the following as the helper graphs are used only in the layer in which they are constructed. Thus, we always refer to the helper graph of the current level.

Let us now define the helper graph $\mathcal{H}_i[\mathcal{C}]$ for class i and component $\mathcal{C}$. For each type-2 node v of layer l we add a node $v_\mathcal{C}$ to $\mathcal{H}_i[\mathcal{C}]$ iff the following three conditions are met:

1) $\Psi(v) \notin \Psi(\mathcal{C})$.
2) v has a neighbor in $\mathcal{C}$.
3) v does not have a neighbor belonging to another component of class i.

For each node v_C we added to $\mathcal{H}_i[\mathcal{C}]$, we add for each type-1 neighbor w of v a node w_C to $\mathcal{H}_i[\mathcal{C}]$, if w has a neighbor in another component $\mathcal{C}'$ of class i but no neighbor in component $\mathcal{C}$. Intuitively , this procedure ensures that we added the potential long connector path of component $\mathcal{C}$ to $\mathcal{H}_i[\mathcal{C}]$ using a type-2 node as the node closer to C and a type-1 node as the node closer to the neighboring component of the same class. An edge between v_C and w_C is added to $\mathcal{H}_i$ as there is an edge between v and w in $\mathcal{G}$. We illustrate the construction of the helper graph $\mathcal{H}_i$ in Figure 7.3.

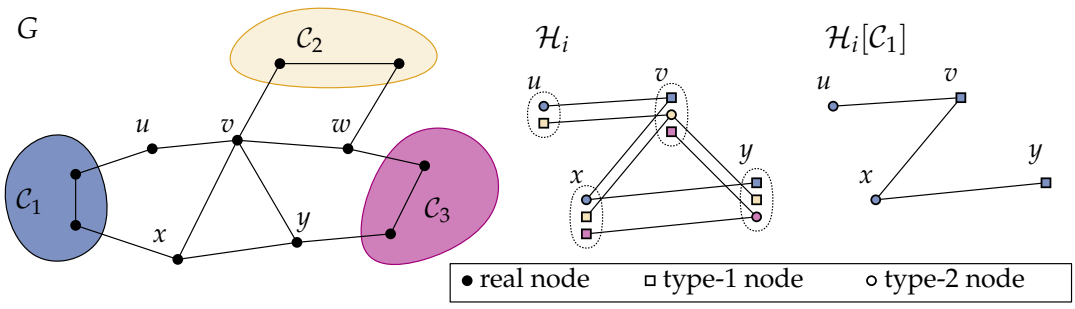


Figure 7.3: A graph G with three components of a class i, along with the helper graph $\mathcal{H}_i$ and $\mathcal{H}_i[\mathcal{C}_1]$ restricted to component C_1. Note that w is not in the helper graph as it is on a long connector path.

The connection between long connector paths of a component and edges in $\mathcal{H}_i$ is shown in the following lemma.

Lemma 7.4. *There is an edge (v_C, w_C) in $\mathcal{H}_i[\mathcal{C}]$ iff there is a long connector path from $\mathcal{C}$ to another component of class i through v and w on the current layer.*

Proof. Let us first assume an edge (v_C, w_C) is added to $\mathcal{H}_i[\mathcal{C}]$. Then v has at least one neighbor in $\mathcal{C}$, which we denote by s. Also, v has a neighbor w (of type-1) which does not have a neighbor in component $\mathcal{C}$ but has at least one in a component $\mathcal{C}' \neq \mathcal{C}$ of class i. Let us denote this neighbor by u. We claim that $P = (s, v, w, u)$ is a long connector path (cf. Section 7.2 for the definition), which holds as a) P has one endpoint in $\mathcal{C}$, the other in $\mathcal{C}' \neq \mathcal{C}$ of class i, b) P has two internal vertices, c) P cannot be shortened as v does not have a neighbor in a component $\mathcal{C}' \neq \mathcal{C}$ of class i and w does not have a neighbor in $\mathcal{C}$, and d) v is of type-2 while w is of type-1.

Let us now assume we have a long connector path $P = (s, v, w, u)$. Then it holds that 1) w does not have a neighbor in $\mathcal{C}$, 2) s is in $\mathcal{C}$, and 3) w is not in $\mathcal{C}$ and v is of type-2. Thus, v_C is added to $\mathcal{H}_i[\mathcal{C}]$. Furthermore, w is of type-1 and has a neighbor u that is in another component $\mathcal{C}' \neq \mathcal{C}$ of class i but no neighbor of component $\mathcal{C}$, which implies that w_C and the corresponding edge (v_C, w_C) are added as well. □

The matching algorithm is executed on $\mathcal{H}_i$, which is the union of helper graphs for each component, however, observe that we know for each edge in $\mathcal{H}_i$ from which component it is induced.

Observation 7.5. *Given an arbitrary layer $l \geq L$, a class i, and the corresponding helper graph $\mathcal{H}_i$. Then each edge in the helper graph can be attributed to exactly one component $\mathcal{C}$.*

We have shown that the construction ensures that there is a vertex disjoint long connector path through v and w for component $\mathcal{C}$ iff there is an edge between v_C and w_C in $\mathcal{H}_i[\mathcal{C}]$. Thus a matching induces long connector paths. We shall argue in Section 7.4.3 that we can compute a matching of size $\Omega(k)$ in $\mathcal{H}_i$ for each component of class i with $\Omega(k)$ long connector paths. However, let us first describe the distributed algorithm to construct $\mathcal{H}_i$

7.4.2 *Distributed Construction of $\mathcal{H}_i$*

Due to Step B.1 of the algorithm, which is executed for each layer before constructing the graph $\mathcal{H}_i$, each type-2 node v knows the classes and components of its neighbors. Thus v can decide whether a node v_C should be added for neighboring components $\mathcal{C}$. Note that due to Lemma A.5 (in Appendix A.4) a type-2 node lies only on one long connector path for each class, however, up to t components may have a long connector path through v, see Observation A.6 in Appendix A.4. If v adds v_C for a component $\mathcal{C}$ to $\mathcal{H}_i[\mathcal{C}]$, it transmits this information along with the class i and the ID of $\mathcal{C}$ (which is also used in Step B.1) to its type-1 neighbors. These type-1 neighbors can now easily check whether they have a neighbor in $\mathcal{C}$ and resign, or verify if at least one neighbor is in another component of class i due to information obtained during Step B.1. If so, w adds w_C and the edge between w_C and v_C to $\mathcal{H}_i[\mathcal{C}]$.

Lemma 7.6. *For each class i on layer l it requires $\mathcal{O}(\Delta)$ rounds to construct $\mathcal{H}_i$.*

Proof. First note that each real node v simulates exactly the two virtual copies of v on layer l. Due to Conditions 2) and 3) in the definition of $\mathcal{H}_i$, the type-2 copy of a node participates in $\mathcal{H}_i$ only if its neighbors of class i belong to the same component $\mathcal{C}$. In this case, v sends the ID of $\mathcal{C}$, which requires one message. After receiving these messages, each type-1 node w transmits one message for each message they received from a type-2 node. Note that as w has received at most one message from each neighbor, w responds to at most Δ messages. Hence, this results in $\mathcal{O}(\Delta)$ messages. □

The following observation follows from the fact that a type-2 node is only added to $\mathcal{H}_i$ if all its neighbors of the class i are in one component, while one type-1 copy is added for each message received by another type-2 node. It helps bounding the runtime of our matching algorithm operating on $\mathcal{H}_i$.

Observation 7.7. *For each node $v \in V$, there is at most one type-2 copy in $\mathcal{H}_i$, but up to Δ type-1 copies in $\mathcal{H}_i$.*

7.4.3 *Matching Internal Vertices*

Let us now consider how to distributively compute a matching of cardinality $\Omega(k)$ in the helper graph $\mathcal{H}_i$ for each component. We shall use this in the next section to prove that each component finds a long connector paths with constant probability in each layer. We use a simple distributed maximal matching algorithm, which is based on a parallel matching algorithm by Israeli and Itai [74]. The matching algorithm is similar to that of Censor-Hillel et al. [21] , however, in our case each node in the helper graph $\mathcal{H}_i$ is simulated by only one node and not by several nodes that have to coordinate their actions through components. The algorithm makes use of the special structure of $\mathcal{H}_i$.

Lemma 7.8. *The helper graph $\mathcal{H}_i$ is bipartite.*

Proof. As described in the previous section, all nodes in the helper graph $\mathcal{H}_i$ are either added by a type-1 or type-2 node of $\mathcal{G}$. Hence, we may say that the nodes of the helper graph also have these types, which induces a partitioning of the nodes of $\mathcal{H}_i$. For two nodes v_C and w_C to be connected in $\mathcal{H}_i$, it must hold that v is of type-2 and w of type-1 or vice versa. Thus, all edges in $\mathcal{H}_i$ connect nodes of different types, which shows that the graph is bipartite. □

Using this lemma the matching algorithm operates as follows. A node is *active* exactly if none of the adjacent edges is matched, and an edge is *active* if both adjacent nodes are active. In each step of the matching algorithm, we assign random numbers from a sufficiently large range to all active edges such that no two edges have the same number w.h.p. Since $\mathcal{H}_i$ is bipartite, assigning the numbers is particularly easy as each type-2 node can pick a number for each incident edge. Each active type-2 node then selects the edge with the largest number and sends its choice to

its neighbors. In this step only the selected edges may be added to the matching. At this point, there is at most one edge selected at each type-2 node. However, each type-1 node may have received more than one proposal. To satisfy the matching condition, each type-1 node that has received at least one proposal picks the proposed edge with the largest number and adds it to the matching. The two adjacent nodes and their incident edges become inactive. It can be shown that after $\mathcal{O}(\log n)$ steps all edges are deactivated with high probability and thus, a maximal matching is achieved [74]. Let us now show that such a maximal matching is of cardinality $\Omega(k)$ if the corresponding component has $\Omega(k)$ long connector paths.

Lemma 7.9. *Given a component $\mathcal{C}$ of class i with $\Omega(k)$ long connector paths. A maximal matching in $\mathcal{H}_i[C]$ is of cardinality at least $\Omega(k)$.*

Proof. If follows from Lemma 7.1 and the one-to-one correspondence of the long connector paths and the edges in $\mathcal{H}_i$ of Lemma 7.4 that there are $\Omega(k)$ independent edges in $\mathcal{H}_i[C]$. Thus, the maximum matching is of size at least $\Omega(k)$. As the maximal matching is at least a 2-approximation to the maximum matching, it is also of size $\Omega(k)$. □

After showing that the matching is of sufficient size, we prove that this allows us to identify the $\Omega(k)$ long connector paths for each component.

Lemma 7.10. *Consider a component $\mathcal{C}$ of class i with $\Omega(k)$ long connector paths. Then a maximal matching in $\mathcal{H}_i$ identifies $\Omega(k)$ long connector paths for $\mathcal{C}$.*

Proof. Let us consider a maximal matching in $\mathcal{H}_i$, and component $\mathcal{C}$ as required. According to Observation 7.5, we can consider the subgraph $\mathcal{H}_i[\mathcal{C}]$ of $\mathcal{H}_i$ corresponding to component $\mathcal{C}$ as disjoint from other parts of $\mathcal{H}_i$. Thus, the matching is maximal also in $\mathcal{H}_i[\mathcal{C}]$. It holds by Lemma 7.9 that the size of the maximal matching is $\Omega(k)$. It remains to show that two independent edges in $\mathcal{H}_i[\mathcal{C}]$ correspond to two internally vertex disjoint connector paths. Consider the edges (v_C, w_C) and (v'_C, w'_C) and assume the corresponding long connector paths with internal vertices v, w and v', w' are not internally vertex disjoint. Thus, either $v = v'$ or $w = w'$ which implies either $v_C = v'_C$ or $w_C = w'_C$. This contradicts the assumption as the edges are not independent. As each matched edge is independent, the set of matched edges in $\mathcal{H}_i[\mathcal{C}]$ corresponds to a set of internally vertex-disjoint long connector paths of cardinality $\Omega(k)$. □

The correspondence between a maximal matching in $\mathcal{H}_i$ and long connector paths in G is depicted in Figure 7.4. Let us now consider the number of time slots required to compute the maximal matching. The matching algorithm is executed once for every class i, and operates on the virtual graph $\mathcal{H}_i$. The next lemma proves that $\mathcal{O}(\Delta)$ rounds are sufficient for each step of the matching algorithm.

Lemma 7.11. *In each step of the matching algorithm on $\mathcal{H}_i$, we transmit over each real edge at most twice in each direction. Thus $\mathcal{O}(\Delta)$ rounds are sufficient for each step.*

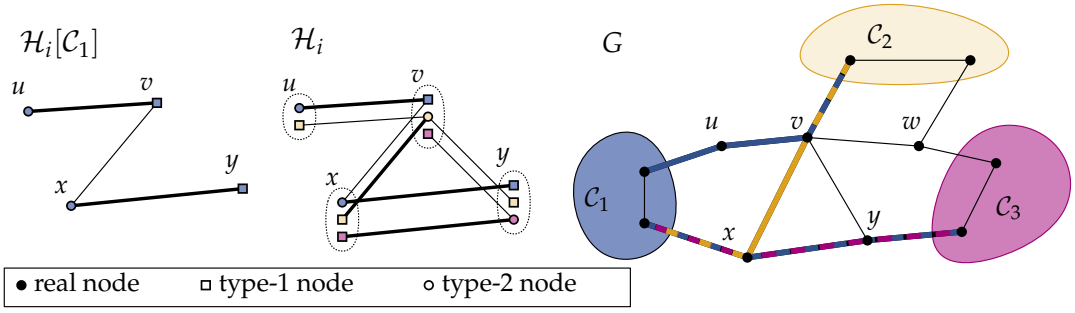


Figure 7.4: A maximal matching in the helper graph $\mathcal{H}_i$ is marked by bold lines. The corresponding long connector paths are marked in G. Several long connector paths may use the same nodes, however, each path is at most used twice (once from each direction).

Proof. Let v be an arbitrary real node, and note that there may be up to Δ copies of v as type-1 node in $\mathcal{H}_i$, but only one copy of v as type-2 node in $\mathcal{H}_i$ according to Observation 7.7. Consider any real edge from v to an arbitrary neighbor w. We may assume that there is at least one copy of the edge (v, w) in the helper graph $\mathcal{H}_i$, as otherwise this edge is not used for the matching algorithm at all. Since the type-2 copy of v sends only one message over one of its incident edges, it uses the edge (v, w) at most once. After the type-1 copies of v have received the messages from the type-2 nodes, they respond to one of them. Hence, each type-1 copy sends at most one message. It remains to show that no two type-1 copies use the same real edge. Assume that there were two type-1 copies of v that transmit over the real edge (v, w). This would imply that both type-1 copies have received a message from the type-2 copy of w over the real edge (w, v). However, we have shown above that each real edge is used at most once by the type-2 nodes, which contradicts our assumption. Thus, in one step each real edge transmits at most one message from a type-1 and one from a type-2 copy of v. Combined with the analogous considerations for w this results in two messages per edge per direction. As we operate in the $\mathcal{CONGEST}$ model, this results in $\mathcal{O}(\Delta)$ rounds per step. □

It follows from [74] that $\mathcal{O}(\log n)$ rounds are sufficient to compute a maximal matching with high probability.

Corollary 7.12. *Our distributed randomized matching algorithm computes a maximal matching in $\mathcal{H}_i$ in $\mathcal{O}(\Delta \log n)$ time.*

This implies that we can find $\Omega(k)$ long connector paths for all components of one class that have less than $k/2$ short connector paths in time $\mathcal{O}(\Delta \log n)$. As we have $t = \Theta(k)$ classes, this results in $\mathcal{O}(k\Delta \log n)$ rounds for Step B.3 on each layer. Let us now prove that the long connector paths can indeed be used to connect the components with at least constant probability on each layer.

7.4.4 *From Long Connector Paths to Connected Components*

As components with at least $k/2$ short connector paths are connected using those connector paths, we keep focusing on components with at least $k/2$ long connector paths. In the previous section we showed how to find $\Omega(k)$ vertex-disjoint long connector paths for each such component. As type-1 nodes already selected a random class to connect those components that have a sufficient number of short connector paths, the class of the type-2 nodes on the current layer remains to be selected. Since each type-2 node lies on at most one long connector path per class, there are at most t long connector paths per type-2 node. On these paths, however, the internal type-1 nodes may have chosen classes that differ from the class of the path. Intuitively, this means that the path cannot be used to connect two components of the same class since one of the internal nodes has already picked the wrong class. Therefore, as described in Step B.4, the type-2 nodes discard these long connector paths and select the class of one of the remaining paths at random. If no long connector paths remains, the node selects a random class.

We show in this section that this is sufficient to guarantee that a constant fraction of the components are connected with constant probability. Let us consider an arbitrary component $\mathcal{C}$ of class i. There are two challenges. The first is to show that each connector path connects to another component of the same class with probability in the order of $1/k$. This is non-trivial, as the type-1 node on each long connector path already selected a random class, which upper bounds the probability by $1/t$. The second challenge is, that the events that two type-2 nodes on different connector paths of $\mathcal{C}$ select class i are not necessarily independent. This can be circumvented by using a tail bound once the probability for each event is upper and lower bounded independently of the outcome of other events. Let us now state the result.

Lemma 7.13. *Given a component $\mathcal{C}$ of class i on an upper layer l with $\Omega(k)$ long connector paths. The probability that one of the long connector paths can connect the component with another component of the same class is at least δ.*

As the proof is similar to the original one, we shall only sketch the main ideas here. For more technical details we refer to [21] and [130].

The analysis is structured in three parts. The first part considers how likely it is that a type-2 node v selects another class, given that the corresponding type-1 node is of the correct class. In order to bound this probability, a random discard step for long connector paths is introduced. This allows to show that all other possible long connector paths through v are discarded with constant probability. This results in a probability in the interval $[1/4t, 1/t]$ for the event that both internal vertices of a long connector path of $\mathcal{C}$ select class i—independent of the class selected by nodes on other long connector paths of $\mathcal{C}$. In the second part it is shown that the proven bound holds even if the random discard step is not used (this is required, as it is not used in the algorithms). The third and final step uses the independent bounds on the probability of a long connector path to select class i with a tail bound to

show that for at least one of the $\Omega(k)$ long connector paths both internal vertices selected the same class i.

7.5 CONCLUSION

The algorithm presented in this chapter computes a fractional connected dominating set packing in the $\mathcal{CONGEST}$ model of distributed computation. It is based on an algorithm by Censor-Hillel, Ghaffari and Kuhn [20, 21], however, our distributed implementation computes the long connector paths explicitly, instead of matching components under the assumption that sufficiently many long connector paths exist. The runtime of our algorithm is $\mathcal{O}(\log^2 n(\min\{\frac{n\log n}{k}, D + \sqrt{n\log n}\log^* n\} + k\Delta))$, which is beneficial for large networks with moderate density, i.e. if $k\Delta \in o(\min\{\frac{n\log n}{k}, D, \sqrt{n\log n}\log^* n\}\log n)$. We expect future large-scale wireless ad hoc and sensor networks to satisfy such conditions. Open problems in this setting are: Can the dependence on k or Δ be reduced without communicating through the components to compute the connector paths? And is it possible to reduce the complexity of the protocol to simplify implementation on small sensor nodes?

Part III

CONCLUSION

8

CONCLUSION

In this thesis we studied several problems related to establishing initial and efficient communication in wireless ad hoc or sensor networks and proposed distributed algorithms to solve these problems.

In the first part we focussed on algorithms in the realistic SINR model of interference and studied two important and basic problems: In Chapter 3 we considered the local broadcast problem, which is the problem of allowing each node to successfully transmit one message to its neighbors. Solving this problem enables nodes in a network to establish initial communication without precomputing a communication structure or a backbone. Thus, the method can be used in wireless sensor networks just after their deployment. We studied the problem for heterogenous transmission powers and established a general bound on the probabilistic interference. This bound allowed us to generalize known local broadcasting algorithms from the case of uniform transmission power to this more general setting. The second problem we considered is distributed node coloring, which is the problem of assigning a color to each node such that no two neighbors have the same color. Such a coloring can be used to establish more efficient communication. We conclude Chapter 3 by generalizing a coloring algorithm to the case of heterogenous transmission powers using the results already obtained in the chapter. In Chapter 4 we proposed two algorithms that solve the problem in the classical setting of uniform transmission powers. The first algorithm, Rand4DColoring, is extremely simple and achieves a (4Δ)-coloring, while the second algorithm, ColorReduction, is more involved and computes a $(\Delta + 1)$-coloring. Our algorithms improve the state-of-the-art either by reducing the number of required colors or the runtime. We evaluated our algorithms experimentally and compared them to the existing distributed node coloring algorithms for the SINR model using a network simulator in Chapter 5. We found that both algorithms are faster than their competitors in all considered settings. In particular Rand4DColoring distributively computed a valid coloring very fast, even faster than one round of local broadcasting.

In the second part of this thesis we put our focus on higher-level algorithms such as those designed for the $\mathcal{CONGEST}$ model. In Chapter 6 we aimed at establishing communication schemes that enabled the efficient execution of algorithms designed for higher-level models in the SINR model. As the $\mathcal{CONGEST}$ model is round-based and does not consider interference, each round can be emulated by one execution of local broadcasting, however, this is not yet efficient. Thus, we

proposed a method to compute a local broadcasting schedule of asymptotically optimal length using a node coloring combined with position information. This allows to execute $\mathcal{CONGEST}$ algorithms in time asymptotically optimal. In Chapter 7 we improved an existing algorithm that increases the broadcast throughput in networks of high connectivity and operates in the $\mathcal{CONGEST}$ model. The algorithm uses multiple fractional connected dominating sets, each of which can be seen as a separate backbone structure. We proposed a new method to explicitly compute connector paths, which is used in the construction of the backbones and improves the runtime of the algorithm for large or moderately dense networks.

Outlook

There are several lines of research and interesting theoretical and practical open problems that we mentioned at the end of each chapter. Thus, we take a broader look on the algorithmic research for wireless sensor networks here.

Although the gap between theory and practice is considerable in many fields, this is particularly true for research on distributed algorithms for wireless ad hoc and sensor networks. Thus, although we design our algorithm for worst case models, it is important to bear more practical settings in mind. This includes testing the algorithms using simulation tools, as e.g. done in Chapter 5, however, it is by far not limited to that. Implementing algorithms on real hardware and testing them in real-world environments is not yet popular enough, although several testbeds for wireless sensor networks are available to researchers or even the general public [36].

Another interesting direction is to incorporate dynamics, such as mobility and node failures. So far, this is often considered only for graph-based models, however, the problem is also relevant in the SINR model. We observed in the experiments of our distributed node coloring algorithms that the simple Rand4DColoring algorithm, which did not build or maintain any structure, performed best in the mobile setting. One immediate question is: Can we construct theoretically sound algorithms that can cope with network dynamics without building and maintaining complex structures?

Another research direction that potentially has both theoretical and practical implications is to further explore the possibilities of the SINR model. Recall that a better throughput performance can be achieved (in practice!) under the SINR model than theoretically possible in less realistic models as the protocol model [105]. However, currently most distributed algorithm are not yet able to transfer this positive result and achieve a better performance in the SINR model than in the protocol model. An important research direction is to study how the specifics of the SINR model can be exploited to obtain faster algorithms. One approach that could lead into this direction is to differentiate successful message reception by more than the broadcasting range, as for example done in [79].

A

APPENDIX

In this appendix we provide additional material deferred from the main part of this thesis. In the first section we describe the modifications to Sinalgo to correct a flaw in the simulators SINR model implementation, which is deferred from Section 2.3. In Appendix A.2 we provide a proof related to an extended local broadcasting strategy used in Chapter 4. In Appendix A.3 we report results of experiments for the deployment strategies not presented in the main part of Chapter 5, and in Appendix A.4 we provide additional material for Chapter 7.

A.1 SINALGO - PATCH FOR SINR MODEL

In this section we report on a modification of the SINR interference model that is delivered with Sinalgo version 0.75.3 [33]. The modifications are required to ensure that the SINR interference constraints are correctly evaluated. Without modification, the simulation framework considers for a transmitted packet p the signal emitted during the transmission of the same packet p both as desired signal and interference. To correct this issue we equip each packet with a broadcast ID (or transmission ID), and ensure that we do only consider the interference of other packets (i.e., other transmissions). We state the modification in detail in Algorithms A.1 to A.3. The name of the algorithms gives the path to the respective file relative to the src folder of Sinalgo. The line number before and after the code marks the lines between which the code should be added (all line numbers are relative to the unchanged file).

Algorithm A.1: projects/defaultProject/models/interferenceModels/SINR.java

```
116
   /* detect if a packet is from the same broadcast as this packet.
   * If so, ignore the active packet.
   * If the broadcast id is -1, the packet is not from a broadcast,
   * and duplicate packets are found via pack == p
   */
   if (p.broadcastId != -1 && (pack.origin.ID == p.origin.ID &&
             && pack.broadcastId == p.broadcastId))
       continue;
   }
117
```

Algorithm A.2: sinalgo/nodes/messages/Packet.java

```
104
   /** broadcast id, allows to determine whether
   *  2 packets origined from the same broadcast
   */
   public int broadcastId;
105

194
   pack.broadcastId = -1;
195

259
   broadcastId = -1;
260
```

Algorithm A.3: sinalgo/nodes/Node.java

```
 77
   import sinalgo.tools.Tools;
 78

1487
   int broadcastId = Tools.getRandomNumberGenerator().nextInt(100000);
1488

1495
   sentP.broadcastId = broadcastId;
1496
```

A.2 DISTRIBUTED NODE COLORING: EXTENDING LOCAL BROADCASTING

We show in this section that local broadcasting with constant success probability in time inversely proportional to the transmission probability can be achieved. This extends known results regarding local broadcasting, which guarantee local broadcasting with high probability for a fixed number of time slots. As in these results, we require the nodes transmission probability to conform with the requirement that the sum of transmission probabilities from each broadcasting region is at most 1, which is stated in Lemma A.1. This local property is used in the following theorem to ensure that both the interference by few local transmissions as well as the summed interference by all globally simultaneous transmissions can be handled.

Lemma A.1. *Let all nodes in the network execute Algorithm 4.1, Algorithm 4.2, or Algorithm 4.3, and let v be an arbitrary node. Then the sum of transmission probabilities from within v's broadcasting range is at most* 1.

Proof. Depending on the algorithm, nodes in v's broadcasting range are either transmitting with probability p_1 or p_2. It holds that most nodes transmit with probability p_1 for all algorithms. For the use of probability p_2 in Algorithm 4.2 it holds that at most a 5 nodes from within each broadcasting range transmit with probability p_1, since in a broadcasting range in which more than 5 nodes have the same color, two of them must be neighbors. This would violate the validity of the given coloring. For Algorithm 4.3 it holds that at most 90 nodes within v's broadcasting range use p_1 according to Lemma 4.9.

Let us now consider both cases jointly, and let p_w be the current transmission probability of w, which is either p_1, p_2 or 0 if w is currently not trying to transmit (e.g. if $c_v^t \neq i$ in Algorithm 4.2). Let us now bound the sum of transmission probabilities from within B_v.

$$\begin{aligned}\sum_{\substack{w\in B_v,\\ w \text{ transmits}}} p_w &\leq \sum_{w\in B_v} p_1 + \sum_{\substack{w\in B_v,\\ w \text{ transmits with } p_2}} p_2 \\ &\leq \Delta \cdot \frac{1}{2\Delta^A} + 90 \cdot \frac{1}{180} \leq 1,\end{aligned}$$

where the last inequality holds since $r^A > 2r^B$ which implies $\Delta \leq \Delta^A$. □

For the remaining proof we require some additional definitions. For an arbitrary node v the proximity range around v is defined as $r^A = r^B \left(3^3 2^\alpha \beta \cdot \left(\frac{\alpha-1}{\alpha-2}\right)\right)^{\frac{1}{\alpha-2}}$, and denote set of nodes within the transmission range of v by A_v. Let $\chi := \frac{2\pi}{3\sqrt{3}} \frac{(r^A+2r^B)^2}{(r^B)^2}$, which is, intuitively speaking, a bound on the number of independent broadcasting regions that can be fit in a disk of radius r^A. Note that these definitions differ from those in Section 3.2, however, this is only a technicality that could be circumvented by modifying the transmission probability. Let us now state the lemma, which is first stated in Section 4.2.1. We shall prove it in the following.

Lemma 4.1. *Let v be a node transmitting with probability p_1. Then v successfully transmits to its neighbors with probability $\geq 11/12$ within κ_0 time slots. Transmissions with probability p_ℓ for κ_ℓ time slots are successful w.h.p. for $\ell \in \{1, 2\}$.*

The lemma implies that in each phase of Algorithm 4.1, the currently selected color is transmitted to all neighbors with constant probability. This is a fundamental part enabling us to analyze Algorithm 4.1 and correctly account for the uncertainty in the message transmission in this algorithm in Section 4.3.1. The lemma includes the standard local broadcasting bounds known from [55], but also enables the algorithms to use a more coordinated (and faster) medium access based on the coloring. Local broadcasting achieves successful message transmission from each node in the network in $\mathcal{O}(\Delta \log n)$ time slots, while the more coordinated approach used in the color reduction scheme achieves successful message transmission from a constant number of nodes in each broadcasting region to their neighbors in $\mathcal{O}(\log n)$ time slots.

The proof of the lemma is along the lines of the proof that local broadcasting can be achieved in $\mathcal{O}(\Delta \log n)$ time slots [55] and similar to bounding the interference in Section 3.3.

Proof of Lemma 4.1. Let us first consider the first part, which enables local broadcasting with constant success probability. Let us therefore prove two claims.

Claim A.2. *The probability $P_{none}^{A_v}$ that v is the only node in the proximity region that transmits a signal in the current time slot is at least $(1/4)^\chi$, and thus constant.*

Proof of the claim. Let us consider the probability that any other node in v's proximity region attempts to transmit. Let p_w be the current transmission probability of w. The probability that v is the only node in the proximity region that transmits a signal in the current time slot is at least

$$P_{\text{none}}^{A_v} \geq \prod_{w \in A_v \setminus \{v\}} (1 - p_w) \geq \left(\frac{1}{4}\right)^{\sum_{w \in A_v \setminus \{v\}} p_w} \geq \left(\frac{1}{4}\right)^{\sum_{\substack{u \in A_v \setminus \{v\}, \\ u \text{ independent}}} \sum_{w \in B_u} p_w}$$
$$\geq \left(\frac{1}{4}\right)^{\sum_{\substack{u \in A_v \setminus \{v\}, \\ u \text{ independent}}} 1} \geq \left(\frac{1}{4}\right)^{\chi},$$

where the second inequality holds due to Fact 3.1 in [55], the third inequality follows by covering the nodes in A_v by broadcasting ranges of independent nodes in A_v. The next inequality is implied by Lemma A.1, while the last inequality holds as χ is (roughly speaking) an upper bound on the number of independent nodes in A_v. The claim follows from observing that χ is indeed constant as the number of disks required to cover a (by a constant) larger disk is constant. □

As the proof of the following claim is in parts as in [55], we omit the bound on the interference received from nodes outside of the proximity area of v on nodes in the broadcasting range of v.

Claim A.3. *The probability P_{SINR}^{v} that the SINR constraint holds for a given transmission is at least $1/2$.*

Proof of the claim. The proof is based on the concept of rings around the transmitting node v. With increasing distance the number of nodes in a ring increases, however, also the effects on nodes in the broadcasting range of v decreases. Based

on Lemma A.1, the bound on the interference received at an arbitrary node w in v's broadcasting range can be bound by $\frac{P}{4\beta(r^B)^\alpha}$. Now, applying the Markov inequality it holds that the probability that the interference is more than twice this level (and thus the SINR constraint is violated) is at most $1/2$. It follows that the SINR constraint holds with probability at least $1/2$. □

By combining these probabilities with the transmission probability it follows that the probability that v successfully transmits a message to all neighbors in a given time slot is at least $p \cdot P_{\text{none}}^{A_v} \cdot P_{\text{SINR}}^{v}$. As we set $\lambda := \left(P_{\text{none}}^{A_v} \cdot P_{\text{SINR}}^{v}\right)^{-1}$, the probability for a successful transmission to all neighbors after $\kappa_0 = \frac{\lambda \ln 12}{p}$ time slots is at least

$$1 - \left(1 - \frac{p}{\lambda}\right)^{\frac{\lambda \ln 12}{p}} \geq 1 - e^{-\ln 12} \geq 1 - \frac{1}{12} \geq \frac{11}{12},$$

which implies the first part of Lemma 4.1.

Let us now consider the second part. Note that in both algorithms most nodes transmit with probability p_1 and only few, i.e., a constant number of nodes in each broadcasting range, are allowed to transmit with probability p_2 simultaneously. Having all nodes transmitting with p_1 for $\kappa_1 \in \mathcal{O}(\Delta \log n)$ time slots is essentially the well-known result on local broadcasting by Goussevskaia et al. [55]. The transmission probability p_2 is constant and selected such that the constant number of nodes required by the respective algorithms may use p_2 without violating Lemma A.1.

Similar to the argumentation before both claims hold and transmitting with probability p_ℓ for κ_ℓ time slots ($\ell = 1, 2$) yields a failure probability of at most

$$1 - \left(1 - \frac{p_\ell}{\lambda}\right)^{\kappa_\ell} \geq 1 - e^{c \ln n} \geq 1 - \frac{1}{n^c},$$

which completes the proof. □

A.3 EXPERIMENTS: OTHER DISTRIBUTIONS

In Chapter 5 we showed detailed data only for the random deployment strategy, for other deployments we restricted ourselves to show only the overall best results. In this section we present additional data obtained from the experiments described in the referenced chapter. This data justifies our selection of the parameters as used to obtain the overall best results.

Let us briefly describe the contents of each table. The overall setting is described in Section 5.3. In each table we report the average number of conflicts and the average runtime and mark the best or the selected combination as bold. In Table A.1 we consider variants of our phase-based (4Δ)-coloring algorithm RAND4DCOLORING, in which nodes simply select a new random color at the end of a phase if a conflict was detected during the phase. For COLORREDUCTION we need to determine the parameter FACTOR, the results for different values are given in Table A.2. For all deployment strategies we selected the same FACTOR of 0.6 to be an optimal balance

between number of conflicts and runtime. In Table A.3 we consider ColorReduction and its variant CRRandColor, which replaces the valid color of each node with a random color. We observe that, despite differences in the average runtime and the number of conflicts, the results are similar and as for 2Δ available colors good results are achieved for all deployments. For the algorithms MWColoring and YuColoring we show the results for various values of the parameter factor in Tables A.4 and A.5, respectively. We select factor $= 0.2$ as optimal value for both algorithms and all deployment strategies.

The results of our heuristic improvements of CRRandColor, MWColoring, and YuColoring for different values of duration′ are given in Tables A.6 and A.7. In these heuristics we reduce the number of conflicts by allowing nodes to reset to certain points in the algorithms once a conflict is detected. This increases the runtime for the standard duration, however, using a decreased duration′ we can decrease both the average runtime and the average number of conflicts. The values selected as optimal are marked bold and use a $1/16$ or $1/8$ fraction of duration as duration′.

Table A.1: Runtime and number of conflicts for RAND4DCOLORING and its variants.

Deployment	Algorithm	Runtime	Conflicts
Cluster	RAND4DCOLORING	3321	0.00
	RAND4DRESPECTCOLORING	15 639	0.00
	RAND4DFINALCOLORING	16 110	0.00
	RAND1DCOLORING	11 822	0.00
Cluster&Grid	RAND4DCOLORING	2186	0.00
	RAND4DRESPECTCOLORING	10 172	0.00
	RAND4DFINALCOLORING	10 456	0.00
	RAND1DCOLORING	8211	0.00
Cluster&PGrid	RAND4DCOLORING	2016	0.00
	RAND4DRESPECTCOLORING	9847	0.00
	RAND4DFINALCOLORING	10 199	0.00
	RAND1DCOLORING	6627	0.00
Cluster&Random	RAND4DCOLORING	2316	0.00
	RAND4DRESPECTCOLORING	10 175	0.00
	RAND4DFINALCOLORING	10 349	0.00
	RAND1DCOLORING	8153	0.00
Grid	RAND4DCOLORING	974	0.00
	RAND4DRESPECTCOLORING	4244	0.00
	RAND4DFINALCOLORING	4354	0.00
	RAND1DCOLORING	3358	0.00
PGrid	RAND4DCOLORING	1372	0.00
	RAND4DRESPECTCOLORING	6114	0.00
	RAND4DFINALCOLORING	6283	0.00
	RAND1DCOLORING	4548	0.00
Random	RAND4DCOLORING	1256	0.00
	RAND4DRESPECTCOLORING	5668	0.00
	RAND4DFINALCOLORING	5865	0.00
	RAND1DCOLORING	4174	0.00

Table A.2: Average number of conflicts and average runtime for ColorReduction using different parameters factor. We report the values for each deployment strategy.

factor	0.05	0.1	0.2	0.3	0.4	0.6	0.8
Cluster							
conflicts	0.00	0.04	0.02	0.17	0.18	**1.04**	2.74
runtime	902 421	468 046	240 713	163 781	126 039	**88094**	69 837
Cluster&Grid							
conflicts	0.00	0.00	0.06	0.17	0.20	**1.05**	3.15
runtime	590 431	297 464	151 579	103 595	80 001	**55760**	44 176
Cluster&PGrid							
conflicts	0.00	0.04	0.02	0.18	0.63	**3.32**	9.55
runtime	583 200	294 533	150 570	103 126	78 824	**55157**	43 619
Cluster&Random							
conflicts	0.00	0.04	0.04	0.04	0.30	**1.31**	4.31
runtime	578 623	292 450	150 245	102 685	78 946	**55005**	43 716
Grid							
conflicts	0.00	0.08	0.12	0.11	0.08	**0.30**	1.17
runtime	272 122	137 497	70 855	48 354	37 137	**25770**	20 258
PGrid							
conflicts	0.02	0.02	0.00	0.04	0.06	**0.14**	0.52
runtime	375 157	190 144	97 660	66 541	51 139	**35624**	27 964
Random							
conflicts	0.00	0.04	0.10	0.00	0.12	**0.51**	2.47
runtime	339 013	171 099	87 924	59 995	46 266	**32224**	25 384

Table A.3: Average runtime for ColorReduction and CRRandColor for colorings of different sizes. The runtimes are almost identical although CRRandColor uses only a random color to replace the valid coloring used in ColorReduction.

Number of colors		$\Delta+1$	2Δ	3Δ	4Δ
Cluster					
ColorReduction	conflicts	4.99	0.88	0.88	0.64
	runtime	51 476	67 624	88 101	108 721
CRRandColor	conflicts	13.19	**1.05**	0.76	0.67
	runtime	53 566	**67881**	88 529	109 178
Cluster&Grid					
ColorReduction	conflicts	2.56	1.08	0.98	1.31
	runtime	30 861	42 682	55 536	69 465
CRRandColor	conflicts	5.67	**1.03**	0.63	0.95
	runtime	31 713	**42692**	55 599	68 964
Cluster&PGrid					
ColorReduction	conflicts	6.00	3.45	3.32	3.49
	runtime	32 024	42 094	55 148	68 682
CRRandColor	conflicts	10.62	**3.23**	3.77	3.67
	runtime	32 913	**42385**	55 091	68 015
Cluster&Random					
ColorReduction	conflicts	2.63	1.53	1.34	1.17
	runtime	30 337	42 466	54 884	68 399
CRRandColor	conflicts	5.94	**1.83**	1.12	1.24
	runtime	30 939	**42034**	55 113	68 619
Grid					
ColorReduction	conflicts	9.22	0.42	0.42	0.20
	runtime	16 128	20 310	25 929	31 297
CRRandColor	conflicts	31.01	**0.82**	0.42	0.32
	runtime	17 555	**20367**	25 798	31 257
PGrid					
ColorReduction	conflicts	1.54	0.18	0.10	0.04
	runtime	20 367	27 669	35 683	43 631
CRRandColor	conflicts	5.31	**0.10**	0.10	0.15
	runtime	20 996	**27817**	35 682	43 633
Random					
ColorReduction	conflicts	3.29	0.86	0.55	0.61
	runtime	18 638	24 824	32 197	39 737
CRRandColor	conflicts	6.52	**0.93**	0.65	0.63
	runtime	19 766	**24758**	32 287	39 696

Table A.4: Average number of conflicts and average runtime for MWColoring using different parameters factor. We report the values for each deployment strategy.

factor		0.05	0.1	0.2	0.3	0.4	0.6
Cluster	conflicts	0.14	0.24	**0.56**	1.22	2.03	3.66
	runtime	197 790	111 416	**73670**	67 658	65 895	64 854
Cluster&Grid	conflicts	0.02	0.20	**0.34**	0.66	1.66	3.19
	runtime	130 064	73 089	**47041**	42 956	41 848	40 854
Cluster&PGrid	conflicts	0.16	0.28	**1.06**	1.47	2.9	7.89
	runtime	129 632	70 384	**46142**	41 730	40 823	38 883
Cluster&Random	conflicts	0.18	0.16	**0.44**	0.84	1.33	3.78
	runtime	129 266	74 177	**46363**	41 599	39 944	39 601
Grid	conflicts	0.02	0.06	**0.26**	1.16	2.04	4.1
	runtime	76 474	41 798	**25456**	20 680	18 918	17 221
PGrid	conflicts	0.04	0.06	**0.28**	0.48	0.84	1.14
	runtime	95 826	53 807	**32812**	27 421	25 652	23 765
Random	conflicts	0.10	0.12	**0.42**	1.02	1.48	2.71
	runtime	81 195	44 700	**27982**	23 995	22 807	21 870

Table A.5: Average number of conflicts and average runtime for YuColoring using different parameters factor. We report the values for each deployment strategy. C=Cluster, G=Grid, PG=PGrid, R=Random

factor		0.05	0.1	0.2	0.3	0.4	0.6
C	conflicts	0.42	0.57	**0.9**	1.55	3.84	13.61
	runtime	382 860	233 591	**164839**	145 760	135 567	137 233
C&G	conflicts	0.5	0.52	**0.94**	2.19	4.2	15.68
	runtime	298 736	189 883	**141003**	127 842	127 151	120 294
C&PG	conflicts	0.56	1.20	**1.30**	3.56	9.03	36.42
	runtime	286 326	175 088	**126488**	112 753	107 833	109 203
C&R	conflicts	0.48	0.58	**0.88**	2.52	6.05	23.38
	runtime	277 447	172 869	**122267**	110 815	104 874	106 367
Grid	conflicts	1.04	1.06	**2.01**	3.7	6.16	20.48
	runtime	379 283	195 925	**113105**	86 592	73 141	60 833
PGrid	conflicts	1.00	0.96	**1.12**	1.83	3.04	10.35
	runtime	402 421	214 349	**129054**	100 451	88 521	76 147
Random	conflicts	0.62	0.71	**1.39**	2.90	6.41	22.43
	runtime	286 167	160 088	**99946**	82 707	72 660	67 131

Table A.6: Average number of conflicts and average runtime for the correcting variants CRRCorrecting, MWCorrecting, and YuCorrecting for different values of duration'. In this table: Deployments involving the cluster deployment

Fraction of duration		1/32	1/16	1/8	1/4	1/2	1
Resulting duration'		143	287	575	1150	2300	4600
Cluster							
CRRCorrecting	conflicts	0.00	0.00	**0.00**	0.00	0.00	0.00
	runtime	24 425	17 860	**17802**	25 023	40 185	68 493
MWCorrecting	conflicts	0.63	0.94	**0.45**	0.10	0.00	0.00
	runtime	30 931	21 637	**23531**	33 245	50 780	105 200
YuCorrecting	conflicts	3.13	**0.74**	0.14	0.02	0.00	0.00
	runtime	12 843	**20849**	31 481	49 630	92 327	176 551
Cluster&Grid							
CRRCorrecting	conflicts	0.00	0.00	**0.00**	0.00	0.02	0.00
	runtime	17 008	12 473	**11333**	15 642	24 069	43 139
MWCorrecting	conflicts	0.32	0.59	**0.36**	0.04	0.00	0.00
	runtime	20 963	14 189	**13535**	18 601	30 008	54 896
YuCorrecting	conflicts	3.13	**1.23**	0.22	0.02	0.02	0.00
	runtime	7615	**15835**	25 747	42 543	78 202	154 572
Cluster&PGrid							
CRRCorrecting	conflicts	0.00	**0.02**	0.02	0.00	0.00	0.00
	runtime	13 609	**10896**	11 133	15 440	23 863	42 658
MWCorrecting	conflicts	0.38	**0.42**	0.41	0.02	0.02	0.00
	runtime	19 031	**13353**	13 402	17 490	30 973	55 544
YuCorrecting	conflicts	3.11	**0.64**	0.17	0.02	0.00	0.00
	runtime	7681	**14620**	22 982	38 887	71 253	139 299
Cluster&Random							
CRRCorrecting	conflicts	0.00	0.00	**0.00**	0.00	0.00	0.00
	runtime	16 903	12 789	**11884**	16 009	24 272	42 773
MWCorrecting	conflicts	0.23	0.33	**0.44**	0.04	0.00	0.00
	runtime	22 256	14 631	**12780**	17 341	28 919	53 017
YuCorrecting	conflicts	3.74	**0.81**	0.20	0.08	0.00	0.00
	runtime	7391	**15060**	24 173	38 689	69 636	135 511

Table A.7: Average number of conflicts and average runtime for the correcting variants CRRCorrecting, MWCorrecting, and YuCorrecting for different values of duration'. In this table: Grid, PGrid and Random deployment

Fraction of duration		1/32	1/16	1/8	1/4	1/2	1
Resulting duration'		143	287	575	1150	2300	4600
Grid							
CRRCorrecting	conflicts	0.00	0.00	**0.00**	0.00	0.00	0.00
	runtime	7538	5343	**5113**	7013	11 839	20 756
MWCorrecting	conflicts	0.04	**0.12**	0.21	0.02	0.02	0.00
	runtime	8300	**5740**	5741	8287	15 636	27 958
YuColoring	conflicts	5.86	**1.77**	0.26	0.09	0.02	0.04
	runtime	3625	**8654**	16 819	33 089	68 253	135 904
PGrid							
CRRCorrecting	conflicts	0.00	0.00	**0.00**	0.00	0.00	0.00
	runtime	10 804	7904	**7017**	9638	15 267	27 889
MWCorrecting	conflicts	0.08	0.12	**0.12**	0.06	0.00	0.00
	runtime	11 831	7925	**7567**	10 558	18 263	35 860
YuCorrecting	conflicts	7.09	**2.15**	0.63	0.14	0.07	0.02
	runtime	4770	**11479**	20 947	38 641	77 174	155 428
Random							
CRRCorrecting	conflicts	0.00	0.00	**0.00**	0.00	0.00	0.00
	runtime	8965	6984	**6489**	8883	14 348	25 218
MWCorrecting	conflicts	0.13	0.23	**0.10**	0.02	0.02	0.00
	runtime	11 065	7688	**6834**	9105	15 762	31 027
YuCorrecting	conflicts	4.62	**1.23**	0.28	0.09	0.00	0.00
	runtime	4370	**9807**	16 652	29 635	58 189	116 646

A.4 FCDS-ALGORITHM: NETWORK AND COMPONENTS

In this section we present additional material deferred from Chapter 7. We reference two results required for our analysis, before we briefly describe the protocol we use to identify connected components.

Enough short connector paths are sufficient

We state the lemma without a formal proof, which is given as part of the proof of Lemma 4.4 in [21].

Lemma A.4 (part of Lemma 4.4 in [21])**.** *Given a class i and a component $\mathcal{C}$ of layer $l > L \log n$ with at least $k/2$ short connector paths, $\mathcal{C}$ has at least one short connector path of class I with probability at least δ.*

Number of connector paths for type-2 node

Lemma A.5 (Proposition 4.2 in [20])**.** *For an arbitrary class i and a type-2 node v, v lies on at most one long connector path of one component of $\mathcal{C}$.*

Which implies the following Observation.

Observation A.6. *Each type-2 node lies on at most $t \in \Theta(k)$ long connector paths.*

Identifying Connected Components

To identifying and communicate through connected component, we use the protocol described in [21, Theorem B.2]. There are two protocols that can be used, depending on the maximum diameter D' of the components in the virtual graph maximum diameter D' of the components, which is in $\mathcal{O}(\frac{n \log n}{k})$ whp. If it is relatively small, i.e. $\frac{n \log n}{k} = o(D + \sqrt{n \log n} \log^* n)$, a simple protocol can be used, while a variation of a protocol to identify connector components by Thurimella [124] is used otherwise.

Let us now consider the simpler variant. Each node transmits its class, and the smallest node ID it received so far (including its own). Nodes discard received IDs if they are transmitted by nodes with different classes. After $D' = \mathcal{O}(\frac{n \log n}{k})$ rounds, each node in each component received the smallest ID of the component, which is selected as the component ID and the components root node. The union of paths from the root to nodes of the components can be used as communication tree in the component.

The more complex protocol, which is a variation of the algorithm to identify connected components by Thurimella [124] is originally based on an minimum spanning tree (MST) algorithm by Garay, Kutten and Peleg [51], which was improved to the current runtime bound by a new MST algorithm in [93]. The protocol allows each node in a network to learn the smallest ID in its component in $\mathcal{O}(D + \sqrt{n} \log^* n)$ rounds. The ID of each virtual node v_l (of layer l) is set to (ID_v, l, type), where ID_v is the ID of the corresponding real node, l the virtual nodes layer, and type its type (either 1 or 2). The algorithm by Thurimella

is executed on $\mathcal{G}$, which has a diameter in $\mathcal{O}(D)$, and $\mathcal{O}(n \log n)$ nodes, resulting in $\mathcal{O}(D + \sqrt{n \log n} \log^* n)$ rounds for identifying the connected components.

BIBLIOGRAPHY

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. "Wireless sensor networks: a survey." In: *Computer Networks* 38.4 (2002), pp. 393–422. DOI: 10.1016/S1389-1286(01)00302-4 (cit. on pp. 1, 2).

[2] Cristina Alcaraz, Pablo Najera, Javier Lopez, and Rodrigo Roman. "Wireless sensor networks and the internet of things: Do we need a complete integration?" In: *Proc. 1st Internat. Workshop on the Security of the Internet of Things (SecIoT'10)*. 2010 (cit. on p. 2).

[3] Joan M. Aldous and Robin J. Wilson. *Graphs and applications: an introductory approach*. Vol. 1. Springer Science & Business Media, 2000 (cit. on p. 120).

[4] Noga Alon, Amotz Bar-Noy, Nathan Linial, and David Peleg. "On the Complexity of Radio Communication." In: *Proc. 21th Ann. ACM Symp. Theory Comput. (STOC'89)*. ACM Press, 1989, pp. 274–285. DOI: 10.1145/73007.73033 (cit. on pp. 102, 110, 111).

[5] Robert B. Ash. *Basic probability theory*. Dover Publications, 2008 (cit. on p. 7).

[6] Chen Avin, Asaf Cohen, Yoram Haddad, Erez Kantor, Zvi Lotker, Merav Parter, and David Peleg. "SINR diagram with interference cancellation." In: *Proc. 23th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA'12)*. SIAM. 2012, pp. 502–515 (cit. on p. 11).

[7] Chen Avin, Yuval Emek, Erez Kantor, Zvi Lotker, David Peleg, and Liam Roditty. "SINR diagrams: Convexity and its applications in wireless networks." In: *Journal of the ACM* 59.4 (2012), 18:1–18:34. DOI: 10.1145/2339123.2339125 (cit. on p. 11).

[8] Chen Avin, Zvi Lotker, Francesco Pasquale, and Yvonne Anne Pignolet. "A note on uniform power connectivity in the physical signal to interference plus noise (SINR) model." In: *Theor. Comput. Sci.* 453 (2012), pp. 2–13. DOI: 10.1016/j.tcs.2011.12.066 (cit. on p. 11).

[9] Baruch Awerbuch, Michael Luby, Andrew V. Goldberg, and Serge A. Plotkin. "Network decomposition and locality in distributed computation." In: *Proc. 30th Ann. IEEE Symp. Foundations Comput. Sci. (FOCS'89)*. IEEE. 1989, pp. 364–369. DOI: 10.1109/SFCS.1989.63504 (cit. on p. 51).

[10] F. Bai and A. Helmy. "A Survey of Mobility Modeling and Analysis in Wireless Adhoc Networks." In: *Wireless Ad Hoc and Sensor Networks*. Springer, 2006 (cit. on p. 15).

[11] Joe Bardwell. "Converting signal strength percentage to dBm values." In: *WildPackets' White Paper* (2002) (cit. on p. 53).

[12] Leonid Barenboim. "Deterministic ($\Delta + 1$)-Coloring in Sublinear (in Δ) Time in Static, Dynamic and Faulty Networks." In: *Proc. 2015 ACM Symp. on Principles of Distributed Computing (PODC'15)*. PODC '15. ACM, 2015, pp. 345–354. DOI: 10.1145/2767386.2767410 (cit. on p. 51).

[13] Leonid Barenboim and Michael Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2013 (cit. on pp. 50, 51, 54, 59).

[14] Leonid Barenboim, Michael Elkin, and Fabian Kuhn. "Distributed $(\Delta + 1)$-Coloring in Linear (in Δ) Time." In: *SIAM J. Comput.* 43.1 (2014), pp. 72–95. DOI: 10.1137/12088848X (cit. on p. 51).

[15] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. "The locality of distributed symmetry breaking." In: *Proc. 53rd Ann. IEEE Symp. Foundations Comput. Sci. (FOCS'12)*. IEEE. 2012, pp. 321–330. DOI: 10.1109/FOCS.2012.60 (cit. on p. 51).

[16] Leonid Barenboim and David Peleg. "Nearly Optimal Local Broadcasting in the SINR Model with Feedback." In: *Proc. 22nd Internat. Colloq. Structural Inform. and Communication Complexity (SIROCCO'15)*. Lecture Notes Comput. Sci. Springer, 2015, pp. 164–178. DOI: 10.1007/978-3-319-25258-2_12 (cit. on pp. 11, 12, 23, 102, 103, 104, 107).

[17] Reuven Bar-Yehuda, Amos Israeli, and Alon Itai. "Multiple communication in multihop radio networks." In: *SIAM J. Comput.* 22.4 (1993), pp. 875–887. DOI: 10.1137/0222055 (cit. on p. 117).

[18] Marijke H. L. Bodlaender and Magnús M. Halldórsson. "Beyond geometry: towards fully realistic wireless models." In: *Proc. 2014 ACM Symp. on Principles of Distributed Computing (PODC'14)*. 2014, pp. 347–356. DOI: 10.1145/2611462.2611476 (cit. on p. 12).

[19] Marijke H. L. Bodlaender, Magnús M. Halldórsson, and Pradipta Mitra. "Connectivity and aggregation in multihop wireless networks." In: *Proc. 2013 ACM Symp. on Principles of Distributed Computing (PODC'13)*. 2013, pp. 355–364. DOI: 10.1145/2484239.2484265 (cit. on p. 12).

[20] Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. "A New Perspective on Vertex Connectivity." In: *Proc. 25th. Ann. ACM-SIAM Symp. Discrete Algorithms (SODA'14)*. 2014, pp. 546–561. DOI: 10.1137/1.9781611973402.41 (cit. on pp. 115, 116, 117, 118, 119, 122, 129, 147).

[21] Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. "Distributed Connectivity Decomposition." In: *Proc. 2014 ACM Symp. on Principles of Distributed Computing (PODC'14)*. ACM, 2014, pp. 156–165. DOI: 10.1145/2611462.2611491 (cit. on pp. 115, 116, 117, 119, 120, 121, 122, 125, 128, 129, 147).

[22] Dan Chen, Zhixin Liu, Lizhe Wang, Minggang Dou, Jingying Chen, and Hui Li. "Natural Disaster Monitoring with Wireless Sensor Networks: A Case Study of Data-intensive Applications upon Low-Cost Scalable Systems." In: *Mobile Netw. Appl.* 18.5 (2013), pp. 651–663. DOI: 10.1007/s11036-013-0456-9 (cit. on p. 1).

[23] Imrich Chlamtac and Shay Kutten. "On Broadcasting in Radio Networks–Problem Analysis and Protocol Design." In: *Communications, IEEE Transactions on* 33.12 (12/1985), pp. 1240–1246. DOI: 10.1109/TCOM.1985.1096245 (cit. on p. 9).

[24] Brent N. Clark, Charles J. Colbourn, and David S. Johnson. "Unit disk graphs." In: *Discrete Mathematics* 86.1 (1990), pp. 165–177. DOI: 10.1016/0012-365X(90)90358-O (cit. on p. 9).

[25] Richard Cole and Uzi Vishkin. "Deterministic Coin Tossing and Accelerating Cascades: Micro and Macro Techniques for Designing Parallel Algorithms." In: *Proc. 18th Ann. ACM Symp. Theory Comput. (STOC'86)*. ACM, 1986, pp. 206–219. DOI: 10.1145/12130.12151 (cit. on p. 51).

[26] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3rd ed.)* MIT Press, 2009 (cit. on p. 7).

[27] Bevan Das and Vaduvur Bharghavan. "Routing in ad-hoc networks using minimum connected dominating sets." In: *Proc. Internat. Cartographic Conf. (ICC'97)*. Vol. 1. IEEE Comp. Soc. 1997, pp. 376–380 (cit. on p. 117).

[28] Sebastian Daum, Seth Gilbert, Fabian Kuhn, and Calvin Newport. "Broadcast in the Ad Hoc SINR Model." In: *Proc. 27th Internat. Symp. on Distributed Computing (DISC'13)*. Vol. 8205. Lecture Notes Comput. Sci. Springer, 2013, pp. 358–372. DOI: 10.1007/978-3-642-41527-2_25 (cit. on p. 12).

[29] Bilel Derbel, Mohamed Mosbah, and Akka Zemmari. "Fast Distributed Graph Partition and Application (Extended Abstract)." In: *Proc. 20th Internat. Parallel and Distributed Processing Symp. (IPDPS'06)*. 2006. DOI: 10.1109/IPDPS.2006.1639362 (cit. on p. 102).

[30] Bilel Derbel and El-Ghazali Talbi. "Distributed Node Coloring in the SINR Model." In: *Proc. 30th Internat. Conf. on Distributed Computing Systems (ICDCS'10)*. IEEE Computer Society, 2010, pp. 708–717. DOI: 10.1109/ICDCS.2010.35 (cit. on pp. 12, 22, 23, 34, 35, 36, 37, 42, 43, 44, 50, 51, 52, 54, 62, 63, 66, 69, 70, 74, 102, 103, 104, 107).

[31] Reinhard Diestel. *Graph Theory (4th ed.)* Vol. 173. Graduate texts in mathematics. Springer, 2012 (cit. on p. 7).

[32] Michael Dinitz. "Distributed Algorithms for Approximating Wireless Network Capacity." In: *Proc. 29th. IEEE Internat. Conference on Comput. Comm. (INFOCOM'10)*. INFOCOM'10. IEEE Press, 2010, pp. 1397–1405. DOI: 10.1109/INFCOM.2010.5461905 (cit. on p. 11).

[33] Distributed Computing Group, ETH Zurich. *Sinalgo - Simulator for Network Algorithms*. version 0.75.3. 2008. URL: http://sourceforge.net/projects/sinalgo/ (cit. on pp. 12, 32, 69, 79, 96, 135).

[34] Alina Ene, Nitish Korula, and Ali Vakilian. "Improved Approximation Algorithms for Connected Domatic Partitions and Related Problems." In: *CoRR* abs/1305.4308 (2013). URL: http://arxiv.org/abs/1305.4308 (cit. on p. 117).

[35] Alexander Fanghänel, Thomas Kesselheim, Harald Räcke, and Berthold Vöcking. "Oblivious interference scheduling." In: *Proc. 28th ACM Symp. on Principles of Distributed Computing (PODC'09)*. ACM. 2009, pp. 220–229. DOI: 10.1145/1582716.1582752 (cit. on p. 11).

[36] Muhammad O. Farooq and Thomas Kunz. "Wireless sensor networks testbeds and state-of-the-art multimedia sensor nodes." In: *Appl. Math. Inf. Sci* 8 (2014), pp. 935–940 (cit. on p. 134).

[37] Uriel Feige, Magnús M. Halldórsson, Guy Kortsarz, and Aravind Srinivasan. "Approximating the Domatic Number." In: *SIAM J. Comput.* 32.1 (2002), pp. 172–195. DOI: 10.1137/S0097539700380754 (cit. on p. 117).

[38] Irene Finocchi, Alessandro Panconesi, and Riccardo Silvestri. "An experimental analysis of simple, distributed vertex coloring algorithms." In: *Algorithmica* 41.1 (2004), pp. 1–23. DOI: 10.1007/s00453-004-1104-3 (cit. on pp. 51, 70).

[39] Christina Fragouli and Emina Soljanin. "Network Coding Fundamentals." In: *Foundations and Trends in Networking* 2.1 (2007). DOI: 10.1561/1300000003 (cit. on p. 117).

[40] Fabian Fuchs. "Brief Announcement: Fast and Simple Node Coloring in the SINR Model." In: *Proc. 2015 ACM Symp. on Principles of Distributed Computing (PODC'15)*. ACM, 2015. DOI: 10.1145/2767386.2767445 (cit. on pp. 4, 49).

[41] Fabian Fuchs. "Experimental Evaluation of Distributed Node Coloring Algorithms for Wireless Networks." In: *Proc. 18th Meeting on Algorithm Engineering and Experiments (ALENEX'16)*. SIAM, 2016, pp. 30–38. DOI: 10.1137/1.9781611974317.3 (cit. on p. 69).

[42] Fabian Fuchs. "On Asynchronous Node Coloring in the SINR Model." unpublished manuscript. 2015. URL: http://i11www.iti.kit.edu/f-oancs-15.pdf (cit. on p. 58).

[44] Fabian Fuchs and Roman Prutkin. "Simple Distributed $\Delta + 1$ Coloring in the SINR Model." In: *Proc. 22nd Internat. Colloq. Structural Inform. and Communication Complexity (SIROCCO'15)*. Lecture Notes Comput. Sci. Springer, 2015, pp. 149–163. DOI: 10.1007/978-3-319-25258-2_11 (cit. on pp. 4, 49).

[43] Fabian Fuchs and Roman Prutkin. "Simple Distributed Delta+1 Coloring in the SINR Model." In: *CoRR* abs/1502.02426 (2015). URL: http://arxiv.org/abs/1502.02426 (cit. on p. 49).

[45] Fabian Fuchs and Dorothea Wagner. "Arbitrary Transmission Power in the SINR Model: Local Broadcasting, Coloring and MIS." In: *CoRR* abs/1402.4994 (2014). URL: http://arxiv.org/abs/1502.02426 (cit. on p. 21).

[46] Fabian Fuchs and Dorothea Wagner. "Local Broadcasting with Arbitrary Transmission Power in the SINR Model." In: *Proc. 21st Internat. Colloq. Structural Inform. and Communication Complexity (SIROCCO'14)*. Vol. 8576. Lecture Notes Comput. Sci. Springer, 2014, pp. 180–193. DOI: 10.1007/978-3-319-09620-9_15 (cit. on pp. 4, 21).

[47] Fabian Fuchs and Dorothea Wagner. "On Local Broadcasting Schedules and CONGEST Algorithms in the SINR Model." In: *Proc. 9th Internat. Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS'13)*. Vol. 8243. Lecture Notes in Computer Science. Springer, 2013, pp. 170–184. DOI: 10.1007/978-3-642-45346-5_13 (cit. on pp. 5, 101).

[48] Fabian Fuchs and Matthias Wolf. "On the Distributed Computation of Fractional Connected Dominated Set Packings." In: *CoRR* abs/1508.04278 (2015). under review. URL: http://arxiv.org/abs/1508.04278 (cit. on p. 115).

[49] Takahito Fujii, Michito Takahashi, Masaki Bandai, Tomoyuki Udagawa, and Iwao Sasase. "An efficient MAC protocol in wireless ad-hoc networks with heterogeneous power nodes." In: *Proc. 5th Internat. Symp. Wireless Personal Multimedia Communications (WPMC'02)*. Vol. 2. IEEE. 2002, pp. 776–780 (cit. on p. 22).

[50] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. "Timing-sync protocol for sensor networks." In: *Proc. 1st Internat. Conf. on Embedded Networked Sensor Systems (SenSys'03)*. ACM. 2003, pp. 138–149. DOI: 10.1145/958491.958508 (cit. on p. 67).

[51] Juan A. Garay, Shay Kutten, and David Peleg. "A sublinear time distributed algorithm for minimum-weight spanning trees." In: *SIAM J. Comput.* 27.1 (1998), pp. 302–316. DOI: 10.1137/S0097539794261118 (cit. on p. 147).

[52] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979 (cit. on p. 50).

[53] Stephen M. George, Wei Zhou, Harshavardhan Chenji, Myounggyu Won, Yong Oh Lee, Andria Pazarloglou, Radu Stoleru, and Prabir Barooah. "DistressNet: a wireless ad hoc and sensor network architecture for situation management in disaster response." In: *IEEE Communications Magazine* 48.3 (2010), pp. 128–136. DOI: 10.1109/MCOM.2010.5434384 (cit. on p. 1).

[54] Olga Goussevskaia, Magnús M. Halldórsson, and Roger Wattenhofer. "Algorithms for Wireless Capacity." In: *IEEE/ACM Trans. Netw.* 22.3 (2014), pp. 745–755. DOI: 10.1109/TNET.2013.2258036 (cit. on p. 11).

[55] Olga Goussevskaia, Thomas Moscibroda, and Roger Wattenhofer. "Local Broadcasting in the Physical Interference Model." In: *Proc. 5th ACM Inernat. Workshop on Foundations of Mobile Computing (DialM-POMC'08)*. ACM Press, 2008, pp. 35–44. DOI: 10.1145/1400863.1400873 (cit. on pp. 10, 11, 15, 22, 23, 25, 27, 29, 31, 45, 49, 50, 102, 103, 104, 138, 139).

[56] Olga Goussevskaia, Yvonne Anne Oswald, and Rogert Wattenhofer. "Complexity in geometric SINR." In: *Proc. 8th ACM Internat. Symp. on Mobile Ad Hoc Networking and Computing (MobiHoc'07)*. ACM. 2007, pp. 100–109. DOI: 10.1145/1288107.1288122 (cit. on p. 11).

[57] Olga Goussevskaia, Yvonne Anne Pignolet, and Roger Wattenhofer. "Efficiency of Wireless Networks: Approximation Algorithms for the Physical Interference Model." In: *Foundations and Trends in Networking* 4.3 (11/2010). DOI: 10.1561/1300000019 (cit. on p. 11).

[58] Olga Goussevskaia, Roger Wattenhofer, Magnús M. Halldórsson, and Emo Welzl. "Capacity of Arbitrary Wireless Networks." In: *Proc. 28th IEEE Internat. Conference on Comput. Comm. (INFOCOM'09)*. 2009, pp. 1872–1880. DOI: 10.1109/INFCOM.2009.5062108 (cit. on p. 11).

[59] Sudipto Guha and Samir Khuller. "Approximation algorithms for connected dominating sets." In: *Algorithmica* 20.4 (1998), pp. 374–387. DOI: 10.1007/PL00009201 (cit. on p. 117).

[60] Piyush Gupta and Panganmala R. Kumar. "The capacity of wireless networks." In: *IEEE Trans. on Inform. Theory* 46.2 (2000), pp. 388–404. DOI: 10.1109/18.825799 (cit. on pp. 2, 9, 11, 51).

[61] William K. Hale. "Frequency assignment: Theory and applications." In: *Proceedings of the IEEE* 68.12 (1980), pp. 1497–1514 (cit. on p. 4).

[62] Magnús M. Halldórsson. "Wireless scheduling with power control." In: *ACM Transactions on Algorithms (TALG)* 9.1 (2012), 7:1–7:20. DOI: 10.1145/2390176.2390183 (cit. on p. 11).

[63] Magnús M. Halldórsson, Stephan Holzer, and Nancy A. Lynch. "A Local Broadcast Layer for the SINR Network Model." In: *Proc. 2015 ACM Symp. on Principles of Distributed Computing (PODC'15)*. 2015, pp. 129–138. DOI: 10.1145/2767386.2767432 (cit. on pp. 12, 103).

[64] Magnús M. Halldórsson and Pradipta Mitra. "Nearly optimal bounds for distributed wireless scheduling in the SINR model." In: *Proc. 38th Internat. Colloquium on Automata, Languages, and Programming (ICALP'11)*. Springer, 2011, pp. 625–636. DOI: 10.1007/978-3-642-22012-8_50 (cit. on p. 11).

[65] Magnús M. Halldórsson and Pradipta Mitra. "Towards Tight Bounds for Local Broadcasting." In: *Proc. 8th ACM Inernat. Workshop on Foundations of Mobile Computing (FOMC'12)*. ACM Press, 07/2012. DOI: 10.1145/2335470.2335472 (cit. on pp. 11, 22, 23, 27, 29, 30, 31, 49, 67, 103, 104).

[66] Magnús M. Halldórsson and Pradipta Mitra. "Wireless capacity and admission control in cognitive radio." In: *Proc. 28th IEEE Internat. Conference on Comput. Comm. (INFOCOM'09)*. IEEE. 2012, pp. 855–863. DOI: 10.1109/INFCOM.2012.6195834 (cit. on p. 11).

[67] Magnús M. Halldórsson and Pradipta Mitra. "Wireless capacity with arbitrary gain matrix." In: *Theor. Comput. Sci.* 553 (2014), pp. 57–63. DOI: 10.1016/j.tcs.2013.09.035 (cit. on p. 11).

[68] Magnús M. Halldórsson and Pradipta Mitra. "Wireless Connectivity and Capacity." In: *Proc. 23th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA'12)*. SODA '12. SIAM, 2012, pp. 516–526 (cit. on pp. 11, 22).

[69] Magnús M. Halldórsson and Tigran Tonoyan. "The Price of Local Power Control in Wireless Scheduling." In: *CoRR* abs/1502.05279 (2015). URL: http://arxiv.org/abs/1502.05279 (cit. on p. 11).

[70] Magnús M. Halldórsson, Yuexuan Wang, and Dongxiao Yu. "Leveraging Multiple Channels in Ad Hoc Networks." In: *Proc. 2015 ACM Symp. on Principles of Distributed Computing (PODC'15)*. 2015, pp. 431–440. DOI: 10.1145/2767386.2767437 (cit. on p. 51).

[71] Nathaniel Hobbs, Yuexuan Wang, Qiang-Sheng Hua, Dongxiao Yu, and Francis C. M. Lau. "Deterministic Distributed Data Aggregation under the SINR Model." In: *Proc. 9th. Ann. Conf. Theory and Applic. of Models of Comp. (TAMC'12)*. Vol. 7287. Lecture Notes Comput. Sci. Springer, 2012, pp. 385–399. DOI: 10.1007/978-3-642-29952-0_38 (cit. on p. 12).

[72] Stephan Holzer and Roger Wattenhofer. "Optimal Distributed All Pairs Shortest Paths and Applications." In: *Proc. 31th ACM Symp. on Principles of Distributed Computing (PODC'12)*. ACM Press, 2012, pp. 355–364. DOI: 10.1145/2332432.2332504 (cit. on p. 102).

[73] Thomas Moscibroda and Roger Wattenhofer. "The Complexity of Connectivity in Wireless Networks." In: *Proc. 25th IEEE Internat. Conference on Comput. Comm. (INFOCOM'06)*. IEEE Computer Society Press, 04/2006, pp. 1–13. DOI: 10.1109/INFOCOM.2006.23 (cit. on p. 11).

[74] Amos Israeli and A. Itai. "A Fast and Simple Randomized Parallel Algorithm for Maximal Matching." In: *Inform. Process. Lett.* 22.2 (02/1986), pp. 77–80. DOI: 10.1016/0020-0190(86)90144-4 (cit. on pp. 125, 126, 127).

[75] Thomas Janson and Christian Schindelhauer. "Analyzing randomly placed multiple antennas for MIMO wireless communication." In: *Proc. 8th IEEE Internat. Conf. on Wireless and Mobile Computing, Networking and Communications (WiMob'12)*. IEEE. 2012, pp. 745–752. DOI: 10.1109/WiMOB.2012.6379159 (cit. on p. 12).

[76] Thomas Janson and Christian Schindelhauer. "Broadcasting in Logarithmic Time for Ad Hoc Network Nodes on a Line Using Mimo." In: *Proc. 25th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA'13)*. SPAA '13. ACM, 2013, pp. 63–72. DOI: 10.1145/2486159.2486190 (cit. on p. 12).

[77] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. "Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with ZebraNet." In: *ACM Sigplan Notices* 37.10 (2002), pp. 96–107 (cit. on p. 2).

[78] Tomasz Jurdzinski and Dariusz R. Kowalski. "Distributed Backbone Structure for Algorithms in the SINR Model of Wireless Networks." In: *Proc. 26th Internat. Symp. on Distributed Computing (DISC'12)*. Vol. 7611. Lecture Notes Comput. Sci. Springer, 10/2012, pp. 106–120. DOI: 10.1007/978-3-642-33651-5_8 (cit. on pp. 12, 102, 103, 104, 106, 107, 108).

[79] Tomasz Jurdzinski, Dariusz R. Kowalski, Michal Rozanski, and Grzegorz Stachowiak. "Distributed Randomized Broadcasting in Wireless Networks under the SINR Model." In: *Proc. 27th Internat. Symp. on Distributed Computing (DISC'13)*. Vol. 8205. Lecture Notes Comput. Sci. Springer, 2013, pp. 373–387. DOI: 10.1007/978-3-642-41527-2_26 (cit. on pp. 12, 134).

[80] Tomasz Jurdzinski, Dariusz R. Kowalski, Michal Rozanski, and Grzegorz Stachowiak. "On the impact of geometry on ad hoc communication in wireless networks." In: *Proc. 2014 ACM Symp. on Principles of Distributed Computing (PODC'14)*. 2014, pp. 357–366. DOI: 10.1145/2611462.2611487 (cit. on p. 12).

[81] Tomasz Jurdzinski, Dariusz R. Kowalski, and Grzegorz Stachowiak. "Distributed Deterministic Broadcasting in Uniform-Power Ad Hoc Wireless Networks." English. In: *Proc. 19th Internat. Symp. on Fundamentals of Computation Theory (FCT'13)*. Vol. 8070. Lecture Notes Comput. Sci. Springer, 2013, pp. 195–209. DOI: 10.1007/978-3-642-40164-0_20 (cit. on p. 12).

[82] Tomasz Jurdziński and Grzegorz Stachowiak. "Probabilistic algorithms for the wakeup problem in single-hop radio networks." In: *Proc. 13th. Internat. Symp. on Algorithms and Computation (ISAAC'02)*. Springer, 2002, pp. 535–549. DOI: 10.1007/3-540-36136-7_47 (cit. on p. 26).

[83] Erez Kantor, Zvi Lotker, Merav Parter, and David Peleg. "The topology of wireless communication." In: *Proc. 43th Ann. ACM Symp. Theory Comput. (STOC'11)*. ACM. 2011, pp. 383–392. DOI: 10.1145/1993636.1993688 (cit. on p. 11).

[84] Elliott Kaplan and Christopher Hegarty. *Understanding GPS: principles and applications*. Artech house, 2005 (cit. on p. 105).

[85] Richard M. Karp. "Reducibility Among Combinatorial Problems." In: *Proc. of a Symposium on the Complexity of Computer Computations*. 1972, pp. 85–103. URL: http://www.cs.berkeley.edu/~luca/cs172/karp.pdf (cit. on p. 4).

[86] Bastian Katz, Markus Völker, and Dorothea Wagner. "Energy efficient scheduling with power control for wireless networks." In: *Proc. 8th Internat. Symp. on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt'10)*. IEEE. 2010, pp. 160–169 (cit. on p. 11).

[87] Thomas Kesselheim. "A Constant-factor Approximation for Wireless Capacity Maximization with Power Control in the SINR Model." In: *Proc. 22th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA'11)*. SODA '11. SIAM, 2011, pp. 1549–1559. DOI: 10.1137/1.9781611973082.120 (cit. on p. 11).

[88] Thomas Kesselheim and Berthold Vöcking. "Distributed Contention Resolution in Wireless Networks." In: *Proc. 24th Internat. Symp. on Distributed Computing (DISC'10)*. Springer, 2010, pp. 163–178. DOI: 10.1007/978-3-642-15763-9_16 (cit. on pp. 11, 22).

[89] Donald E. Knuth. *Fundamental Algorithms*. Vol. 1. The Art of Computer Programming. Addison-Wesley, 2011 (cit. on pp. 108, 110).

[90] Kishore Kothapalli, Christian Scheideler, Melih Onus, and Christian Schindelhauer. "Distributed coloring in $O(\sqrt{\log n})$ bit rounds." In: *Proc. 20th Internat. Parallel and Distributed Processing Symp. (IPDPS'06)*. 2006. DOI: 10.1109/IPDPS.2006.1639281 (cit. on p. 51).

[91] Fabian Kuhn, Nancy Lynch, and Calvin Newport. "The Abstract MAC Layer." In: *J. Distr. Comp.* 24.3-4 (11/2011), pp. 187–206 (cit. on p. 103).

[92] Fabian Kuhn and Roger Wattenhofer. "On the complexity of distributed graph coloring." In: *Proc. 25th ACM Symp. on Principles of Distributed Computing (PODC'06)*. 2006, pp. 7–15. DOI: 10.1145/1146381.1146387 (cit. on p. 51).

[93] Shay Kutten and David Peleg. "Fast Distributed Construction of Small k-Dominating Sets and Applications." In: *J. Algorithms* 28.1 (1998), pp. 40–66. DOI: 10.1006/jagm.1998.0929 (cit. on p. 147).

[94] Hongxing Li, Qiang Sheng Hua, Chuan Wu, and Francis C. M. Lau. "Minimum-latency aggregation scheduling in wireless sensor networks under physical interference model." In: *Proc. 13th ACM Internat. Conf. on Modeling, Analysis, and Simulation of Wireless and Mobile Systems (MSWIM'10)*. ACM. 2010, pp. 360–367. DOI: 10.1145/1868521.1868581 (cit. on p. 12).

[95] Xiang-Yang Li, XiaoHua Xu, ShiGuang Wang, ShaoJie Tang, GuoJun Dai, JiZhong Zhao, and Yong Qi. "Efficient data aggregation in multi-hop wireless sensor networks under physical interference model." In: *Proc. 6th IEEE Internat. Conf. on Mobile Adhoc and Sensor Systems (MASS'09)*. IEEE. 2009, pp. 353–362. DOI: 10.1109/MOBHOC.2009.5336978 (cit. on p. 12).

[96] Nathan Linial. "Distributive graph algorithms Global solutions from local data." In: *Proc. 28th Ann. IEEE Symp. Foundations Comput. Sci. (FOCS'87)*. IEEE. 1987, pp. 331–335. DOI: 10.1109/SFCS.1987.20 (cit. on p. 50).

[97] Nathan Linial. "Locality in distributed graph algorithms." In: *SIAM Journal on Computing* 21.1 (1992), pp. 193–201. DOI: 10.1137/0221015 (cit. on p. 50).

[98] Michael Luby. "A simple parallel algorithm for the maximal independent set problem." In: *SIAM J. Comput.* 15.4 (1986), pp. 1036–1053. DOI: 10.1137/0215074 (cit. on pp. 51, 54).

[99] Luca Mainetti, Luigi Patrono, and Antonio Vilei. "Evolution of wireless sensor networks towards the Internet of Things: A survey." In: *Proc. 19th Internat. Conf. on Software, Telecomm. and Computer Networks (SoftCOM), 2011*. 09/2011, pp. 1–6 (cit. on p. 2).

[100] Madhav V. Marathe, Alessandro Panconesi, and Larry D. Risinger Jr. "An experimental study of a simple, distributed edge-coloring algorithm." In: *ACM Journal of Experimental Algorithmics* 9 (2004). DOI: 10.1145/1005813.1041515 (cit. on p. 70).

[101] Dániel Marx. "Graph colouring problems and their applications in scheduling." In: *Electrical Engineering* 48.1-2 (2004), pp. 11–16 (cit. on p. 4).

[102] Karl Menger. "Zur allgemeinen Kurventheorie." ger. In: *Fundamenta Mathematicae* 10.1 (1927). german, pp. 96–115 (cit. on p. 120).

[103] Thomas Moscibroda and Roger Wattenhofer. "Coloring unstructured radio networks." In: *Proc. 17th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA'05)*. ACM. 2005, pp. 39–48. DOI: 10.1145/1073970.1073977 (cit. on p. 23).

[104] Thomas Moscibroda and Roger Wattenhofer. "Coloring unstructured radio networks." In: *Distributed Computing* 21.4 (2008), pp. 271–284. DOI: 10.1007/s00446-008-0070-4 (cit. on pp. 12, 21, 23, 33, 35, 51, 69, 70, 103).

[105] Thomas Moscibroda, Roger Wattenhofer, and Yves Weber. "Protocol design beyond graph-based models." In: *Proc. of the ACM Workshop on Hot Topics in Networks (HotNets-V)*. 2006, pp. 25–30 (cit. on pp. 10, 134).

[106] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Chapman & Hall/CRC, 2010 (cit. on p. 10).

[107] Ayfer Özgür, Olivier Lévêque, and David Tse. "Hierarchical cooperation achieves optimal capacity scaling in ad hoc networks." In: *IEEE Transactions on Information Theory* 53.10 (2007), pp. 3549–3572 (cit. on p. 11).

[108] Alessandro Panconesi and Aravind Srinivasan. "On the complexity of distributed network decomposition." In: *J. Algorithms* 20.2 (1996), pp. 356–374. DOI: 10.1006/jagm.1996.0017 (cit. on p. 51).

[109] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial Mathematics, 2000 (cit. on pp. 9, 22).

[110] Satya Krishna Pindiproli and Kishore Kothapalli. "Experimental Analysis of Distributed Coloring Algorithms." In: *IEEE Internat. Advance Computing Conf. (IACC'09)*. 03/2009, pp. 147–152. DOI: 10.1109/IADCC.2009.4808997 (cit. on pp. 51, 70).

[111] Neeraj Poojary, Srikanth V. Krishnamurthy, and Son Dao. "Medium access control in a network of ad hoc mobile nodes with heterogeneous power capabilities." In: *Proc. IEEE Internat. Conf. on Communications (ICC'01)*. Vol. 3. IEEE. 2001, pp. 872–877. DOI: 10.1109/ICC.2001.937363 (cit. on p. 22).

[112] Krishna N. Ramachandran, Elizabeth M Belding-Royer, Kevin C. Almeroth, and Milind M. Buddhikot. "Interference-Aware Channel Assignment in Multi-Radio Wireless Mesh Networks." In: *Proc. 25th. IEEE Internat. Conference on Comput. Comm. (INFOCOM'06)*. Vol. 6. 2006, pp. 1–12. DOI: 10.1109/INFOCOM.2006.177 (cit. on p. 9).

[113] Ram Ramanathan. "A unified framework and algorithm for channel assignment in wireless networks." In: *Wireless Networks* (1999), pp. 81–94. DOI: 10.1023/A:1019126406181 (cit. on pp. 8, 49).

[114] Theodore S. Rappaport. *Wireless communications : principles and practice*. 2. ed., 18. print. Prentice Hall communications engineering and emerging technologies series. Prentice Hall, 2009 (cit. on p. 15).

[115] Lawrence G. Roberts. "ALOHA Packet System with and Without Slots and Capture." In: *SIGCOMM Comput. Commun. Rev.* 5.2 (04/1975), pp. 28–42. DOI: 10.1145/1024916.1024920 (cit. on p. 10).

[116] Kay Römer and Friedemann Mattern. "The design space of wireless sensor networks." In: *IEEE Wireless Communications* 11.6 (2004), pp. 54–61. DOI: 10.1109/MWC.2004.1368897 (cit. on p. 2).

[117] Christian Scheideler, Andrea W. Richa, and Paolo Santi. "An O(log n) Dominating Set Protocol for Wireless Ad-Hoc Networks under the Physical Interference Model." In: *Proc. 9th ACM Internat. Symp. on Mobile Ad Hoc Networking and Computing (MOBIHOC'08)*. 2008, pp. 91–100. DOI: 10.1145/1374618.1374632 (cit. on p. 12).

[118] Markus Schlegel. "Experimental Evaluation of Distributed Node Coloring Algorithms in the SINR Model." Bachelor thesis. Karlsruhe, Germany: Karlsruhe Institute of Technology (KIT), 2015. URL: http://i11www.iti.uni-karlsruhe.de/_media/teaching/theses/ba-schlegel-15.pdf (cit. on p. 71).

[119] Johannes Schneider and Roger Wattenhofer. "A Log-star Distributed Maximal Independent Set Algorithm for Growth-bounded Graphs." In: *Proc. 27th ACM Symp. on Principles of Distributed Computing (PODC'08)*. ACM, 2008, pp. 35–44. DOI: 10.1145/1400751.1400758 (cit. on p. 51).

[120] Johannes Schneider and Roger Wattenhofer. "Coloring unstructured wireless multi-hop networks." In: *Proc. 28th ACM Symp. on Principles of Distributed Computing (PODC'09)*. ACM. 2009, pp. 210–219. DOI: 10.1145/1582716.1582751 (cit. on pp. 23, 62, 63).

[121] Yi Shi, Y. Thomas Hou, Jia Liu, and Sastry Kompella. "Bridging the gap between protocol and physical models for wireless networks." In: *Mobile Computing, IEEE Transactions on* 12.7 (2013), pp. 1404–1416. DOI: 10.1109/TMC.2012.118 (cit. on p. 9).

[122] Fikret Sivrikaya and Bülent Yener. "Time synchronization in sensor networks: a survey." In: *IEEE Network* 18.4 (2004), pp. 45–50. DOI: 10.1109/MNET.2004.1316761 (cit. on pp. 67, 105).

[123] Xu Su and Rajendra V. Boppana. "On the impact of noise on mobile ad hoc networks." In: *Proc. 2007 Internat. Conf. on Wireless Communications and Mobile Computing (IWCMC'06)*. ACM. 2007, pp. 208–213. DOI: 10.1145/1280940.1280986 (cit. on p. 15).

[124] Ramakrishna Thurimella. "Sub-Linear Distributed Algorithms for Sparse Certificates and Biconnected Components." In: *J. Algorithms* 23.1 (1997), pp. 160–179. DOI: 10.1006/jagm.1996.0832 (cit. on pp. 121, 147).

[125] Raymond S. Tomlinson. "Selecting Sequence Numbers." In: *Proc. of 1975 ACM SIGCOMM/SIGOPS Workshop on Interprocess Communications*. ACM, 1975, pp. 11–23. DOI: 10.1145/800272.810894 (cit. on p. 36).

[126] Tigran Tonoyan. "On some bounds on the optimum schedule length in the SINR model." In: *Proc. 9th Internat. Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS'13)*. Springer, 2013, pp. 120–131. DOI: 10.1007/978-3-642-36092-3_14 (cit. on p. 11).

[127] David Tse and Pramod Viswanath. *Fundamentals of wireless communication*. Cambridge university press, 2005 (cit. on p. 21).

[128] Guoqiang Wang, Damla Turgut, Ladislau Bölöni, Yongchang Ji, and Dan C. Marinescu. "A MAC layer protocol for wireless networks with asymmetric links." In: *Ad Hoc Networks* 6.3 (2008), pp. 424–440. DOI: 10.1016/j.adhoc.2007.03.004 (cit. on p. 22).

[129] Weizhao Wang, Yu Wang, Xiang-Yang Li, Wen-Zhan Song, and Ophir Frieder. "Efficient interference-aware TDMA link scheduling for static wireless networks." In: *Proc. 12th Ann. Internat. Conf. on Mobile Computing and Networking (MobiCom'06)*. ACM. 2006, pp. 262–273. DOI: 10.1145/1161089.1161119 (cit. on p. 9).

[130] Matthias Wolf. "On the Distributed Computation of Fractional Connected Dominated Set Packings." Bachelor thesis. Karlsruhe, Germany: Karlsruhe Institute of Technology (KIT), 2014. URL: http://i11www.iti.uni-karlsruhe.de/_media/teaching/theses/ba-wolf-14.pdf (cit. on pp. 115, 122, 128).

[131] Tai-Kuo Woo, Stanley YW Su, and Richard Newman-Wolfe. "Resource allocation in a dynamically partitionable bus network using a graph coloring algorithm." In: *IEEE Transactions on Communications* 39.12 (1991), pp. 1794–1801. DOI: 10.1109/26.120165 (cit. on p. 4).

[132] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. "Wireless sensor network survey." In: *Computer Networks* 52.12 (2008), pp. 2292–2330. DOI: 10.1016/j.comnet.2008.04.002 (cit. on p. 2).

[133] Dongxiao Yu, Qiang-Sheng Hua, Yuexuan Wang, and Francis C. M. Lau. "An O(log n) Distributed Approximation Algorithm for Local Broadcasting in Unstructured Wireless Networks." In: *Proc. 8th Internat. Conf. on Distributed Computing in Sensor Systems (DCOSS'12)*. 2012, pp. 132–139. DOI: 10.1109/DCOSS.2012.39 (cit. on pp. 11, 22, 29, 30, 31, 49, 67, 103, 104).

[134] Dongxiao Yu, Qiang-Sheng Hua, Yuexuan Wang, Haisheng Tan, and Francis C. M. Lau. "Distributed Multiple-Message Broadcast in Wireless Ad-Hoc Networks under the SINR Model." In: *Proc. 19st Internat. Colloq. Structural Inform. and Communication Complexity (SIROCCO'12)*. Vol. 7355. Lecture Notes Comput. Sci. Springer, 2012, pp. 111–122. DOI: 10.1007/978-3-642-31104-8_10 (cit. on p. 12).

[135] Dongxiao Yu, Qiang-Sheng Hua, Yuexuan Wang, Jiguo Yu, and Francis C. M. Lau. "Efficient distributed multiple-message broadcasting in unstructured wireless networks." In: *Proc. 2013 IEEE Internat. Conference on Comput. Comm. (INFOCOM'13)*. 2013, pp. 2427–2435. DOI: 10.1109/INFCOM.2013.6567048 (cit. on p. 12).

[136] Dongxiao Yu, Yuexuan Wang, Qiang-Sheng Hua, and Francis C. M. Lau. "Distributed (Δ+1)-coloring in the physical model." In: *Theor. Comput. Sci.* 553 (2014), pp. 37–56. DOI: 10.1016/j.tcs.2014.05.016 (cit. on pp. 12, 22, 23, 50, 51, 53, 69, 70, 75, 96).

[137] Dongxiao Yu, Yuexuan Wang, Qiang-Sheng Hua, and Francis C. M. Lau. "Distributed Local Broadcasting Algorithms in the Physical Interference Model." In: *Proc. 7th Internat. Conf. on Distributed Computing in Sensor Systems (DCOSS'11)*. IEEE Computer Society, 2011, pp. 1–8. DOI: 10.1109/DCOSS.2011.5982163 (cit. on pp. 11, 22, 29, 31, 103, 104).

[138] Jiguo Yu, Lili Jia, Wei Li, Xiuzhen Cheng, Shengling Wang, Rongfang Bie, and Dongxiao Yu. "A Self-Stabilizing Algorithm for CDS Construction with Constant Approximation in Wireless Networks under SINR Model." In: *Proc. 35th Internat. Conf. on Distributed Computing Systems (ICDCS'15)*. 2015, pp. 792–793. DOI: 10.1109/ICDCS.2015.112 (cit. on p. 12).

[139] Yan Zhang, Honglin Hu, and Jijun Luo. *Distributed antenna systems: open architecture for future wireless communications*. CRC Press, 2007 (cit. on p. 15).

[140] Megat Zuhairi, Haseeb Zafar, and David Harle. "On-demand routing with unidirectional link using path loss estimation technique." In: *Proc. Wireless Telecommunications Symposium (WTS'12)*. 2012, pp. 1–7. DOI: 10.1109/WTS.2012.6266131 (cit. on p. 22).

CURRICULUM VITÆ

Name	Fabian Fuchs
Date of Birth	2 June 1987
Place of Birth	Sinsheim, Germany
Nationality	German

04/2014 – 07/2014	Research intern at Microsoft Research Silicon Valley in Mountain View, California, USA.
since 04/2012	Ph.D. student and research assistant in the research group Algorithmics of Prof. Dr. Dorothea Wagner. Faculty for Informatics, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
03/2012	Diploma (equiv. to Master's degree) in computer science at the Faculty of Informatics, KIT, Karlsruhe, Germany
10/2007 – 03/2012	Studies of Computer Science at the Faculty of Informatics, KIT, Karlsruhe, Germany (previously: Karlsruhe University)
06/2006	Abitur (university entrance qualification), Wilhelm-Maybach-Schule, Heilbronn, Germany

LIST OF PUBLICATIONS

[1] Fabian Fuchs. "Experimental Evaluation of Distributed Node Coloring Algorithms for Wireless Networks." In *Proceedings of the 18th Workshop on Algorithm Engineering and Experiments (ALENEX'16)*. SIAM, 2016, pp. 30–38. DOI: 10.1137/1.9781611974317.3.

[2] Fabian Fuchs and Roman Prutkin. "Simple Distributed Degree + 1 Coloring in the SINR Model." In *Proceedings of the 22th International Colloquium on Structural Information and Communication Complexity (SIROCCO'15)*. Lecture Notes in Computer Science. Springer, 2015, pp. 149–163. DOI: 10.1007/978-3-319-25258-2_11

[3] Fabian Fuchs. "Brief Announcement: Fast and Simple Node Coloring in the SINR Model." In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing (PODC'15)*. ACM Press, 2015, pp. 139–141. DOI: 10.1145/2767386.2767445.

[4] Fabian Fuchs and Dorothea Wagner. "Local Broadcasting with Arbitrary Transmission Power in the SINR Model." In *Proceedings of the 21th International Colloquium on Structural Information and Communication Complexity (SIROCCO'14)*. Vol. 8576. Lecture Notes in Computer Science. Springer, 2014, pp. 180–193. DOI: 10.1007/978-3-319-09620-9_15.

[5] Fabian Fuchs and Dorothea Wagner. "On Local Broadcasting Schedules and CONGEST Algorithms in the SINR Model." In *Proceedings of the 9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS'13)*, Vol. 8243. Lecture Notes in Computer Science. Springer, 2013, pp. 170–184. DOI: 10.1007/978-3-642-45346-5_13.

[6] Edith Cohen, Daniel Delling, Fabian Fuchs, Andrew V. Goldberg, Moises Goldszmidt, and Renato F. Werneck. "Scalable Similarity Estimation in Social Networks: Closeness, Node Labels, and Random Edge Lengths." In *Proceedings of the ACM Conference on Online Social Networks (COSN'13)*. Vol. 1. ACM Press, 2013, pp. 131–142. DOI: 10.1145/2512938.2512944.

[7] Fabian Fuchs, Markus Völker, and Dorothea Wagner. "Simulation-based Analysis of Topology Control Algorithms for Wireless Ad Hoc Networks." In *Proceedings of the 1st Mediterranean Conference on Algorithms (MedAlg'12)*, volume 7659 of *Lecture Notes in Computer Science*. Springer, 2012 , pp. 188–202. DOI: 10.1007/978-3-642-34862-4_14.

DECLARATION

Ich versichere hiermit, diese Dissertation selbstständig angefertigt und alle benutzten Hilfsmittel vollständig angegeben zu haben. Außerdem versichere ich, kentlich gemacht zu haben, was aus Arbeiten anderer und eigener Veröffentlichungen unverändert oder mit Änderungen entnommen wurde.

Karlsruhe, 27. Jan. 2016

Fabian Fuchs