DISS. ETH NO. 20083
DISS. TIK NO. 129

Decentralized Coordination: Methods and Applications

A dissertation submitted to

ETH ZURICH

for the degree of

Doctor of Sciences

presented by

JOHANNES SCHNEIDER

MSc ETH INFK, ETH Zürich

born 10.10.1979

citizen of
Zurich (ZH)

accepted on the recommendation of

Prof. Roger Wattenhofer, examiner
Prof. Uzi Vishkin, co-examiner
Prof. Rachid Guerraoui, co-examiner

2011

Abstract

We look at several problems in the field of distributed and parallel computing. Part I presents symmetry breaking techniques in the message passing model, i.e. each entity in a network can communicate through exchanging a distinct message with each of its neighbors in a synchronized round without failures. We introduce Multi-Trials, a new technique for symmetry breaking for distributed algorithms and apply it to various problems in general graphs. For instance, the technique is demonstrated by giving three randomized algorithms for distributed (vertex or edge) coloring and a deterministic algorithm for computing ruling sets. Additionally, we present a coloring algorithm whose running time and number of colors needed depends on the chromatic number of the graph. We also generalize the deterministic coin tossing technique from rings to general graphs. The optimality of our algorithm is proven for the maximal independent set problem for geometric graphs such as unit disc graphs. Furthermore, we show how to trade among different complexity measures with a focus on different coloring and MIS algorithms. Finally, we derive techniques and bounds on trading among different complexity measures, such as the number of exchanged bits, messages and the number of communication rounds.

Part II investigates various problems related to wireless networks. In our theoretical investigation we look at different communication models and primarily focus on geometric graphs. The models share the assumption that the network topology is unknown and nodes wake up asynchronously. Furthermore, in a (synchronized) communication round the same message is broadcast to all neighbors but might be lost due to collisions of concurrent transmitters. We look at a model where nodes can detect collisions, e.g. through carrier sensing, and a model, where nodes are (almost) completely unaware of the number of transmitting neighbors. We derive several lower and upper bounds in both models for problems such as coloring, maximal independent set and broadcast. We show that the asymptotic gain in the time complexity when using collision detection depends heavily on the task at hand (and the maximal degree of the graph), i.e. it ranges from exponential down to no asymptotic gain at all.

In the practical part we derive (and evaluate on actual sensor nodes) a form of pulse-position modulation, i.e. message-position modulation (MPM), to reduce the payload of a message by encoding parts of the message through its transmission time. MPM allows to conserve energy and channel usage, thus reducing the risk of collisions and increasing available bandwidth.

Additionally, we present a technique called "Three Plane Localization" to improve accuracy of range based and range free localization schemes. The key

idea is to intentionally create interference at a node by scheduling concurrent transmissions of nearby nodes.

Part III deals with parallel shared memory systems, e.g. a conventional (multi-core) PC. We discuss aspects of transactional memory, (tree) data structures and graph based (parallel) algorithms. To improve throughput in transactional memory systems we analyze and present different strategies for contention management as well as load adaption mechanisms. Our contention management policies try to optimize the throughput by making the right decision in case of conflicts, i.e. decide which transaction has to abort, wait or can continue. However, new transactions are allowed to start (independent of overall system contention) and transactions are (usually) not delayed for a long time. We present and analyze two new algorithms for contention management.

We also look at load adaption strategies, i.e. how to optimize throughput by (temporarily) keeping computational resources, i.e. cores, idle. Opposed to prior work our load adapting schemes are simple and fully distributed, while maintaining the same throughput rate. Our experimental results show a substantial overall improvement for our best performing strategies compared to the best existing contention management policies (without load adaption).

For data structures representable by directed acyclic graphs, i.e. rooted trees, we show how to partition them to allow for efficient complex operations, which lie beyond inserts, deletes and finds. The approach potentially improves the performance of any operation modifying more than one element of the data structure. It covers common data structures implementable via linked lists or trees such as sets and maps.

Finally, we give a graph decomposition technique that creates entirely independent subproblems for graph problems such as coloring and dominating sets. The solutions of the subproblems can be computed as well as composed to an overall solution without synchronization on a shared memory system. The technique allows to trade performance for solution quality.

Zusammenfassung

Wir untersuchen einige Probleme im Bereich des verteilten und parallelen Rechnens. Teil I fokussiert auf Symmetriebrechung im Message-Passing-Modell, d.h. jede Einheit in einem Netzwerk kann durch den Austausch einer Nachricht mit jedem seiner Nachbarn in einer synchronisierten Runde ohne Fehler kommunizieren. Wir stellen die Multi-Trials Technik vor und wenden sie auf verschiedene Probleme in allgemeinen Graphen an. Die Technik dient beispielsweise als Grundlage für drei randomisierte Algorithmen für verteiltes Färben eines Graphen und für ein deterministisches Verfahren zur Berechnung von Ruling Sets. Zu dem diskutieren wir einen Algorithmus zum Färben von Graphen in Abhängigkeit der chromatischen Zahl des Graphens.

Wir verallgemeinern die deterministische "Coin-Tossing" Technik von Ringen auf allgemeine Graphen. Wir beweisen die Optimalität des Algorithmus für das maximal unabhängige Mengen Problem für geometrische Graphen wie Einheitskreis Graphen. Des weiteren geben wir Austauschbeziehungen zwischen verschiedenen Komplexitätsmassen für verteilte Algorithmen an, wobei unser Fokus auf Symmetrybrechungsalgorithmen liegt.

Teil II untersucht verschiedene Probleme im Zusammenhang mit drahtlosen Netzwerken. Wir betrachten verschiedene Kommunikationsmodelle und geometrische Graphen. Die Modelle teilen die Annahme, dass die Netzwerk-Topologie unbekannt ist und Knoten asynchron aufwachen. Ferner wird in einer (synchronisierten) Kommunikationsrunde die gleiche Nachricht an alle Nachbarn gesendet. Diese könnte aufgrund von Kollisionen von mehreren gleichzeitigen Sendern jedoch verloren gehen. Wir betrachten ein Modell, bei dem Knoten Kollisionen erkennen können, z. B. durch Carrier Sensing, und ein Modell, in dem Knoten pro Runde (fast) keine Information über die Anzahl der sendenden Nachbarn erhalten. Wir leiten mehrere untere und obere Schranken in beiden Modellen für Probleme wie Farbgebung, maximal unabhängige Mengen und Broadcast. Wir zeigen, dass der asymptotische Gewinn in der Zeitkomplexität bei der Verwendung von Kollisionserkennung stark von der Aufgabenstellung (und des maximale Grades des Graphen) abhängt. Er reicht von exponentiell bis kein asymptotischer Gewinn.

Im praktischen Teil beschreiben wir (und evaluieren mit echten Sensorknoten) eine Form von Puls-Position-Modulation, dh Message-Position-Modulation (MPM), um die gesendete Nachricht durch Verschlüsselung eines Teils der Nachricht über seine Übertragungszeit zu verkleinern. MPM reduziert die Energie- und Kanalnutzung, wodurch die Gefahr von Kollisionen reduziert wird und sich die verfügbare Bandbreite erhöht.

Darüber hinaus präsentieren wir eine Technik namens “Three Plane Localization”, um die Genauigkeit der Lokalisierung von Sensorknoten zu verbessern.

Teil III beschäftigt sich mit Parallelrechnern, welche über gemeinsam genutzten Speicher kommunizieren, z. B. ein konventioneller Mehrkern PC. Wir diskutieren Aspekte von Transactional Memory, (Baum-)Datenstrukturen und Graphen- basierten (parallelen) Algorithmen. Zur Verbesserung des Durchsatzes in Transactional Memory Systemen untersuchen wir verschiedene Strategien für Contention Management als auch Last-Adaptions Mechanismen. Diese Strategien müssen entscheiden was mit zwei Transaktionen geschieht, welche im Konflikt stehen. Eine Transaktion kann abgebrochen oder verzögert werden oder fortfahren.

Für Datenstrukturen, welche als gerichtete azyklische Graphen darstellbar sind, zeigen wir, wie diese zu partitionieren sind, um komplexe Operationen, die über das Einfügen, Löschen und Suchen hinaus gehen, effizient zu berechnen.

Schließlich präsentieren wir einen Technik, die völlig unabhängig Teilprobleme für Graphen Probleme wie Farbgebung erzeigt. Die Lösung der Subprobleme als auch das Zusammenfügen der Teillösungen zu einer Gesamtlösung kann ohne Synchronisation erfolgen. Die Technik erlaubt es Rechenzeit für Lösungsqualität einzutauschen.

Acknowledgements

To begin with, I would like to express my gratitude to the persons that made this thesis possible: My parents, Anton and Evi, and my grandparents, in particular, Hertha Valandro. With their education they truly laid the foundation for this thesis. I am also grateful to my advisor Roger Wattenhofer for guiding me through my graduate studies. My gratitude also goes to all members of the Distributed Computing Group, in particular to my long term roommates (Raphael Eidenbenz and Remo Meier) and my collaborator in supervising several student thesis Philipp Sommer. Michael Kuhn was also a good sparring partner for ice-hockey as well as for table soccer. During my research stay in Israel I enjoyed working with Michael Elkin and Leonid Barenhoim very much. Thank you for all the interesting discussions. My time in Berlin with Stefan Schmid was equally great. Thanks also goes to my co-examiners professor Uzi Vishkin and professor Rachid Guerraoui who took the time to review this thesis despite their busy schedules. I also want to thank my advisor during my master thesis professor Osamu Watanabe whose brilliance in science and cleverness in teaching served as an excellent guide for me. Not to forget in the list of people I want to acknowledge are the undergraduate students I worked with for their dedicated work. Particularly, I want to acknowledge Fabian Landau and David Hasenfratz with whom I co-authored two publications. I also want to express my gratitude to Fabian Kuhn for his valuable feedback and Thomas Moscriboda for interesting conversations at multiple conferences. I am also grateful to all persons not listed here (e.g. system administrators, administrative stuff), but who contributed in a positive way to my time (as Ph.D. student).

Contents

Chapter 1

Introduction

Decentralized coordination is a fundamental problem not just in parallel as well as distributed computing. By a *decentralized* system we refer to a system where multiple (equal) entities must communicate to solve a problem or should do so due to efficiency reasons. For example, it might not be feasible to compute the solution in a central manner, e.g., a single entity of the network collects all information, solves the problem and distributes the solution through the network. In other words no unit in the network has access to all relevant input values for the problem or cannot efficiently solve the problem for the whole network on its own. Any community, country or company faces a trade-off between the amount of decentralized and central decision making. For example, in the army it is not clear how large a group should be that is led by a commander. Solidiers in a group have equal rank. Clearly, the larger the group the more cumbersome (and communication intensive) it becomes to find an agreement within the group. However, the number of levels in the hierarchy also increases the amount of communication. Making qualitative statements might seem demanding. Quantifying such trade-offs is for sure a hard tasks. Some of the models employed in the thesis are very general. We hope that the models together with their algoritms and proofs might help other researchers to shed some light on such basic questions in the future.

In the technical domain, there are many decentralized systems that have gained in relevance in the last decades. For instance, the rise of multi-core systems in recent years has boosted the importance of concurrent computing. All cores/CPUs must work together to harness the power of these systems. Another example are wireless networks which have also become omnipresent in the last decade. Networks should automatically configure themselves upon deployment, e.g. compute transmission schedules and derive routing schemes. More concretely, when low power sensor networks are installed the nodes

should determine a feasible (collision free) transmission schedule to gather data without requiring any information about the environment or network. Around 2005 many companies have believed that business of (low power) sensor networks would take-off dramatically. This is not surprising since the application domains are numerous, ranging from industry to ordinary households. The military has dreamed about throwing thousands of sensors out of a plane that organize themselves and deliver information about the enemy. There are many reasons, why these expectations have not been fulfilled to that extend. One is that wireless networks are very dynamic. In wireless networks links and nodes might be newly introduced, removed or may fail and thus routing tables, transmission schedules etc. must be updated fast or computed quickly from scratch. It is generally not recommended for large networks to gather all topology information at a single node. This might take too much time, i.e. the solution might be outdated even before a node receives the messages containing it. Bandwidth and, in particular, energy constraints might also make this approach unfavorable. Therefore, algorithms are typically distributed, e.g. each node participates in the computation of the solution. A node is either unaware or only partly aware of the overall network topology and typically communication takes place through message exchange. Many early approaches in the beginning of the 21st century to build low power wireless sensor networks have failed using a centralized approach due to the dynamics of the network. A decade later, industry standards like WirelessHART have condensed the experiences from research. They accounted to some extend for these difficulties. However, the approach relies heavily on central network managers. Large deployments (including a few hundred of nodes) have yet to be made. It is doubtful that the promise of complete self-organization and robustness is really true for big networks. Our work addresses scenarios with a large number of wireless entitites and we believe that our methodoligies will be of particular interest as soon as large (and dense) networks start to emerge. It is a common believe of many scientists (in the ubiquitous computing communinity) that more or and more items, e.g. fridge, TV, phones etc., are equipped with sensors and linked together, e.g. through the internet.

Distributed systems are sometimes supposed to be failure tolerant, i.e. the system must still work in case a single entity, such as a core or sensor node, fails and potentially behaves maliciously. For example, this is essential for banking applications where the crash of a server should not result in lost (or incorrect) banking transactions. In this thesis we do not consider fault tolerance, i.e. we assume that entities work as they are supposed to. However, we believe that many of our algorithms can be adapted to deal with common failures in the network. In the case of wireless networks this belief is based on the observation that in case a node fails often a solution is

(partially) recomputed. Thus, the same problem is solved but on a smaller scale.

A distributed algorithm is most often characterized by the amount and duration of necessary communication, whereas the amount of computation is usually irrelevant (at least as long as it remains polynomial in the input size). Communication in turn, can be measured by the number of exchanged bits, messages and the time needed. Although time is mainly considered the most important factor, in some cases other measures, i.e. the number of bits transmitted, are more important. For example, an extremely fast algorithm requiring large messages might not be feasible due to bandwidth limitations. Adressing such constraints by non-centralized algorithms adds to the problem complexity. Some problems are simple in a centralized manner like greedy coloring but require sophisticated algorithms in a distributed setting. Solving a problem in a distributed manner might also come at the price of reduced solution quality. When only local knowledge is taken into account, many problems yield worse solutions compared to the case where a single entity solves the problem by looking at the entire input.

In contrast to distributed systems, in parallel systems (like multi-core computers) each processor has access to memory shared by all processors. However, concurrent access to the same memory often requires synchronization to avoid problems like lost updates. In other words, one processor must work on a memory cell atomically, i.e. read the value of the cell, modify it in its local registers and write it back into the shared memory without any other processor being able to modify the memory cell concurrently. Since synchronization is typically very time consuming on modern computer architectures compared to simple operations, it should be kept at a minimum. For instance, a counter which is frequently incremented by different processors should be avoided since it potentially limits scalability of the system. Meaning that eventually adding further cores will not yield additional performance gain, since the system is "blocked" due to the synchronization required to provide a consistnt value of the shared counter to all cores. Minimizing the amount of synchronization is an error-prone and time-consuming task for a software engineer. Usually, the burden of writing correct sequential code is already quite heavy for a programmer. Thus, for standard procedures like storing and retrieving data from a concurrent data structures ready to use algorithms (or even better libraries) should be provided. Generally, given the high labor costs of educated people, for any task some kind of computer-assisted parallelization scheme is preferable to letting a software developer spend its energy and time. This might hold even more in the future.

In Part I we look at symmetry breaking techniques resulting in algorithms with provable worst case guarantees as well as good average case behavior as shown by simulations. Symmetry breaking is often related to scheduling and

usually refers to the situation, where we start from an unorganized (partial) network, where we want to select one or several entities in the network. The chosen entities might be allowed to use a certain resource, i.e. transmit a message in a wireless network or access a memory cell in a multi-core computer. We also elaborate on trade-offs between different complexity measures for distributed (symmetry breaking) algorithms.

In Part II we discuss a variety of coordination tasks such as computing a transmission schedule in different models of wireless networks by deriving lower and upper bounds for various problems. In the practical part, we additionally briefly look into the area of localization, i.e. determining the position of a node.

In Part III we dig into parallel algorithms for handling concurrent tree data structures as well as into minimizin the amount of synchronization required to solve certain graph problems (on a multi-core computer). However, the main focus is scheduling for transactional memory. Transactional memory allows for easy and efficient concurrent programming by simply wrapping source code into transactions. We discuss how to resolve conflicts among transactions efficiently and how to adapt the load of a transactional memory system.

Part I

Symmetry Breaking

Chapter 2

Introduction

Symmetry breaking is a fundamental problem in distributed computing, and as such immensely studied. Its applications range from resource scheduling for parallel threads in a multi-core environment (i.e. MIS) to transmission scheduling in wireless networks (i.e. coloring) on to network decompositions.

When multiple unorganized units have to be coordinated, communication costs usually outweigh the costs for (local) computation. Good coordination is usually a necessary means to solve a problem. Once a communication infrastructure has been set up, problems can be solved efficiently. Many people have experienced themselves that deciding together on a person out of a group to solve a problem can be very cumbersome and time-consuming. Sometimes long discussions evolve. Things get even more complicated if communication is not only direct, i.e. if a group of people discusses and some people of the group need to come to an agreement also with other (non-present) people. All too often, it is easier to solve the problem oneself instead of going through the communication process. This problem occurs, in particular, if there is a group of homogenous people without any dedicated boss and there are no clear arguments which person should address the issue. In such situations all too frequently long discussions take place and in the end, not seldom, a coin is flipped to make a decision, i.e. random choices are made. In some sense, we derive algorithms to make such decisions efficiently.

In the standard *message passing model* the complexity of an algorithm is measured in the number of communication rounds needed. Per round a node can exchange an arbitrary message with each of its neighbors.[1] This model is very general and, as we shall see in the wireless networks part (Part II), efficient transformation (with adaptations) of algorithms in the message passing model to other models is sometimes possible. In real world systems

[1] Often these messages are of polylogarithmic size in the number of nodes.

simple symmetry breaking is often accompanied with additional constraints. However, frequently, such constraints can be incorporated directly. For example, when computing a transmission schedule for all nodes, often some nodes should obtain more time to transmit than others. A simple coloring (corresponding to a TDMA transmission schedule) is not sufficient, since it yields the same time (number of slots) for each node. However, our TDMA algorithm (see Part II) based on the algorithm in the message passing model, is to be modified only slightly to address this issue. A node can just demand more colors, i.e. time slots, to assign a non-uniform number of slots, i.e. transmission times, to nodes. Even if the constraints cannot be employed directly, initial symmetry breaking can dramatically facilitate communication means to solve the problems. It allows for network decompositions, e.g. each node in a MIS leads a cluster and nodes not in the MIS, obey the nodes in the MIS. In each decomposition a solution can be computed by a leader or by restricting communication of nodes to other cluster members. This can severely simplify the communication process and reduce the amount of communication. Thus, (hierarchical) decompositions, which can be found in any company or state, are often a necessary means to solve a problem efficiently.

After stating the model and definitions as well as discussing related work in Sections 2.1 and 2.2 we introduce the *Multi-Trials* technique and apply it to the coloring, maximal independent set and ruling set problems in Chapter 3. In the following chapter (a distributed) coloring algorithm with few colors, i.e. below $\Delta + 1$, is presented. In Chapter 5 we generalize the deterministic coin tossing technique from directed rings to general graphs and give an optimal algorithm for the maximal independent set problem for unit disc graphs (and their generalizations). Afterwards, in Chapter 6 we derive lower and upper bounds among three complexities measures, i.e. time, bit and message complexity, with a focus on (but not restricted to) symmetry breaking algorithms.

2.1 Model and Definitions

The communication network is modeled with a graph $G = (V, E)$. For a node v its neighborhood $N^r(v)$ represents all nodes within r hops of v (not including v itself). By $N(v)$ we denote $N^1(v)$ and by $N_+(v)$ the neighborhood of v including v, i.e. $N(v) \cup v$. The degree $d(v)$ of a node v is defined as $|N(v)|$. $d_+(v)$ denotes $|N_+(v)|$, $\Delta := \max_{u \in V} d(u)$ and $\Delta_{N_+(v)} := \max_{u \in N_+(v)} d(u)$. In a (vertex) coloring any two neighboring nodes u, v have a different color. A set $T \subseteq V$ is said to be independent in G if no two nodes $u, v \in T$ are neighbors. A set is (α, β)-ruling if every two nodes in the set have distance at least α and any node not in the set has a node in the set within distance β. A set $S \subseteq V$ is a maximal independent set (MIS), if it is (2,1)-ruling. A

MIS S of maximum cardinality, i.e. $|S| \geq \max_{\text{MIS } T} |T|$, is called a maximum independent set (MaxIS).

Definition 1. *A graph $G = (V, E)$ is of bounded-independence if there is a bounding function $f(r)$ such that for each node $v \in V$, the size of a MaxIS in the neighborhood $N^r(v)$ is at most $f(r)$, $\forall r \geq 0$. We say that G if of polynomially bounded-independence if $f(r)$ is a polynomial $p(r)$.*

Equivalently, one might use the notion of a d-local α-doubling metric, i.e. a metric is d-local α-doubling if any ball of radius $r \leq d$ can be covered by at most α balls of radius $r/2$. A graph $G = (V, E)$ is of bounded-independence if it obeys the d-local $f(d)$-doubling metric for some function f.

In particular, this means that for a constant c the value $f(c)$ is also a constant. A subclass of bounded-independence graphs are quasi unit disk graphs and unit disk graphs, which are often used to model wireless communication networks and have $f(r) \in \mathrm{O}(r^2)$. However, we do not require the graph to be of polynomially bounded-independence, e.g. $f(r)$ might as well be exponential in r.

Our algorithm is uniform, i.e. it does not require any knowledge of the total number of nodes n. Communication among nodes is done in synchronous rounds without collisions, i.e. each node can exchange one distinct message of size $\mathrm{O}(\log n)$ bits with each neighbor. We understand that such a powerful communication layer is unrealistic in many application domains; in wireless networks for instance transmission collisions will happen, and must be addressed. We will discuss this in detail in the conclusions.

Our algorithm is non-uniform, i.e. every node knows an upper bound on the total number of nodes n. However, the maximum degree Δ in the graph is unknown. Communication among nodes is done in synchronous rounds without collisions, i.e. each node can exchange one distinct message of polylogarithmic size with each neighbor. Nodes start executing the algorithm concurrently.

The term $\log^b n$ denotes $(\log n)^b$. The term $\log^{(b)} n$ equals $\underbrace{\log \log \ldots \log}_{b \text{ times}} n$. The function $\log^* n$ states how often one has to take the (iterated) logarithm to get 1, i.e. $\log^{(\log^* n)} n = 1$. The tetration ${}^b a$ expresses $\underbrace{a^{a^{\cdot^{\cdot^{a}}}}}_{b \text{ times}}$.

Every node has a unique ID represented by l bits, where l is upper bounded by $\log n$. An ID and all other binary numbers are in little endian notation and have the form: $x^l, x^{l-1}, ..., x^1$, where $x^i \in \{0, 1\}$. Observe that for technical reasons the low order bit has index 1 (not 0).

2.2 Related Work

Symmetry breaking is one of the main problems in distributed computing. Deterministic algorithms need a way to distinguish between nodes, i.e. IDs. In their pioneering work Cole and Vishkin [24] established the "deterministic coin tossing" method. They applied it to compute a MIS in a ring graph. For more than two decades their deterministic coin tossing technique has been a method of choice for breaking symmetries. Consequently it has been used in various algorithms [44, 4, 76, 74, 73]. In [24] each node v has a successor $s(v)$, which is a neighbor of v. Based on its own serial number r_v (initially, the ID of node v) and the one of its successor $r_{s(v)}$, it iteratively computes a new serial number r'_v, which is the least bit number i in which both serial numbers differ with the bit at position i in r_v appended. For example, if $r_v := 10110$ and $r_{s(v)} := 11110$, we have $i = 3 = 11$ (binary) for little endian notation with bit 3 of r_v being 0 and thus $r'_v := 110$. After $O(\log^* n)$ computation node v's color corresponds to its serial number. However, this only allows for a node to get a new serial number, which is distinct from its successor (but not necessarily from other neighbors). Thus it seems that the technique is limited to simple graphs of low degree such as rings or other constant-degree graphs. In contrast to Cole/Vishkin our technique extends to unbounded degree. Roughly speaking, in our algorithm a node does not monotonously stick to the same successor but takes into account the serial numbers of all nodes whenever a new serial number is computed based on the previous one. More precisely, a node's "successor" is the node with current minimum serial number. Furthermore, we allow a node to pause for a while, i.e. it does not compute a new serial number for a couple of rounds. Additionally, a node might restart the computation of a serial number, e.g. a new serial number based on its own ID and the IDs of its (non-pausing) neighbors is computed.

The MIS problem has also been studied in graphs beyond the ring (and other constant degree graphs). In sparse graphs, such as planar graphs, an algorithm based on Nash-Williams decompositions computes a MIS in a sublogarithmic number of communication rounds [17]. On trees [87] a MIS can be computed in time $O(\sqrt{\log n \log\log n})$. In general graphs the simple and elegant randomized algorithm by Luby [90] with running time of $O(\log n)$ (see as well [1, 62]) outperforms for $\Delta \in \Omega(\log n)$ the fastest known deterministic distributed algorithms, i.e. $O(\Delta + \log^* n)$ [14] and $O(n^{\sqrt{c/\log n}})$ with constant c[97]. In Luby's algorithm neighbors try to enter the MIS based on their degrees, the deterministic algorithm uses network decompositions introduced in [9]. In general graphs every algorithm requires at least $\Omega(\sqrt{\log n/\log\log n})$ or $\Omega(\log\Delta/\log\log\Delta)$ communication rounds for computing a MIS [77]. Recently, the bounds have been improved to $\Omega(\sqrt{\log n})$ or $\Omega(\log\Delta)$ [78].

For geometric graphs the relevant lower bound is by Linial [88]. He showed that even on a ring topology at least time $\Omega(\log^* n)$ is required to compute a MIS. Only very recently (and probably stirred by the interest in wireless networking) MIS algorithms for geometric graphs (such as UDG or BIG) have been discovered. The currently fastest randomized algorithm by Gfeller and Vicari [43] runs in $O(\log\log n \cdot \log^* n)$ time. Using randomization a set S is created, where every node $v \in S$ has at most $O(\log^5 n)$ neighbors in S. Thereafter the fastest deterministic algorithm up to now [73] with time complexity $O(\log \Delta \cdot \log^* n)$ is used, which computes an $O(\log \Delta)$-ruling independent set in time $O(\log \Delta \cdot \log^* n)$ in the first phase. (For an α-ruling independent set, every two nodes in the set have distance at least two and any node not in the set has a node in the set within distance α.) The $O(\log \Delta)$-ruling independent set is transformed into an 3-ruling independent set, and this set in turn is taken to compute a MIS using again the deterministic coin tossing technique. Our algorithm is not only exponentially faster than the state-of-the-art [43], it is also deterministic, and last not least simpler. Thanks to Linial's lower bound we know that it is asymptotically optimal. If a node knows the distances to its neighbors, then [76] also achieves asymptotic optimality for computing a MIS in BIG. The main idea is to maintain a set of active nodes (initially, all nodes are active) and only consider edges between active nodes of distance at most r. In every iteration the radius r is doubled (up to $1/2$) and a MIS is computed on all active nodes. Only nodes in the MIS stay active. The degrees in the considered graphs are small and thus a MIS can be computed efficiently (via a coloring). It is rather surprising that we can match the bound of [76] without any distance information.

For symmetry breaking a number of techniques exist beyond the previously mentioned. In [17, 13, 97] network decompositions are used, i.e. a graph is partitioned into subgraphs and then in each subgraph (one after the other or in parallel) the problem (i.e. coloring [13] and MIS [97] using ruling sets) is solved. For defective colorings [14, 72, 15] several nodes initially choose the same color. However, through multiple iterations the number is reduced until a proper coloring is achieved. Other algorithms [1, 62, 90] are also of iterative nature and let each node attempt to join the MIS with some probability in every round, or randomly choose a color until it is distinct from all neighbors. Sometimes several schemes are combined, e.g. [69, 15]. Our approach can be seen as an improvement of the iterative technique. All problems studied in this work are simple in a centralized setting and all allow for straight-forward sequential greedy algorithms running in linear time. However, coming up with sub-linear algorithms is not easy, even for a seemingly simple problem such as an $O(\Delta)$ coloring in the weak message passing model, where a node can concurrently send and receive a (distinct) message to each of its neighbors. Deterministically, a $\Delta+1$ coloring is achievable in time $O(\Delta^2+\log^* n)$

[44] or in time $O(\Delta + \log^* n)$ [14, 72] or in time $O(n^{O(1)/\sqrt{\log n}})$ [97]. Using randomization, it is computable in time $O(\log n)$ [90, 1].[2] In [90] every node iteratively picks a random color of all still available colors and keeps it, if no neighbor has chosen the same color. For growth-bounded graphs (GBG)[74], where the size of a maximum independent set within distance r is bounded by $f(r)$ (e.g. Unit Disk Graphs), only time $\Theta(\log^* n)$ is needed to compute a $\Delta + 1$ coloring [106]. For arbitrary graphs, an $O(\Delta)$ coloring can be computed in a randomized way in $O(\sqrt{\log n})$ [69]. Using $O(\Delta^{1+o(1)})$ colors [15] gives an algorithm running in time $O(f(\Delta) \log \Delta \log n)$ where $f(\Delta) = \omega(1)$ is an arbitrary slow growing function in Δ. Using $O(\Delta^2)$ colors [88] gives a deterministic algorithm running in time $O(\log^* n)$ and also a lower bound of time $\Omega(\log^* n)$ for three coloring of an n-cycle. The upper bound iteratively computes an $O(\Delta^2 \log^2 k)$ coloring [36], where k is the number of colors used in the current coloring. An algorithm with less complex local computation but same time complexity (in terms of communication rounds) is stated in [16]. In [88] it is also proven that even for d-regular trees any algorithm running in time $O(\log_d n)$ uses at least $\Omega(\sqrt{d})$ colors.

A lower bound of $\Omega(\Delta/\log^2 \Delta + \log^* m)$ communication rounds to obtain an $O(\Delta)$ for special algorithms, i.e. deterministic iterative algorithms that assign a new color to a node in every round based only on the (current) colors of its neighbors, is given in [80]. Also for variants of greedy algorithms [42] several lower and upper bounds have been derived. Most interestingly, a lower bound stating that any distributed algorithm choosing computing a greedy coloring requires time at least $\Omega\log n/\log\log n$.

Overall $\Delta + 1$ coloring has probably attracted more attention than employing $O(\Delta)$ or more colors. Using less than $\Delta + 1$ colors is not possible for complete graphs – not even in a centralized setting, where the entire graph is known. An algorithm in [66] parallelizes Brooks' sequential algorithm to obtain a Δ coloring from a $\Delta + 1$ coloring. In a centralized setting the authors of [6] showed how to approximate a three-colorable graph using $O(n^{0.2111})$ colors. Some centralized algorithms iteratively compute large independent sets, e.g., [18]. It seems tempting to apply the same ideas in a distributed setting, e.g., a parallel minimum greedy algorithm for computing large independent sets is given in [53]. It has approximation ratio $(\Delta + 2)/3$. However, the algorithm runs in time polynomial in Δ and logarithmic in n and thus is far from efficient. For some restricted graph classes, there are algorithms that allow for better approximations in a distributed setting. A Δ/k coloring for $\Delta \in O(\log^{1+c} n)$ for a constant c with $k \leq c_1(c) \log \Delta$ where constant c_1 depends on c is given in [46]. It works for quite restricted graphs

[2] In [92] it is claimed that an $O(\Delta)$ coloring is computed in time $O(\log^*(n/\Delta))$. However, there is an error in the analysis pointed out by [80], increasing the number of colors to $O(\Delta^2)$ and time to $O(\log^* n)$.

(only), i.e., graphs that are Δ-regular, triangle free and $\Delta \in O(\log^{1+c} n)$. Throughout the algorithm a node increases its probability to be active. An active node chooses a color uniformly at random. The algorithm runs in $O(k + \log n/\log \Delta)$ rounds. Constant approximations of the chromatic number are achieved for growth bounded graphs (e.g. unit disk graphs) (i.e., our algorithm in Chapter 5) and for many types of sparse graphs [13]. In [19] the existence of graphs of arbitrarily high girth was shown such that $\chi \in \Omega(\Delta/\log \Delta)$. Since graphs of high girth locally look like trees and trees can be colored with two colors only, this implies that coloring is a non-local phenomenon. Thus, a distributed algorithm that only knows parts of the graph and is unaware of global parameters such as χ, has a clear disadvantage compared to a centralized algorithm.

Edge and vertex coloring are closely related, i.e. a vertex coloring algorithm on a line graph of G corresponds to an edge coloring of G. In the line graph of G all edges in G are nodes and two nodes are adjacent, if their corresponding edges share a common vertex in G. The randomized edge coloring in [45] is efficient for graphs of relatively large degree, i.e. $\Delta \in \Omega(n^{c/\log\log n})$, it runs in time $\mathrm{O}(\log\log n)$ using $\mathrm{O}(\Delta)$ colors. For graphs of smaller degree but still much larger than polylogarithmic, i.e. $\Delta \in \Omega(n^{1/\sqrt{\log n}})$ it requires time $\mathrm{O}(\sqrt{\log n})$.

An (α, β)-ruling set [9] induces a network decomposition, such that any component has diameter at least α and at most β. In [9] it is shown how to compute a $(k, k\log n)$-ruling set in time $\mathrm{O}(k \log n)$. In [43] a $(1, \log\log\Delta)$-ruling set is computed in time $\mathrm{O}(\log\log\Delta)$ such that each node in the ruling set has at most $\mathrm{O}(\log^5 n)$ neighbors also in the ruling set. We target sparser sets, i.e. independent ruling sets. However, different quality measures exist, e.g. in [8] several types of network decompositions (and covers) are considered. Our strength lies primarily in getting network decompositions of balanced and small (sub-logarithmic) diameter efficiently. Our notion is in particular of interest if the main concern is equivalent (up to constant factors) communication time among all nodes within each cluster.

Chapter 3

Multi-Trials Technique

3.1 Introduction

So far, per message exchange all techniques for computing a maximal independent set (MIS) or a coloring have performed one attempt to obtain either a color or to join the MIS. Our technique transcends this approach and allows to perform multiple trials per communication round, hence Multi-Trials. Our constant time coloring algorithm shows how to get a color by approximating the result of multiple communication rounds through one communication round and local computation. Our MIS algorithm increases the number of possibilities a node can choose from to break symmetry from one (i.e. join the MIS or not) to many.

The application of our technique leads to a variety of interesting findings. For an overview of the results related to coloring see Figure 3.1. We also apply the Multi-Trials method to compute ruling sets. A set is (α, β)-ruling if every two nodes in the set have distance at least α and any node not in the set has a node in the set within distance β. Ruling sets are a natural way to obtain network decompositions, i.e. each node attaches itself to a closest node in the ruling set. Network decompositions in turn are fundamental to exploit the locality of a problem. That is to say, to efficiently distribute a task to several components of the network, which typically solve a subproblem and later combine the partial solutions. What kind of decomposition is best, depends on the task at hand. However, the diameter of a network component is usually an important parameter, since it determines the time for the exchange of information among all nodes in a component. In many cases it is natural to ask for components of equal diameter. We are the first to achieve network decompositions of similar diameter, i.e. of constant ratio of α and β, in sub-logarithmic time. We

Previous work		This work	
Colors	Time	Colors	Time
$\Delta + 1$	$O(\log n)$ [90, 1, 63]	$\Delta + 1$	$O(\log \Delta + \sqrt{\log n})$
	$O(\Delta + \log^* n)$ [72, 14]		
$O(\Delta)$	$O(\sqrt{\log n})$ [69]	$O(\Delta + \log n)$	$O(\log \log n)$
$O(\Delta + n^d)$	$O(\log \log n)$ [45]	$O(\Delta + \log^{1+1/\log^* n} n)$	$O(\log^* n)$
$O(\Delta \log^2 n)$	$O(1)$ [80]	$O(\Delta \log^{(c)} n + \log^{1+1/c} n)$	$O(1)$
$O(\Delta^{1+o(1)})$	$O(\log \Delta \log n)$		
$O(\Delta^2)$	$O(\log^* n)$ [88, 92]		

Table 3.1: Comparison of coloring algorithms, where c is an arbitrary constant and $d := O(1/\log \log n)$

describe deterministic and randomized variants of an algorithm trading time for distance to a node in the ruling set. For example, we give a randomized algorithm computing a $(2k, 2k(c+1))$-ruling set in time $O(k \cdot 2^c \cdot \log^{1/c} n)$ for any integers $k, c > 0$. Our novel approach is based on partitioning the ID of a node. Our algorithm uses each partition as a separate trial.

By definition a MIS is the same as a (2,1)-ruling set. For graphs of sub-logarithmic degree our method is an improvement of prior randomized algorithms for the MIS problem. For graphs, where the size of a maximum IS within distance r of a node is bounded by $f(r)$, we are the first to achieve running time linear in $f(r)$.

After stating the technique and using it for three problems, we give a theoretical and a brief experimental analysis in the last two sections of this chapter.

3.2 Technique

We apply our Multi-Trials technique to solve coloring and MIS problems as well as to compute ruling sets. We show several ways how to allow for many trials per communication round, even when only few seem possible.

3.2.1 Coloring

Instead of randomly choosing a single color and exchanging the color with its neighbors, a node gives a (random) preference for each color and transmits all its preferences at once, see Algorithm *ColorTrials*. A preference is a random number in $[0, \Delta_{N(v)}]$ for a node v, where $\Delta_{N(v)}$ is the size of the largest neighborhood of a neighbor $u \in N(v)$. This implies that the number of

Algorithm ColorTrials(Available colors $C(v)$ for node v)

1: $S(v) := \{(c, r)|\forall c \in C(v)$ choose a number $r \in [0, \Delta_{N(v)}]$ uniformly at random)$\}$
2: Transmit $S(v)$ to all uncolored neighbors $u \in N(v)$
3: **for each** $(c_v, r_v) \in S(v)$ **do**
4: **if** $r_v > \max\{r|(c_v, r) \in S(u), u \in N(v)\}$ **then** $color(v) := c_v$ **end if**
5: **end for each**

Algorithm ConstDeltaColoring, i.e. $(1+\epsilon)\Delta$ for $\epsilon > 1/2^{\log^* n}$

1: $color(v) := none$
2: $C(v) := \{0, 1, ..., (1+\epsilon)\Delta_{N(v)} + \log^{1+1/\log^* n} n\}$
3: **repeat**
4: *ColorTrials*$(C(v))$
5: $N(v) := \{u|u \in N(v) \wedge color(u) = none\}$
6: $C(v) := C(v) \setminus \{color(u)|u \in N(v)\}$
7: **until** $color(v) \neq none$

selectable preferences for a color is always at least as large as the number of preferences chosen for a color by the neighbors of node v. This ensures that the probability that node v has a neighbor with the same random preference for a color is small (i.e. constant). If node v's preference is unique and largest for a color, it can take the color. However, it only keeps one (arbitrary) color, it is allowed to take. A colored node informs its neighbors about its obtained color and stops Algorithm *ColorTrials*. Colored nodes and their incident edges are removed from the graph $G = (V, E)$ and, therefore, $N(v)$ denotes all uncolored neighbors upon calling *ColorTrials*.

Algorithms *ConstDeltaColoring* and *ConstTimeColoring* simply repeat *ColorTrials*. The only difference is that for constant time the number of initially available colors is larger, i.e. $\{0, 1, ..., \Delta_{N(v)} \log^{(c)} n + \log^{1+1/c} n\}$ for some constant c and some node v. For both algorithms the number of unused colors, i.e. colors not taken by a neighbor, is always larger than logarithmic. Thus any node performs more than a logarithmic number of trials to get a color in any communication round. For computing a $\Delta + 1$ coloring, Algorithm *DeltaPlus1Coloring* also iteratively calls Algorithm *ColorTrials* with an initial set of colors $C(v) := \{0, 1, ..., \Delta\}$ until only few unused colors left (more precisely, $|C(v)| < \sqrt{\log n}$). The remaining uncolored nodes can be colored by our MULTI-TRIALS technique as shown in Algorithm *RankingTrials* in Section 3.2.2 (or defective colorings[72, 14]).

For the sake of simplicity, we have described the algorithms using large

Algorithm CoordinateTrials (maxColor d, parameter c)

1: $j := 0$; $coord(v)[i] :=$ the number given by bits $[i(\log d)/c, (i+1)(\log d)/c]$ of $color(v)$
2: **repeat**
3: $Rank(v) := [x_0, x_1, \ldots, x_{c-1}]$, s.t. $(x_i = 1 \Leftrightarrow coord(v)[i] = j) \wedge (x_i = 0 \Leftrightarrow coord(v)[i] \neq 0)$
4: **for** $k = 1..2^c$ **do**
5: **if** $Rank(v) = k$ **then** Transmit *Stop*
6: **elsif** received Stop **then** Exit **end if**
7: **end for**
8: $j := j + 1$
9: **until** $\nexists u \in N_+(v)$ executing algorithm
10: Join the ruling set

Subroutine IncProb(threshold t)

1: Transmit p_v to all nodes $u \in N^2(v)$
2: **while** $\forall u \in N_+(v) | \sum_{w \in N_+(u)} p_w < 1/t$ **do**
3: $p_v := p_v \cdot t$
4: Transmit p_v to all nodes $u \in N^2(v)$
5: **end while**

messages. In the Analysis Section we show that messages of polylogarithmic size suffice.

3.2.2 Algorithm RankingTrials

In Algorithm *RankingTrials* we assume that each node has at most $\Delta \leq \sqrt{\log n}$ uncolored neighbors. The goal is to color all nodes with $\Delta + 1$ colors. To make effective use of the Multi-Trials technique, we must ensure that a node can perform many trials in one communication round, i.e. even if Δ is much smaller than $\sqrt{\log n}$ a node should perform at least $\sqrt{\log n}$ trials per round. One idea is to create a ranking of the nodes using $2\sqrt{\log n} + 1$ ranks, i.e. each node can try to get any of the at least $\sqrt{\log n} + 1$ ranks that are not used by any of its neighbors. Then nodes pick a color depending on this ordering, i.e. once every node v has a rank $Rank(v)$, the uncolored node with smallest rank among its neighbors chooses a color until all nodes are colored. However, it is possible to perform even more trials per round, i.e. in order to obtain a rank, a node must obtain a priority $RP \in [0, 2\sqrt{\log n}]$ for that rank. The rank priorities RP determine which rank a node gets. Thus, a

Algorithm RandRulingSet(parameter c)

1: $p_v := 1/n$; $color(v) := none$
2: **for** $i = 1..c$ **do** IncProb($2^{(\log n)^{1-i/c}}$) **end for**
3: Participate in computing an $O(\log^{1+1/\log^* n} n)$ coloring with probability $\min(1, 64p_v \log n)$
4: **if** $color(v) \neq none$ **then**
5: CoordinateTrials($O(\log^{1+1/\log^* n} n)$,$c+1$) **end if**

node could perform about $4 \log n$ trials per round. However, to keep messages small we perform fewer trials (but each having higher success probability). A node must get a rank priority for at least $\sqrt{\log n} + 1$ ranks and all rank priorities must be distinct. After obtaining the rank priorities, starting from smallest priority 0 on to priority $2\sqrt{\log n}$ a node keeps the first rank, where its priority is smallest (among its neighbors) and which is not taken by a neighbor.

3.2.3 Ruling Set And Maximal Independent Set

A simple but slow algorithm to compute a ruling set lets a node v with color (or ID) i join the ruling set in the ith round if none of its neighbors $u \in N(v)$ is already in the set. To speed up the process, we split up the digits of a node's color into c equal parts (with the same number of digits), i.e. $coord(v)[0], ..., coord(v)[c-1]$, to perform multiple trials. A node v computes a summary of all trials of an attempt to join the ruling set, i.e. a rank $Rank(v)$ consisting of c bits, where bit i is 1 if and only if the ith coordinate equals the current attempt j, i.e. $coord(v)[i] = j$, and 0 otherwise. Based on the rank a node either continues the algorithm (and eventually joins the ruling set) or stops. After a computation of a rank all nodes with rank larger 0 compete to continue and force other nodes to stop the algorithm. More precisely, a node v continues and forces its neighbors with distinct rank to exit the algorithm, if in the kth round of the competition its $Rank(v)$ equals k.

Algorithm *CoordinateTrials* with arguments d and c requires a coloring for a subset $U \subseteq V$ of nodes using colors $\{0, 1, \ldots, d-1\}$. It determines a ruling set with at least one node within distance c for each colored node. A color $color(v) \in [0, d-1]$ of a node v is seen as a point in the c dimensional space $[0, d^{1/c} - 1]^c$, i.e. $color(v) = (coord(v)[0], ..., coord(v)[c-1])$. In the Analysis Section we describe modifications of the algorithm trading communication time for distance to a node in the ruling set.

To compute a $(2,c)$-ruling set deterministically, we start from an initial $O(\Delta^2)$[88] or $O(\Delta^{1+o(1)})$ [15] coloring before calling Algorithm *CoordinateTrials*. Our randomized Algorithm *RandRulingSet(c)* computes a subset of nodes $U \subseteq V$ which gets colored as follows: Each node v starts with a small value $p_v = 1/n$. This value is raised repeatedly by calling Subroutine *IncProb(t)* until the node v or a neighbor satisfies the condition that the sum of the values p_w of nodes $w \in N_+(v)$ exceeds one half. Then with probability $\min(1, 64p_v \log n)$ a node joins set U, i.e. participates in computing an $\mathrm{O}(\log^{1+1/\log^* n} n)$ coloring. Afterwards Algorithm *CoordinateTrials*($O(\log^{1+1/\log^* n} n)$,*c)* is called by every colored node.

To compute a MIS for graphs where the maximum size of an independent set within distance r for every node v is bounded by $f(r)$ one simply iterates algorithm *RandRulingSet(c)* to compute a ruling set R_i in iteration i. After the ith iteration all nodes from the ruling set R_i join the MIS and all neighbors $u \in N_+(v), v \in R_i$ are removed from the graph. Due to Theorem 21, in $\mathrm{O}(2^c \log^{1/c} n)$ time, for every node v at least one node $u \in N_+^{2(c+1)}(v)$ joins the MIS. Since, the size of any MIS within distance $2(c+1)$ is at most $f(2(c+1))$, the time complexity to compute a MIS is $\mathrm{O}(f(2(c+1)) \cdot 2^c \log^{1/c} n)$.

3.3 Theoretical Analysis

3.3.1 Notation and Prerequisites

Recall that $G = (V, E)$ is the graph given by all *uncolored* nodes before calling *ColorTrials*. $\tilde{N}(v) \subseteq N(v)$ denotes the still uncolored neighbors of node v during the foreach loop of *ColorTrials* and $\tilde{d}(v) := |\tilde{N}(v)|$. Let $E(v,c) = x$ denote the event for node v that x uncolored nodes $u \in \tilde{N}_+(v)$ take color $c \in C(v)$ during the execution of *ColorTrials*. The probability of event $E(v,c) = x$ is $Pr(E(v,c) = x)$. We also use the following Chernoff bound:

Theorem 1. *The probability that the number X of occurred independent events $X_i \in \{0,1\}$, i.e. $X := \sum X_i$, is less than $(1-\delta)$ times the expectation $\mathbb{E}[X]$ can be bounded by $Pr(X < (1-\delta)\mathbb{E}[X]) < e^{-\mathbb{E}[X]\delta^2/2}$.*

Observe that the two events $E(u,c)$ and $E(v,c)$ are dependent. For instance, consider four nodes u, v, w, x in a line, i.e. with edges (u,v), (v,w), (w,x), and assume that nodes u and x already obtained a color before calling *ColorTrials*. In this case, for event $E(u,c) > 0$ to occur node v's random number must be larger than w's and the other way round for event $E(x,c) > 0$ to happen. Thus, it is not possible that both events $E(u,c) > 0$ and $E(x,c) > 0$ occur, i.e. $Pr(E(u,c) > 0 | E(x,c) > 0) = 0$, since either v's random number is larger than w's or the other way round or both are equal. In general, the correlation might also be positive, e.g. if nodes form a clique.

To deal with the dependencies among nodes, we derive a bound for n (slightly) dependent events. It follows directly from the union-bound.

Theorem 2. *For n^{k_0} (dependent) events E_i with $i \in \{0, ..., n^{k_0} - 1\}$, such that each event E_i occurs with probability $Pr(E_i) = 1 - \sum_{F \in S_{E_i}} Pr(F) \geq 1 - 1/n^k$, the probability that all events occur is at least $1 - 1/n^{k-k_0}$.*

Proof. For each E_i, $Pr(\text{not } E_i) \leq 1/n^k$. Hence $Pr(\exists i \text{ s.t. not } E_i) \leq (1/n^k) \cdot n^{k_0} = 1/n^{k-k_0}$. Therefore, $Pr(\text{all } E_i \text{ occur}) \geq 1 - 1/n^{k-k_0}$. □

We also give a second (slightly weaker) theorem, having a more basic proof.

Theorem 3. *For n (dependent) events E_i with $i \in \{0, ..., n-1\}$, such that each event E_i occurs with probability $Pr(E_i) = 1 - \sum_{F \in S_{E_i}} Pr(F) \geq 1 - 1/n^k$, the probability that all events occur is at least $1 - 1/n^{k-3}$.*

Let Ω denote the union of all possible (elementary) events F, i.e. $\sum_{F \in \Omega} Pr(F) = 1$. For an event E_i to occur with $0 \leq i \leq n-1$, let certain elementary events $S_{E_i} \subseteq \Omega$ be infeasible for event E_i, i.e. $Pr(E_i | F \in S_{E_i}) = 0$, and let all others $\Omega \setminus S_{E_i}$ cause E_i to happen, i.e. the probability $Pr(E_i)$ of event E_i is $\sum_{F \in \Omega \setminus S_{E_i}} Pr(F) = \sum_{F \in \Omega} Pr(F) - \sum_{F \in S_{E_i}} Pr(F) = 1 - \sum_{F \in S_{E_i}} Pr(F)$.

Proof. We want to show that $Pr(\wedge_{i \in \{0,...,n-1\}} E_i) \geq 1 - 1/n^{k-3}$. Since $Pr(\wedge_{i \in \{0,...,n-1\}} E_i) = Pr(E_0) \cdot Pr(E_1 | E_0) \cdot Pr(E_2 | E_0 \wedge E_1) \cdot ... \cdot Pr(E_{n-1} | \wedge_{i \in \{0,...,n-2\}} E_i)$ we can also derive lower bounds for the conditional probabilities $Pr(E_i | \wedge_{i \in T, T \subset \in \{0,...,n-2\}} E_i)$. We assume a worst case correlation among events E_i, i.e. the occurrence of an event E_i has the worst impact on the probability that another event E_j occurs. Given that the event E_i occurs, all elementary events S_{E_i} for which E_i cannot happen, are known not to occur. They can be excluded from Ω, when computing the probability of another event E_j, i.e. $Pr(E_j | E_i)$. Thus, the elementary events that can occur given E_i are $\Omega_{|i} := \Omega \setminus S_{E_i}$. Since $Pr(E_j | E_i)$ is a probability distribution, the sum of the probabilities $Pr(F | E_i)$ of all elementary events $F \in \Omega_{|i}$ must be one, i.e. $\sum_{F \in \Omega_{|i}} Pr(F | E_i) = 1$. We have that $Pr(F | E_i) = Pr(F) \cdot 1/(1 - \sum_{F \in S_{E_i}} Pr(F)) \leq Pr(F) \cdot 1/(1 - 1/n^k)$, since the occurrence of event E_i only removes the set S_{E_i} with $\sum_{F \in S_{E_i}} Pr(F) \leq 1/n^k$ from Ω, but does not make one elementary event $F_1 \in \Omega_{|i}$ (relatively) more favorable to another $F_2 \in \Omega_{|i}$, i.e. the probabilities $Pr(F | E_i)$ of all remaining events $F \in \Omega_{|i}$ are increased by the same factor $1/(1 - \sum_{F \in S_{E_i}} Pr(F)) \leq 1/(1 - 1/n^k)$. To

compute $Pr(E_j|E_i)$ assuming a worst case correlation, all excluded elementary events S_{E_i} cause E_j to occur and $S_{E_i} \cap S_{E_j} = \{\}$, i.e. all elementary elements S_{E_j} are excluded for $Pr(E_j|E_i)$ to occur. Thus, all elementary events $\Omega_{|i} \setminus S_{E_j}$ cause E_j to occur and $Pr(E_j|E_i) = 1 - \sum_{F \in S_{E_j}} Pr(F|E_i) \geq 1 - \sum_{F \in S_{E_j}} Pr(F) \cdot 1/(1-1/n^k) \geq 1 - 1/n^k \cdot 1/(1-1/n^k) = 1 - 1/(n^k - 1)$. The argument can be generalized to bound any conditional probability $Pr(E_j| \wedge_{i \in T, T \subseteq \{1,\ldots,n-1\}} E_i)$ for any $0 \leq j \leq n-1$. The remaining possible events for E_j to occur given all events E_i with $i \in T$ happen is $\Omega_{|T} := \Omega \setminus \cup_{i \in T} S_{E_i}$. For the probability of an elementary event $F \in S_{E_j}$ holds $Pr(F| \wedge_{i \in T} E_i) = Pr(F) \cdot 1/(1 - \sum_{F \in \cup_{i \in T} S_{E_i}} Pr(F)) \leq Pr(F) \cdot 1/(1 - |T|/n^k)$. The last inequality follows since all sets S_{E_i} are assumed to be disjunct, i.e. $S_{E_i} \cap S_{E_l} = \{\}$, to maximize the probability that an elementary event $F \in S_{E_j}$ occurs . Therefore, $Pr(E_j| \wedge_{i \in T} E_i) = 1 - \sum_{F \in S_{E_j}} Pr(F| \wedge_{i \in T} E_i) \geq 1 - \sum_{F \in S_{E_j}} Pr(F) \cdot 1/(1 - |T|/n^k) = 1 - 1/(1 - |T|/n^k) \sum_{F \in S_{E_j}} Pr(F) \geq 1 - 1/(1 - n/n^k) \cdot 1/n^k = 1 - 1/(n^k - n)$.

Using the bound of the conditional probabilities and $Pr(E_i) \geq 1 - 1/n^k \geq 1 - 1/(n^k - n)$ (the first inequality is by assumption), we obtain: $Pr\left(\wedge_{i \in \{0,\ldots,n-1\}} E_i\right) = Pr\left(E_0\right) \cdot Pr\left(E_1|E_0\right) \cdot Pr\left(E_2| \wedge_{i \in \{0,1\}} E_i\right) \cdot Pr\left(E_3| \wedge_{i \in \{0,1,2\}} E_i\right) \cdot \ldots \cdot Pr\left(E_{n-1}| \wedge_{i \in \{0,\ldots,n-2\}} E_i\right) \geq \prod_{i \in \{0,\ldots,n-1\}} \left(1 - 1/(n^k - n)\right) = \left(1 - 1/(n^k - n)\right)^n \geq 1 - 1/n^{k-3}$ □

Moreover, the probability $Pr(E(v,c) > 0)$ that a node v or some of its uncolored neighbors $\tilde{N}(v)$ get a certain color $c \in C(v)$ depends on the topology of the graph, i.e. on each neighborhood $N(u)$ of every uncolored node $u \in \tilde{N}(v)$. In contrast to the probability $Pr(E(v,c) > 0)$, the probability that an uncolored node u gets color $c \in C(u)$ does not depend on the current number of uncolored neighbors $\tilde{d}(u)$, but only depends on $d(u)$, i.e. the number of uncolored neighbors before executing *ColorTrials*. This is because u is unaware whether a neighbor $w \in N(u)$ got a color $c_1 < c$ while executing *ColorTrials*. Thus, node u must consider all random numbers from all nodes $w \in N(u)$ and must be larger than all of them, even those that already got a smaller color.

Since $\tilde{d}(v)$ might change after every considered color $c \in C(v)$, two events $E(v,c_1)$ and $E(v,c_2)$ are *not* independent for arbitrary colors c_1, c_2. For example, consider a star graph with v in the center, i.e. a tree with v as root and $n-1$ leaves. Say c_1 is the first considered color and c_2 is the second. Assume that all initially uncolored neighbors $N(v)$ of v get the first color c_1. Then the probability of event $E(v,c_2) > 0$ is roughly $1/|N_+(v)| = 1/n$,[1]

[1] $Pr(E(v,c_2) > 0) = 1/n$ would hold if all nodes were known to draw distinct random numbers.

since node v's random number for color c_2 must be larger than the choice of all its neighbors $N(v)$ executing *ColorTrials*. If none of v's neighbors got the first color, the only situation where $E(v, c_2)$ equals 0, is when v drew a random number that is maximum among all nodes $u \in N(v)$ and at least one neighbor chose the same number. This happens with probability less than $1/|N_+(v)| = 1/n$. Thus, in this case, a lower bound of $Pr(E(v, c_2) > 0)$ is $1 - 1/|N_+(v)| = 1 - 1/n$. Therefore, the probability of $Pr(E(v, c_2) > 0)$ depends on the outcome of the first color $E(v, c_1)$.

To deal with the interdependence for different colors we follow the idea of *stochastic domination*. A probability distribution A, where $Pr_A(X = x)$ denotes the probability of outcome x, dominates a probability distribution B, if for any outcome x, A gives a higher probability of receiving an outcome equal to or better than x under B, i.e. $Pr_A(X \geq x) \geq Pr_B(X \geq x)$. More precisely, we use the following basic theorem.

Theorem 4. *For t dependent events $E_i \in \{0, 1\}$ with $i \in [0, t-1]$, such that each event E_i occurs with probability $Pr(E_i = 1 | \sum_{j=0}^{i} E_j \leq y) \geq p$, the probability that at least $\min(y, tp/2)$ events occur is at least $1 - e^{-tp/8}$.*

Proof. Consider the random variable $X_E = \sum_{j=0}^{t} E_j$. We want to compute $Pr(X_E \leq y)$. Consider any event F, i.e. a sequence of events $(E_0 = x_0, E_1 = x_1, ..., E_t = x_t) =: F$, with $X_E \leq y$. For any such event F holds that each event $E_i = 1$ occurs with probability at least p independently of all others, since by assumption $Pr(E_i = 1 | X_E \leq y) \geq p$ for all $i \in [0, t-1]$. Consider the random variable $X_I = \sum_{j=0}^{t} X_j$ which is the sum of t independent events X_i such that each event occurs with probability p. We have $Pr(X_E \geq x) \geq Pr(X_I \geq x)$ for all $x \leq y$, since $Pr(E_i = 1 | X_E \leq y) \geq p = Pr(X_i = 1)$ for all $i \in [0, t-1]$. Thus, to upper bound $Pr(X_E \leq \min(y, tp/2))$ we can use $Pr(X_I \leq \min(y, tp/2))$. Using Theorem 1 with $\delta = 1/2$ and $\mathbb{E}[X_I] = tp$ yields a bound $Pr(X_I \leq tp/2) < e^{-tp/8}$. Therefore, $Pr(X_E \leq \min(y, tp/2)) \leq Pr(X_I \leq \min(y, tp/2)) \leq Pr(X_I \leq tp/2) < e^{-tp/8}$. □

In our case the probability of a certain outcome is defined by Algorithm *ColorTrials* and the given graph G. To derive bounds for the probabilities of events $Pr(E(v, c) = x)$ of a node v we consider a special topology $G'_v = (V', E')$ which is dominated by G, meaning that it is more likely for node v to get a certain number of colored neighbors in G than in G'_v. For such a graph $G'_v = (V', E')$ all terms $E'(v, c)$, $\tilde{d}'(v)$, $N'(v)$ etc. are defined in the same manner as for G.

3.3.2 Analysis of Algorithm ColorTrials

Given a node $v \in V$, let $G'_v = (V', E')$ be the graph obtained by enhancing G, i.e. we begin with $V' = V$ and $E' = E$ and add edges and nodes to G'_v. If $|V'| < \Delta + d(v) + 1$ then add an arbitrary set S of nodes to V', i.e. $V' = V \cup S$ such that $|V'| = \Delta + d(v) + 1$. Each node $u \in N(v)$ is connected to all other nodes $w \in N(v) \setminus u$, i.e. the neighbors $N(v)$ form a clique. Furthermore, each node $u \in N(v)$ is connected to Δ arbitrary nodes $V' \setminus N_+(v)$, i.e. node u has degree $\Delta + d(v)$.

Thus node v has the same neighbors in G and G'_v, i.e. $N(v) = N'(v)$. But the degree of a node $u \in N'(v)$ in G'_v is larger than that of any node $w \in N(v)$ in G and all neighbors $u \in N'(v)$ are connected among themselves. Note, that G'_v is defined for a single node $v \in V$ and can be seen as a worse topology than G for node v.

Lemma 5. *For a node v in a graph $G'_v = (V', E')$ we have that $Pr(\sum_{c \in C(v)} E(v,c) > y) \geq Pr(\sum_{c \in C(v)} E'(v,c) > y)$ for any integer $y \geq 0$.*

Proof. Since node v's neighbors form a clique in G'_v only one node $u \in N_+(v)$ can get a color $c \in C(v)$. Thus $E'(v,c) \in \{0,1\}$, i.e. $Pr(E'(v,c) = i) = 0$ for $i > 1$.[2] Thus, $Pr(E(v,c) > i) \geq Pr(E'(v,c) > i) = 0$ holds $\forall 1 \leq i \leq n$. We show that it also holds for $i = 0$, i.e. $Pr(E(v,c) > 0) \geq Pr(E'(v,c) > 0)$ given $\tilde{d}(v) = \tilde{d}'(v)$. Since $Pr(E'(v,c) > 0) = 1 - Pr(E'(v,c) = 0)$ we have $Pr(E(v,c) > 0) \geq Pr(E'(v,c) > 0) \Leftrightarrow 1 - Pr(E(v,c) = 0) \geq 1 - Pr(E'(v,c) = 0) \Leftrightarrow Pr(E(v,c) = 0) \leq Pr(E'(v,c) = 0)$. To upper bound $Pr(E(v,c) = 0)$ in G, we use the observation that the larger the neighborhood $N(u)$ of a neighbor $u \in N(v)$ (with nodes $w \in N(u)$ being at distance 2 from v), the lower are the chances for u to get colored and, therefore, the higher are the chances for $Pr(E(v,c) = 0)$. This is because u's random number must be larger than that of all its neighbors $w \in N(u)$ for u to get colored and event $E(v,c) > 0$ to occur. Thus, we assume that each neighbor $u \in N(v)$ has Δ neighbors $w \in N(u)$ at distance 2 from v. The probability of $Pr(E(v,c) = 0)$ does not decrease, if, additionally, nodes $u \in N(v)$ are interconnected. Thus we assume that all nodes $u \in N(v)$ form a clique. The described topology giving an upper bound for $Pr(E(v,c) = 0)$ is the topology of v for G'_v used to compute $Pr(E'(v,c) = 0)$ and thus $Pr(E(v,c) = 0) \leq Pr(E'(v,c) = 0)$ and we have shown $Pr(E(v,c) > i) \geq Pr(E'(v,c) > i)$ holds $\forall 0 \leq i \leq n$ given $\tilde{d}(v) = \tilde{d}'(v)$.

Therefore, the number of uncolored nodes $\tilde{d}(v)$ in G decreases at least as fast as $\tilde{d}'(v)$ in G'_v with every considered color c for $\tilde{d}(v) = \tilde{d}'(v)$. When

[2] In contrast, for an event $E(v,c)$ we have that it might also have a non-zero probability that more than one node gets colored, e.g. for a tree $Pr(E(v,c) = x) > 0$ for $x \in \{0, 1, ..., \tilde{d}(v)\}$.

looking at the first color we have $d(v) = \tilde{d}(v) = \tilde{d}'(v)$. Thus $\tilde{d}(v)$ is expected to be less or equal $\tilde{d}'(v)$ after examining all colors $c \in C(v)$, i.e. for the number of newly colored nodes (after the last color $c \in C(v)$) holds $Pr\left(d(v) - \tilde{d}(v) > y\right) \geq Pr\left(d(v) - \tilde{d}'(v) > y\right)$. The argument is analogous to the following: Given two functions $f(x)$ and $g(x)$ with $f(0) = g(0)$ and $f'(x) \leq g'(x) \leq 0$ for $x \geq 0$. Then $f(y) \leq g(y)$ for any $y \geq 0$.

Since for the number of newly colored nodes holds: $d(v) - \tilde{d}(v) = \sum_{c \in C(v)} E(v,c)$, we have: $Pr\left(d(v) - \tilde{d}(v) > y\right) \geq Pr\left(d(v) - \tilde{d}'(v) > y\right) \Rightarrow Pr\left(\sum_{c \in C(v)} E(v,c) > y\right) \geq Pr\left(\sum_{c \in C(v)} E'(v,c) > y\right)$ □

Now, that we have shown that the probability distribution for the number of obtained colored neighbors given by the topology G'_v for node v (and Algorithm *ColorTrials*) is indeed dominated by the one for G (and Algorithm *ColorTrials*), we derive a bound on the probability $Pr(E'(v,c) > 0)$ for the topology G'_v for an arbitrary color $c \in C(v)$. Note, that $Pr(E'(v,c) > 0) = Pr(E'(v,c) = 1)$, since nodes $u \in N'(v)$ form a clique and thus only one node $u \in N'(v)$ can get a color c.

Lemma 6. *The probability of event $E'(v,c) = 1$ is at least $\tilde{d}'_+(v)/(24\Delta)$.*

Proof. If we choose $d'(u) = \Delta + d(v) \leq 2\Delta$ random numbers in $[0, \Delta_{N(u)}] = [0, d'(u)] \subseteq [0, 2\Delta]$ then the probability that no random number is 0 is $(1 - 1/d'_+(u))^{d'(u)} \geq 1/e$. The probability that node v or a neighbor chose one random number equal to 0 is $\tilde{d}'_+(v) \cdot 1/d'_+(u) \cdot (1 - 1/d'_+(u))^{\tilde{d}'_+(v)-1} \geq \tilde{d}'_+(v)/d'_+(u) \cdot (1 - 1/d'_+(u))^{\Delta} \geq \tilde{d}'_+(v)/(2\Delta + 1) \cdot (1 - 1/(2\Delta + 1))^{\Delta} \geq \tilde{d}'_+(v)/(3e\Delta)$.

The probability of event E_1 that for color c exactly one node $u \in \tilde{N}_+(v)$ chooses a random number in $[0, d'(u)]$ is at least $\tilde{d}_+(v)/(3e\Delta)$. The probability of the event E_2 that none of u's neighbors $w \in N(u)$ chooses a random number in this interval is $1/e$. Events E_1 and E_2 are not independent since nodes u and v have (some) common neighbors, i.e. $N(v) \cap N(u) = N(v)$. The probability $Pr(E_2|E_1)$ of event E_2 given event E_1 is at least the probability $Pr(E_2)$ of event E_2, since due to event E_1 node u is the only node in $N_+(v)$ with a random number equal 0 and all other nodes $N_+(v) \setminus u$ are known not to have a random number equal to 0. Therefore, for event E_2 to occur only the random numbers of nodes $N(u) \setminus N(v)$ instead of those of all nodes $N(u)$ must be shown to be distinct from 0. Thus the chance that both events happen is $Pr(E_1 \wedge E_2) = Pr(E_2|E_1) \cdot Pr(E_1) \geq Pr(E_2)Pr(E_1) \geq 1/e \cdot \tilde{d}_+(v) \cdot 1/(3e\Delta) \geq \tilde{d}_+(v)/(24\Delta)$. □

As quantified in Lemma 6, the more uncolored neighbors $\tilde{N}(v)$ node v has, the larger is the probability of event $E'(v,c) = 1$. As long as a node

has a certain number of uncolored neighbors we can guarantee a reasonable minimum probability for an event $E'(v,c) = 1$. This observation is used in the proof of the following theorem showing that the number of uncolored nodes reduces drastically when executing *ColorTrials* using more colors than the maximum size Δ of any uncolored neighborhood.

Theorem 7. *The probability $Pr(E_v)$ of event E_v for node v that $\tilde{d}(v) \leq \max(d(v)/2^s, \log n)$ after executing* ColorTrials *is at least $1 - 1/n^k$ for arbitrary constant k using color set $C(v)$ with $|C(v)| \geq k_0 s\Delta$ and $k_0 \geq 384(k+1)$.*

Proof. Consider a node v in a graph G'_v and a sequence of $k_0\Delta$ colors that nodes $u \in \tilde{N}_+(v)$ attempt to get during *ColorTrials*. Assume that before processing a sequence of $k_0\Delta$ the number of uncolored neighbors is $\tilde{d_0}'(v)$. In Lemma 6 the probability of $Pr(E'(v,c) = 1)$ was shown to be at least $\tilde{d}'(v)/(24\Delta)$. In particular as long as there are at most $\tilde{d_0}'(v)/2$ colors assigned the probability is at least $\tilde{d_0}'(v)/(48\Delta)$ for event $E'(v,c) = 1$, i.e. $Pr(E'(v,c) = 1 | \sum_{i=0}^{c-1} E'(v,c) \leq \tilde{d_0}'(v)/2) \geq \tilde{d_0}'(v)/(48\Delta)$. Thus we can use Theorem 4 with $t = k_0\Delta$, $p = \tilde{d_0}'(v)/(48\Delta)$ and $y = \tilde{d_0}'(v)/2 \leq tp/2$ (for k_0 sufficiently large), yielding a bound of $e^{k_0/48 \cdot \tilde{d}'_0(v)/8} = e^{k_0/384 \cdot \tilde{d_0}'(v)} \geq e^{k_0/384 \cdot \log n} = 1 - 1/n^{k_0/384}$. Therefore, the probability that for any sequence of $k_0\Delta$ colors and any initial degree $\tilde{d_0}(v) \geq \log n$ at least half of the neighbors get colored is $1 - 1/n^{k_0/384}$. For s distinct sequences the probability $Pr(E'_v)$ in G'_v that the degree of node v for $\tilde{d}'(v) > \log n$ (for each sequence) got halved $s < \log n$ times in G'_v becomes $\left(1 - 1/n^{k_0/384}\right)^s > 1 - 1/n^{k_0/384-1}$. Due to Lemma 5 the lower bound of probability $Pr(E'_v)$ is also a lower bound for the event $Pr(E_v)$ in G. □

To derive a bound for all nodes, we must take dependencies among nodes into account. The goal is to show that Theorem 7 holds for all nodes concurrently, i.e. to show that $Pr(\wedge_{v \in V} E_v) \geq 1 - 1/n^k$ for an arbitrary constant k.

Theorem 8. *All nodes $v \in V$ have $\tilde{d}(v) \leq \max(d(v)/2^s, \log n)$ uncolored neighbors with probability $1 - 1/n^k$ for arbitrary constant k after executing* ColorTrials *using color set $C(v)$ with $|C(v)| \geq k_0 s\Delta$ and sufficiently large constant k_0.*

Proof. Let an elementary event F be all random choices made by all nodes v during *ColorTrials* for a graph G, i.e. the random preferences for each node $v \in V$ for each color $c \in C(v)$. Let only the set S_{E_v} of elementary events cause the event E_v not to occur, i.e. $Pr(E_v) = 1 - \sum_{F \in S_{E_v}} Pr(F) \geq 1 - 1/n^k$, where the last inequality is due to Theorem 7. Thus we can use

Theorem 2 to bound the probability that all events E_v with $v \in V$ occur by $Pr(\wedge_{v \in V} E_v) \geq 1 - 1/n^{k-4}$. □

Let us discuss the message size. A node transmits a number of at most $O(\log \Delta)$ bits for every unused color $C(v)$ with $|C(v)| = k_0 s\Delta$ to its neighbors, yielding messages of size $O(|C(v)|s \log \Delta) = O(s\Delta \log \Delta)$. In a modification of Algorithm *ColorTrials* a node picks one color uniformly at random for each sequence $[2i\Delta_{N_+(v)}, 2(i+1)\Delta_{N_+(v)}]$ of $\Delta_{N_+(v)}$ colors for $0 \leq i < k_0 s$ and transmits these $k_0 s$ colors of size $\log \Delta$ resulting in a message of size $O(s \log \Delta)$. The probability that a node v gets a color out of a sequence of $2\Delta_{N_+(v)}$ colors is at least $1/2$ independently of the choices of its neighbors, since any node has at most $\Delta_{N_+(v)}$ uncolored neighbors, but chooses one color among $2\Delta_{N_+(v)}$ many. In case $|C(v)| \leq 2\Delta + 1$ with $\Delta \geq \log n$ a node executes the modified version (described above) of *ColorTrials* c times for some constant c. It picks $c/4$ executions out of the c executions, where it picks one color uniformly at random, i.e. it stays inactive for the other $3c/4$ executions. An execution is a success, if at most $\Delta/2$ neighbors participated with node v. Using Theorem 1 it can be shown that we have a success with probability $1 - 1/n^k$ for arbitrary k (and sufficiently large c). For a successful execution a node gets colored with probability at least $1/2$ since it picks one color out of $\Delta + 1$ and there are at most $\Delta/2$ colors picked by neighbors. Therefore, the required message size is only $O(s \log \Delta)$.

Theorem 7 only applies for nodes of degree $d(v) \in \Omega(\log n)$. The next theorem gives a bound on the reduction of an arbitrary degree $d(v)$ for more than one execution of *ColorTrials*. The proof uses similar ideas to the proof of Theorem 7.

Theorem 9. *After* $O(r)$ ColorTrials *all nodes* $v \in V$ *have* $d(v) \leq \Delta_0/2$*, where* Δ_0 *is the maximum size of any uncolored neighborhood before the first execution, with probability* $1 - 1/n^k$ *for arbitrary constant* k *and* $k_0 \geq 384(k+3)$*, if* $r \cdot \Delta_0 \geq \log n$*.*

Proof. Consider a node v for a graph G'_v. Let $\Delta_0/2 \leq d'_0(v) \leq \Delta_0$ be the initial degree of v before the first execution of *ColorTrials*. Due to Lemma 6 the probability $Pr(E'(v,c) = 1)$ to get a certain color c is at least $\tilde{d'}(v)/(24\Delta) \geq 1/48$ as long as $\tilde{d'}(v) > d'_0(v)/2$, i.e. $Pr(E'(v,c) = 1 | \sum_{i=0}^{c} E'(v,i)) \leq d'_0(v)/2)$. We have $|C_i(v)| > \Delta$ for all $i \in [0, k_0 r - 1]$ (otherwise node v might not be able to compute a coloring, e.g. if it is in a clique). Thus, we can use Theorem 4 with $t = k_0 r\Delta$ executions of *ColorTrials*, $p = 1/48$ and $y = d'_0(v)/2 \leq tp/2$ (for k_0 sufficiently large), yielding a bound of $e^{k_0/48 \cdot \tilde{d'_0}(v)/8} = e^{k_0/384 \cdot \tilde{d_0}'(v)} \geq e^{k_0/384 \cdot \log n} = 1 - 1/n^{k_0/384}$. Due to Theorem 5 this probability bounds also the probability of the event E_v for a node $v \in G$ that the number of uncolored neighbors is halved. Using

Theorem 2 the number of all uncolored neighbors is at most $\Delta_0/2$ for all nodes $v \in V$ with probability $1 - 1/n^{k_0/384-3}$. □

3.3.3 Analysis of Algorithm DeltaPlus1Coloring

First we consider Algorithm *RankingTrials* before discussing the run time and message size required to compute a $\Delta + 1$ coloring.

Theorem 10. *Within time* $O(\sqrt{\log n})$ *Algorithm* RankingTrials *computes a* $\Delta + 1$ *coloring with* $\Delta \leq \sqrt{\log n}$ *with probability* $1 - \frac{1}{n^c}$ *for some constant* c.

Proof. Consider a node $v \in V$ having less than $\sqrt{\log n}$ uncolored neighbors, i.e. $d(v) \leq \sqrt{\log n}$. Initially, it does not have any rank priority, i.e. $RP(v)[i] = none$ for all ranks i. For each rank i there are initially $2\sqrt{\log n} + 1$ available priorities. All rank priorities must be distinct, i.e. $RP(v)[i] \neq RP(v)[j]$ for $i \neq j$ and a node picks at most $\sqrt{\log n}$ rank priorities and at most one per rank i. Since $d(v) \leq \sqrt{\log n}$ at most $\sqrt{\log n}$ priorities get picked by neighbors per rank i. Thus we have for any rank i at least $\sqrt{\log n}+1$ unused priorities throughout the execution of the algorithm. Therefore, the probability that for some rank i within one message exchange a node v chooses a priority out of $\sqrt{\log n}+1$ such that no other uncolored neighbor of at most $\sqrt{\log n}$ many chooses the same random priority is at least $(1-1/(\sqrt{\log n}+1))^{d(v)} \geq 1/e$ independent of the choices of the rank priorities for ranks $j < i$ of all adjacent nodes. Once a node got a priority for some rank i, it does not apply to get another priority for the same rank. Still, at any time during the algorithm a node v has at least $\sqrt{\log n}+1$ different ranks to try to get a rank priority, since the total number of ranks is $2\sqrt{\log n}+1$ and whenever v tries to get a rank priority it has acquired at most $\sqrt{\log n}$ yet. Thus when performing one message exchange a node performs $\sqrt{\log n}+1$ trials. After $O(\sqrt{\log n})$ exchanges we have performed $O(\log n)$ trials. Each has a success probability of at least $1/e$ independent of prior trials. Thus the probability for a single node v to get $\sqrt{\log n}$ rank priorities is $1 - 2^{-O(\log n)}$ using Theorem 1. Using Theorem 2 all nodes obtained $\sqrt{\log n}$ rank priorities with probability $1 - 2^{-O(\log n)}$.

All priorities are distinct and are intended for distinct ranks. When using the rank priorities to get a rank then for each chosen rank priority $RP(v)[i] \neq none$ either a node v gets the rank i or a neighbor has already got it, since a node gets only one rank and a node v has picked $\sqrt{\log n}+1$ rank priorities but has at most $\sqrt{\log n}$ uncolored neighbors, node v gets a rank distinct from its neighbors. Afterwards, when nodes attempt to join according to their ranks, every node get a color. □

Theorem 11. *Within time* $O(\log \Delta + \sqrt{\log n})$ *a* $\Delta + 1$ *coloring is computed with probability* $1 - 1/n^k$ *for arbitrary constant* k.

Proof. Due to Theorem 9 after O(1) executions of Algorithm *ColorTrials* any node v halfs its uncolored neighbors as long as $\Delta \geq \log n$. Thus after $O(\log \Delta)$ executions we have $\Delta \leq \log n$. Assume $\sqrt{\log n} \leq d(v) < \log n$. Assume a node v with $d(v) \geq \Delta_0/2$ performed a sequence of $k_0 \log n/d(v)$ executions of *ColorTrials* for constant k_0, where Δ_0 equals the maximum degree Δ before the first execution of the sequence. Using Theorem 9 with $r = k_0 \log n/\Delta$ the degree $d(v)$ for each node $v \in V$ must be less than $\Delta/2$ with probability at least $1 - 1/n^{k_1+1}$ for an arbitrary constant k_1 (and sufficiently large constant k_0).

The total time until $\tilde{d}(v) < \sqrt{\log n}$ is bounded by $\sum_{i=\log\log n/2}^{\log\log n} O(1) \cdot \log n \cdot (1/2)^i = \sum_{i=0}^{\log\log n/2} O(\sqrt{\log n}) \cdot (1/2)^i = O(\sqrt{\log n})$. The chance that all $O(\log \Delta + \log\log n)$ divisions succeed for all nodes is $(1 - 1/n^{k_1+1})^{O(\log\Delta+\log\log n)} = 1 - 1/n^{k_1}$. Using Theorem 10 the remaining nodes can be colored in time $O(\sqrt{\log n})$. □

Messages of size $O(\sqrt{\log n} \cdot \log\log n)$ are sufficient during Algorithm *RankingTrials*, since $O(\sqrt{\log n})$ numbers have to be transmitted using one message. Each number is in $[0, 2\sqrt{\log n}]$ and can be encoded with $O(\log\log n)$ bits.

For Algorithm *DeltaPlus1coloring* messages of size $O(\log \Delta + \sqrt{\log n} \cdot \log\log n)$ are sufficient. As long as a node has more than $\sqrt{\log n}$ uncolored neighbors, i.e. $\tilde{d}(v) > \sqrt{\log n}$ it suffices to choose randomly one color of $O(\log \Delta)$ bits for each round and transmit it as shown in the remark after Theorem 8.

3.3.4 Analysis of Algorithms ConstDeltaColoring And ConstTimeColoring

After establishing a bound on the running time we investigate the amount of transmitted information.

Lemma 12. *Using at least* $(1 + 1/2^c)\Delta$ *colors, within time* $O(c)$ *the number of uncolored neighbors is at most* $\max(\Delta/(^c2), \log n)$ *for every node with probability* $1 - 1/n^k$ *for arbitrary constant* k.

Proof. Let Δ_0 be the maximum degree Δ before the first execution. Due to Theorem 9 after O(1) executions of Algorithm *ColorTrials* any node v halfs its uncolored neighbors as long as $\Delta \geq \log n$. After $O(c)$ rounds the fraction of uncolored neighbors is less than $1/2^{k_1 c}$ (for an arbitrary constant k_1), i.e. the maximum number of uncolored neighbors is $\Delta \leq \Delta_0/2^{k_1 c}$. Since we use $(1 + 1/2^c)\Delta_0$ colors and the neighbors $N(v)$ of a node v use up at most Δ_0 colors, at least $1/2^c\Delta_0$ colors are available for a node v in every execution of *ColorTrials*.

Using Theorem 8 with $s = 1/2^c\Delta_0/(\Delta_0/2^{k_1 c}) = 2^{(k_1-1)c}$ after one execution of *ColorTrials* the maximum number of uncolored neighbors for node v becomes $\Delta/2^{2^{(k_1-1)c}}$. In an analogous derivation for an initial maximum degree of $\Delta/x \leq \Delta/2^{2^{(k_1-1)c}}$, we can reduce the maximum degree to $\Delta/2^{x/k_0}$ with one message exchange with probability at least $1-1/n^{k+1}$ for any constant k and any sufficiently large constant $k_0 \geq 384(k+2)$ due to Theorem 8. Thus after two message exchanges the maximum number of uncolored neighbors is reduced from Δ to $\Delta/2^{2^{(x/k_0)}/k_0} < \Delta/2^x$ for $x \geq (k_0)^2$. Thus after $O(c)$ rounds all nodes have less than $\Delta/\binom{c}{2} \leq \log n$ uncolored neighbors with probability $(1-1/n^{k+1})^{O(c)} > 1-1/n^k$. □

Theorem 13. *If the number of uncolored neighbors is at most* $\log n$ *for all nodes, i.e.* $\Delta \leq \log n$*, then using* $O(\log^{1+1/c} n)$ *colors within time* $O(c)$ *every node gets colored with probability* $1-1/n^k$ *for arbitrary constant* k*.*

Proof. Consider a node $v \in G'_v$ with $\Delta^{1-1/c} < \tilde{d}'(v) \leq \Delta$. Due to Lemma 6 the probability $Pr(E'(v,c)=1)$ to get a certain color c is at least $Pr(E'(v,c)=1) \geq \Delta^{1-1/c}/(24\Delta) = 1/(24\Delta^{1/c}) \geq 1/(24\log^{1/c} n)$ as long as $\tilde{d}'(v) \geq \Delta^{1-1/c}$, i.e. $Pr(E'(v,c)=1 | \sum_{i=0}^{c} E'(v,c) \leq d'_0(v) - \Delta^{1-1/c})$. Thus we can use Theorem 4 with $t = k_0 \log^{1+1/c} n$, $p = 1/(24\log^{1/c} n)$ and $y = d'(v) - \Delta^{1-1/c} \leq tp/2$ (for k_0 sufficiently large), yielding a bound of $e^{k_0/48 \cdot \log n/8} = e^{k_0/384 \cdot \log n} = 1-1/n^{k_0/384}$. After one round the number of remaining uncolored nodes $d'(v) - y = d'(v) - (d'(v) - \Delta^{1-1/c}) = \Delta^{1-1/c}$. Using Theorem 2 the probability that event E_v occurs for all nodes is at least $1-1/n^{k_0/384-4}$. After c iterations of the loop every node is colored with probability $(1-1/n^{k_0/384-4})^c \geq (1-1/n^{k_0/384-5})$. □

Lemma 14. *Using* $(1+1/2^c)\Delta + O(\log^{1+1/c} n)$ *colors within time* $O(c + \log^* n)$ *every node gets colored with probability* $1-1/n^k$ *for arbitrary constant* k*.*

Proof. Since $\Delta \leq n$ and $^{\log^* n}2 = n$ using Theorem 12 yields that after $O(\log^* n)$ time the maximum number of uncolored neighbors Δ is at most $\log n$ with probability $1-1/n^k$. Using Theorem 13 yields that all nodes are colored with probability $1-1/n^k$. The probability that both eventuate is $(1-1/n^k)^2 \geq 1-1/n^{k-1}$ for an arbitrary constant k. □

Corollary 15. *Using* $(1+1/2^{\log^* n})\Delta + O(\log^{1+1/\log^* n} n)$ *colors within time* $O(\log^* n)$ *every node gets colored with probability* $1-1/n^k$ *for arbitrary constant* k*.*

Theorem 16. *Within time* $O(c)$ *every node gets a color out of* $O(\Delta \log^{(c)} n + \log^{1+1/c} n)$ *colors with probability* $1-1/n^k$ *for arbitrary constant* k*.*

Proof. Using Theorem 9 with $s = k_0 \log^{(c)} n$ and constant k_0 the maximum number of uncolored neighbors Δ is reduced by a factor $2^{\log^{(c)} n} = \log^{(c-1)} n$ with probability $1 - 1/n^k$ for all nodes $v \in V$ after one execution of *ColorTrials*. Using Theorem 7 again with $s = k_0 \log^{(c)} n \log^{(c-1)} n$ the maximum number of still uncolored neighbors Δ is reduced by a factor $2^{\log^{(c)} n \log^{(c-1)} n} > \log^{(c-2)} n$. Thus after c rounds the reduction is by a factor $\log n$ and in the round $c+1$ it is by a factor up to n, such that Δ is at most $\log n$ (Theorem 9 holds only for $\Delta \geq \log n$). Using Theorem 13 all nodes with $\Delta \leq \log n$ are colored in time $O(c)$ with probability $1 - 1/n^k$ yielding an overall probability of $(1 - 1/n^k)^2 \geq 1 - 1/n^{k-1}$ for arbitrary constant k. □

Let us discuss the amount of transmitted information for algorithm *ConstDeltaColoring*. A node transmits a number for every unused color in every round to its neighbors. However, as shown in the remark after Theorem 8 it is sufficient to choose one color for every sequence of $2\Delta_{N_+(v)}$ colors. Thus, choosing $\mathrm{O}(\log n)$ colors each with $\mathrm{O}(\log \Delta)$ bits yields a probability of $1-1/n^k$ for an arbitrary constant k, yielding messages of size $\mathrm{O}(\log \Delta \log n)$ during the execution of *ColorTrials*. Still, Theorem 8 is only valid for $\tilde{d}(v) \geq \log n$. But once $\tilde{d}(v) < \log n$ the size of a color is only $\Omega(\log \log n)$ bits (e.g. we might add $\mathrm{O}(\log^{1+1/c} n)$ to all assigned colors and, thereby, we can assign colors $\{0, 1, ..., \Omega(\log^{1+1/c} n)\}$ to the remaining nodes). Thus the maximum message size in one round is $O(\log n(\log \Delta + \log \log n \log^{1/c} n) = O(\log \Delta \log n)$. The reason for Algorithm *ConstTimeColoring* is the same except that the number of colors is $|C(v)| \in O(\Delta \log^{(c)} n + \log^{1+1/c} n)$ and thus a color requires $O(\log \Delta + \log^{(c+1)} n)$ bits instead of $O(\log \Delta)$.

Through another modification the message sizes can be reduced further. We let nodes get a color in two steps. First, a node picks an interval of colors. Second, it attempts to obtain an actual color from the chosen interval. More precisely, we assume that initially each node v has $|C(v)| = (1 + 1/2^{\log^* n - 2})(\Delta_{N_+(v)} + \log^{1+1/\log^* n} n)$ colors available. Each node v considers disjoint intervals $([0, l-1], [l, 2l-1], ...)$ of colors, where each interval contains $l := (1 + 1/2^{\log^* n - 1}) \log^{1+1/\log^* n} n$ colors and the total number of intervals is given by $|C(v)|/l$. A node v first picks one of these intervals $I(v) \in \{0, 1, ..., |C(v)|/l\}$ of colors uniformly at random. From then on, it only considers a subgraph $G_{I(v)}$ of G, i.e. only neighbors $u \in N(v)$ that have picked the same interval $I(u) = I(v)$. All other neighbors operate on different intervals and have no influence on node v. Then, a coloring is computed in parallel for all subgraphs. That is to say, node v executes Algorithm *ConstDeltaColoring* on $G_{I(v)}$ and tries to get a color or better said an index $ind_{I(v)}$ from $\{0, 1, ..., l-1\}$ in the interval $I(v)$. Its final color is given by the $ind_{I(v)}$ plus the color offset $I(v) \cdot l$ of the chosen interval $I(v)$.

Lemma 17. *Each node v has at most $\log^{1+1/\log^* n} n$ neighbors $u \in N(v)$ with $I(u) = I(v)$ w.h.p.*

Proof. Initially, each node picks independently uniformly at random one interval out of $(1+1/2^{\log^* n-2})\Delta_{N_+(v)}/((1+1/2^{\log^* n-1})\log^{1+1/\log^* n} n) = c_1 \cdot \Delta_{N_+(v)}/\log^{1+1/\log^* n} n$ many with $c_1 = (2^{\log^* n-1}+2)/(2^{\log^* n-1}+1)$. Thus, a node v expects $E \leq \frac{\Delta_{N_+(v)}}{c_1 \cdot \Delta_{N_+(v)}/\log^{1+1/\log^* n} n} = \log^{1+1/\log^* n} n/c_1$ neighbors to have chosen the same interval. Using a Chernoff bound the probability that there are more than a factor $1+c_1/2$ nodes beyond the expectation for a single interval is bounded by $1-2^{-c_1^2/8 \cdot E} = 1-2^{-c_1/8 \log^{1+1/\log^* n} n/c_1} \geq 1-1/n^{c_0}$ for an arbitrary constant c_0. Thus, w.h.p. the number of nodes in an interval is at most $(1+c_1/2) \cdot \log^{1+1/\log^* n} n/c_1 \leq \log^{1+1/\log^* n} n$. The probability that this holds for all intervals can be bounded to be $1-1/n^{c_0-3}$ using Theorem 2. □

Theorem 18. *The algorithm computes an $O(\Delta + \log^{1+1/\log^* n} n)$ coloring with bit complexity $O(\log n \log\log n)$ in time $O(\log^* n)$ w.h.p. (for sufficiently large n)*

Proof. The initial transmission of the interval requires at most $\log n$ bits, i.e. $\log \Delta - \log\log n$. Afterwards, when all nodes are split into subgraphs, the same analysis applies as for the *ConstDeltaColoring* Algorithm with $\Delta \leq \log^{1+1/\log^* n} n$, since each node only competes with at most $\log^{1+1/\log^* n} n-1$ other nodes due to Lemma 17 and we have $(1+1/2^{\log^* n-1})\log^{1+1/\log^* n} n$ available colors. The colors are picked such that the chance of getting a chosen color is constant, i.e. a node u picks one color for every sequence of $2\Delta_{N_+(v)}$ available colors, where $\Delta_{N_+(v)}$ denotes the maximum size of an uncolored neighborhood of an uncolored node $v \in N(u)$ before the current communication round. Thus, each node v that picks a color has probability $1/2$ to actually get a color independent of the choices of its neighbors, since the number of chosen colors of all neighbors together is at most $\Delta_{N_+(v)}$, i.e. half the colors of all available colors $2\Delta_{N_+(v)}$ and node v makes its choice independent of the concurrent choices of its neighbors. Thus, after a node has picked and transmitted $O(\log n)$ colors with probability $1-1/2^{O(\log n)} = 1-1/n^c$ for an arbitrary constant c, a node has obtained a color. Since each color requires $\log\log n$ bits the total bit complexity is $\mathrm{O}(\log n \log\log n)$. We can apply Corollary 15 that gives a running time of $O(\log^* n)$ w.h.p. □

3.3.5 Analysis of Ruling Set Algorithms

Theorem 19. *Within time $2^c d^{1/c}$ Algorithm* CoordinateTrials(d, c) *computes a $(2, c)$-ruling set.*

Proof. If node v stops attempting to join the ruling set, it was forced to stop by some neighbor $u \in N(v)$ with $Rank(u)$ larger 0 that continues itself, since u forced all its neighbors $N(u)$ (except those with the same rank) to stop and therefore is not stopped itself by any neighbor. Thus, all nodes $U \subseteq V$ that continue and are reachable from u by nodes in U only, must have the same rank, i.e. all nodes $u \cup U$ must have some coordinate i that is equal for all of them. If this was not the case, then there must have been two nodes $u, w \in U$ with distinct ranks that both continued. However, due to the algorithm either u forced w to stop or the other way around. Since some coordinate i has the same value for all nodes in U, we know that bit i of $Rank(u)$ must remain 0 from now on for all nodes $u \cup U$, i.e. we say that coordinate i is *done* and has no influence on the further computation.

Any two neighbors u, v attempting to join the ruling set must have one coordinate that is distinct for both of them throughout the algorithm, i.e. $\exists i \in [0, c-1]$ s.t. $coord(v)[i] \neq coord(u)[i]$. Initially, this holds since otherwise they have the same color. If two adjacent nodes u, v consider a coordinate i as done and still execute the algorithm then both must have the same value for coordinate i. Thus, at least one distinct coordinate that is not done remains. Thus, if a node v considers $c-1$ coordinates as done and is still executing the algorithm then for the last coordinate it is either forced to stop by a neighbor u that joins the ruling set or it joins the ruling set itself. This also implies that nodes joining the ruling sets are independent, since otherwise the last coordinate(s) that has not been considered as done would have to be equal.

Consider a node $v \in U \subseteq V$. When node v stops some neighbor w proceeds and and one coordinate for all continuing nodes $w \in W \subseteq U$ is done. Since there are only c coordinates within distance c of v a node joins the ruling set. One iteration out of $d^{1/c}$ many of the repeat loop (i.e. an increment of j) takes time 2^c. Thus the time complexity is $2^c \cdot d^{1/c}$. □

We describe three ways to improve the time complexity of Algorithm *CoordinateTrials* – two of them at the price of having larger distance to a node in the ruling set. First, it is not necessary to check a rank for all values in $[0, 2^c]$ if some coordinates, e.g. i, j, are *done* (see Proof of Theorem 19), i.e. the bits i, j of a rank are fixed to 0. If k coordinates are done, then a rank of a node v (and its neighbors) has k bits fixed to 0 and the rank must be one of 2^{c-k} distinct values. Second, if a node and all its neighbors have rank 0, there is no need to go through all possible 2^{c-k} values for a rank. If a node v only iterates through all possible rank values, if itself or a neighbor has rank larger 0 then at least one neighbor within distance 2 will consider a coordinate as done. Thus, within distance $2c$ a node joins the ruling set within time $d^{1/c} + \sum_{k=0..c} 2^{c-k} = d^{1/c} + 2^{c+1}$, i.e. we can compute a $(2, 2c)$

ruling set in time $d^{1/c}+2^{c+1}$. Third, a node can join the ruling set whenever its rank is strictly larger than that of all its neighbors and stop the algorithm whenever some neighbor has a rank larger than itself. In this case it does not have to iterate through the rank values at all. This results in a running time of $d^{1/c}$ to compute a $(2, 2^{c+1})$ ruling set.

Let $f_{trials}(d)$ be the time complexity depending on the number of colors of Algorithm *CoordinateTrials*. Let $c_{trials}(c)$ be the distance from any node to a node in the ruling set depending on the parameter c.

Theorem 20. *Within time* $O(c\log^{1/c} n + f_{trials}(\log^{1+1/\log^* n} n))$ *Algorithm* RandRulingSet(c) *computes a* $(2,(c+1)+c_{trials}(c+1))$*-ruling set for* $c>0$ *with probability* $1-1/n^3$.

Proof. We show that within time $O(c\log^{1/c} n)$ every node gets a colored node within distance $c+1$, such that all colored nodes form a proper coloring. Once this coloring is computed we call Algorithm *CoordinateTrials*$((\log^{1+1/\log^* n} n, c+1)$.

Initially, a node v starts with $p_v := 1/n = 2^{-\log n}$. After the first iteration of the for loop for node v or one of its neighbors holds that the sum of all values p_u of nodes $u \in N_+(v)$ is at least $2^{(-\log n^{1-1/c})}$ and at most 1. Since the maximum degree of a node is $n-1$ and a node begins with joining probability $1/n$, the sum of the values p_u of node $u \in N_+(v)$ is at most $1/n \cdot n = 1$. Due to Subroutine *IncProb(t)* a value p is not multiplied by t if the sum is larger than $1/t$. Thus the sum never exceeds 1. Consider a node v with a sum of at least $2^{(-\log n^{1-1/c})}$. After the second iteration node v itself or a neighbor has sum at least $2^{(-\log n^{1-2/c})}$. Thus after c iterations a node v must have a node $u \in N^c(v)$, i.e. within distance c, having the sum p_w with $w \in N_+(u)$ being at most 1 and at least 1/2. The duration of the first iteration is given by $\log n/(\log n^{1-1/c}) = \log n^{1/c}$ and in general, the duration of the ith iteration is $\log n^{1-(i-1)/c}/(\log n^{1-i/c}) = \log n^{1/c}$. Since one iteration takes $\log^{1/c} n$ time and we need c iterations, the time complexity is $c\log^{1/c} n$. When node v participates in computing a coloring with probability $\min(1, 64p_v \log n)$ (independently of all other nodes), the probability that more than $1024\log n$ neighbors or none participate is $1/n^8$ using a Chernoff bound (Theorem 1). Using Theorem 2 with probability $1-1/n^4$ all nodes $u \in V$ have at least 1 and at most $1024\log n$ neighbors that participate in computing the coloring.

An $O(\log^{1+1/\log^* n})$ coloring can be computed with probability at least $1-1/n^4$ as shown in Corollary 15. The overall success probability is $(1-1/n^4)\cdot(1-1/n^4) > 1-1/n^3$.

□

Theorem 21. *Within time* $O(2^c \log^{1/c} n)$ *Algorithm* RandRulingSet(c) *computes a* $(2, 2(c+1))$*-ruling set for* $c>0$ *with probability* $1-1/n^3$.

Proof. Due to Theorem 19, Algorithm *CoordinateTrials*$(\log^{1+1/\log^* n} n, c+1)$ computes a $(2, c+1)$-ruling set in time $O(2^{c+1}(\log^{1+1/\log^* n} n)^{1/c+1}) < O(2^c \log^{1/c} n)$. Using Theorem 20 completes the proof. □

In Subroutine *IncProb(t)* a node v transmits its value p_v to all two hop neighbors, resulting in messages of size $O(\Delta \log n)$. Alternatively, a node might transmit p_v to its neighbors and any node $u \in V$ having $\sum_{w \in N_+(u)} p_w \geq 1/t$ informs its neighbors. Thus messages of size $O(\log n)$ are sufficient.

3.4 Experimental Results

We implemented the *Multi-Trials* technique for coloring, i.e. Algorithm Colortrials, where each node picks a random number for each available color and keeps the smallest color for which no neighbor chose a larger or equal number. We compared this algorithm to a few other randomized and deterministic algorithms. The most relevant benchmark being the *Single Trial* Algorithm[63], where a node just picks a single color in each round uniformly at random. Furthermore, we implemented an algorithm *Single Max* interleaving the single trial algorithm [90, 63] and an algorithm that lets a node of maximum ID among its neighbors pick a color. More precisely, the algorithm repeats two steps: In the first step a node chooses a color uniformly at random and keeps it if no neighbor chose the same color[63]. In the second step it picks a color if its ID is maximum among all uncolored neighbors. Additionally, we implemented the deterministic algorithm of [110], where a node iteratively computes ruling sets and nodes in the ruling set can color their neighbors within a certain radius in a greedy manner. For our first algorithm a node computes a ruling set using Algorithm *CoordinateTrials* with radius 0, i.e. only the nodes in the ruling set choose a color. We used a 32 bit ID and split it into 4 bit packages. For our second variant *MaxID* a node joins the ruling set if it is ID is maximum within some radius. The experiments were carried out using Erdős Rényi graphs with varying edge probability p, i.e., each possible edge is added with probability p independently of all others edges. Erdős Rényi graphs allow to model very sparse as well as very dense graphs. Thus, we expect that our benchmark subsumes other graph classes like planar graphs or unit disk graphs. Also, we expect that due to the usage of randomness the chances that subgraphs of the graph are close to the worst (or at least bad) case for the algorithm is higher than when looking at graphs not employing randomness. The number of nodes was 1000. The algorithms were executed with different color palette sizes, i.e. the available colors equal the sequence of integers $[0, 2^i - 1]$ for an integer $i > 0$.

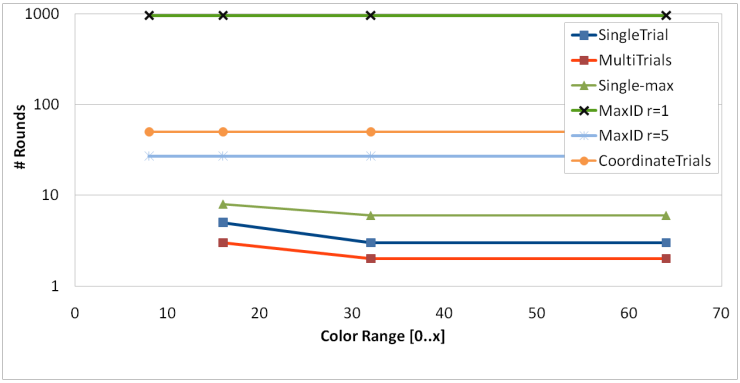


Figure 3.1: Random Graph with edge probability p=0.01.

Figures 3.1, 3.2 and 3.3 show that all greedy based algorithms generally require fewer colors, but are also significantly slower.(Note, the logarithmic scale on the y-axis.) The (implemented) randomized algorithms fail to solve the coloring problem if the number of available colors is somewhat below $\Delta + 1$ where the other algorithms still compute a solution. Nevertheless, the *Multi-Trials* coloring algorithm is fastest among the implemented algorithms.

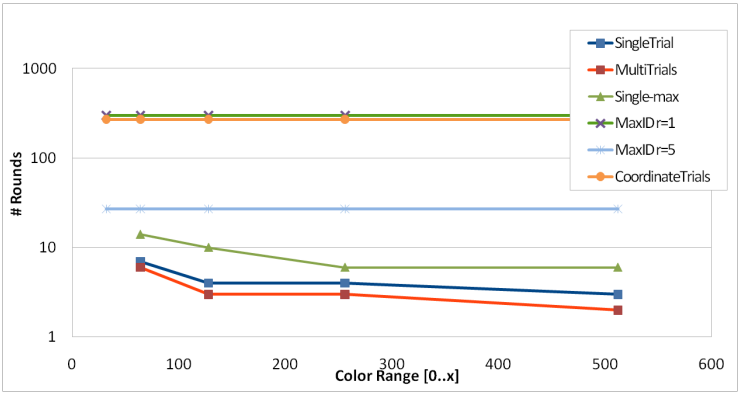


Figure 3.2: Random Graph 1000 with edge probability p=0.1.

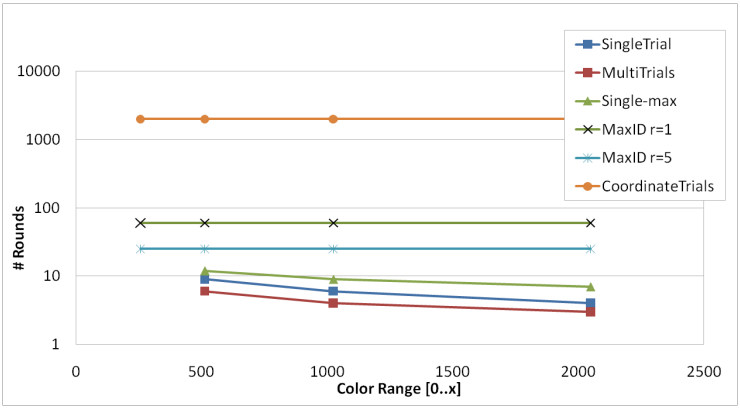


Figure 3.3: Random Graph with edge probability p=0.7.

Algorithm RankingTrials

1: $Rank(v) := none$, $RP(v)[i] := none, 0 \leq i \leq 2\sqrt{\log n}$ {$RP[i]$ equals priority for Rank i}
2: $RP_{free}(v)[i] := \{0, 1, ..., 2\sqrt{\log n}\}, 0 \leq i \leq 2\sqrt{\log n}$ {Available rank priorities for rank i}
3: **repeat**
4: $S(v) := \{(i, r) | RP(v)[i] = none$ and r is chosen randomly s.t.: {Pick a random priority for each available rank}
5: a) $r \in RP_{free}(v)[i]$ {Choose rank priority for rank i not taken by a neighbor}
6: b) $r \notin \{RP(v)[j] |$ for $i \neq j\}$ {Do not take a rank priority already taken by oneself}
7: c) $(i, r) \neq (j, r) \in S(v)$ for $i \neq j$ {Do not try to get the same rank priorities for different ranks}
8: Transmit $S(v)$ to all uncolored nodes $N(v)$
9: **for each** $(c_v, r_v) \in S(v)$ **do**
10: **if** $r_v \notin \{r | (c_v, r) \in S(u), u \in N(v)\}$ **then**
11: $RP(v)[c_v] := r_v$ {Keep priority if noone else wants it}
12: **end if**
13: **endfor**
14: Transmit $RP(v)$ to all nodes $N(v)$
15: $RP_{free}(v)[i] := RP_{free}(v)[i] \setminus \{RP(u)[i] | u \in N(v)\}$, for $0 \leq i \leq 2\sqrt{\log n}$
16: **until** $|\{i | RP(v)[i] \neq none\}| > \sqrt{\log n}$
17: Transmit *Ready* {Inform neighbors and wait for them}
18: **Wait until** recv *Ready* from all uncolored $u \in N(v)$
19: **for** $i = 0...2\sqrt{\log n}$ **do**
20: **if** $Rank(v) = none \wedge \exists l, RP(v)[l] = i$ **then**
21: $Rank(v) := l$; Transmit $Rank(v)$
22: **else**
23: $\forall$received $Rank(u)$ do $RP(v)[Rank(u)] := none$
24: **end if**
25: **end for**
26: **for** $i = 0..2\sqrt{\log n}$ **do**
27: **if** $Rank(v) = i$ **then**
28: $color(v) :=$ arbitrary $c \in C(v)$
29: Transmit $color(v)$ to all nodes $u \in N(v)$
30: **end if**
31: $C(v) := C(v) \setminus \{color(u) |$ received $color(u)\}$
32: **end for**

Chapter 4

Coloring Depending on the Chromatic Number

4.1 Introduction

Coloring is a fundamental problem with many applications. Unfortunately, even in a centralized setting, where the whole graph is known, approximating the chromatic number (the minimal number of needed colors), is currently computationally infeasible for general graphs and believed to take exponential running time. Thus, basically any reduction of the used colors below $\Delta + 1$ – even just to Δ – is non-trivial in general. Looking at the problem in a distributed setting, i.e., without global knowledge of the graph, makes the problem harder, since coloring is not a purely "local" problem, i.e., nodes that are far from each other have an impact on each other (and the chromatic number). Therefore, it is not surprising that all previous work has targeted to compute a $\Delta + 1$ coloring in general graphs as fast as possible (or resorted to very restricted graph classes). However, this somehow overlooks the original goal of the coloring problem, i.e., use as little colors as possible. Though in distributed computing the focus is often on communication, in many cases keeping the number of colors low outweighs the importance of minimizing communication. For example, a TDMA schedule can be derived from a (2-hop) coloring. The length of the schedule (and thus the throughput of the network) is determined by the number of employed colors. Despite the hardness of the coloring problem, intuitively, it should be possible to color a graph with small chromatic number with fewer colors and also a lot faster than a graph with large chromatic number. Our (randomized) algorithm shows that this in indeed the case. The algorithm works without knowledge of the chromatic number χ.

In this chapter we first state the algorithm followed by its analysis.

4.2 Algorithm

The algorithm (but not the analysis) itself is straightforward without many novel ideas. In the first two rounds a node attempts to get a color from a set with less than Δ colors. Then, (essentially) the *Multi-Trials* coloring algorithms (see Chapter 3) are used to color the remaining nodes.

Let $\Delta_0 := \Delta$ be the maximal size of a neighborhood in the graph, where all nodes are uncolored, and let $N_0(v)$ be the neighbors of node v upon the start of the algorithm. Let $N(v)$ be all uncolored neighbors of node v in the current iteration. The algorithm lets an uncolored node v be active twice with a fixed constant probability $1/c_1$. An active node chooses a random color from all available colors in the interval $[0, \Delta_0/2 - 1]$. Node v obtains its chosen color and exits the algorithm, if none of its neighbors $N(v)$ has chosen the same color. After the initial two attempts to get colored each node v computes the set of all colors $C^1_{N(v)}$ that have been colored by neighbors $N(v)$ in iterations 0 and 1 and how many neighbors $d(v)$ are left to color. The number of "conserved" (or saved) colors $s(v)$ (compared to a $\Delta + 1$ coloring) is given by the difference $s(v) := \Delta - d(v) + |C^1_{N(v)}|$. The algorithm can use the conserved colors to either speed up the running time, since more available colors render the problem simpler, e.g., allow for easier symmetry breaking, or to reduce the number of used colors as much as possible. In Algorithm *FastRandColoring* we spend half of the conserved colors for fast execution and preserve the other half to compute a coloring using $\Delta_0 + 1 - s(v)/2$ colors. A node v repeatedly chooses uniformly at random an available color from $[0, \Delta_0 + 1 - s(v)/2]$ using Algorithm *DeltaPlus1Coloring* (Chapter 3) until the number of available colors is at least a factor two larger than the number of uncolored neighbors. Afterwards it executes Algorithm *ConstantDeltaColoring* (Chapter 3) using 2Δ colors. Our algorithm is non-uniform, i.e., every node knows an upper bound on the total number of nodes n and the maximal degree Δ.

4.3 Analysis

We use the following Chernoff bound(s):

Theorem 22. *The probability that the number X of occurred independent events $X_i \in \{0, 1\}$, i.e., $X := \sum X_i$, is less than $(1-\delta)$ times a with $a \leq \mathbb{E}[X]$ can be bounded by $Pr(X < (1-\delta)a) < e^{-a\delta^2/2}$. The probability that the sum is more than $(1+\delta)b$ with $b \geq \mathbb{E}[X]$ with $\delta \in [0, 1]$ can be bounded by $Pr(X > (1+\delta)b) < e^{-b\delta^2/3}$.*

Algorithm FastRandColoring

1: $col(v) := none$
2: $J(v) := [0, \Delta_0/2 - 1]$
3: **for** $i = 0..1$ **do**
4: $choice(v) :=$ With probability $1/c_1$ random color from $J(v)$ else $none$
5: **if** $choice(v) \neq none \wedge \nexists u \in N(v), s.t. choice(u) = choice(v) \vee col(u) = choice(v)$ **then** $col(v) := choice(v)$ and exit **end if**
6: **end for**
7: $C^1_{N(v)} := \{col(u) | u \in N_0(v)\} \setminus none$
8: $s(v) := \Delta_0 - d(v) - |C^1_{N(v)}|$
9: $H(v) := [0, \Delta_0 + 1 - s(v)/2] \setminus C^1_{N(v)}$ {available colors}
10: Execute Algorithm DeltaPlus1Coloring (Chapter 3) using colors $H(v)$ until $|H(v)| \geq 2d(v)$
11: Execute Algorithm ConstDeltaColoring (Chapter 3) using colors $H(v)$

Corollary 23. *The probability that the number X of occurred independent events $X_i \in \{0, 1\}$, i.e., $X := \sum X_i$, is less than $\mathbb{E}[X]/2$ is at most $e^{-\mathbb{E}[X]/8}$ and the probability that is more than $3\mathbb{E}[X]/2$ is bounded by $e^{-\mathbb{E}[X]/12}$.*

Consider any coloring of the graph G using the minimal number of colors χ. Let S_c be a set of nodes having color $c \in [0, \chi - 1]$ for this coloring. For a node v with color c for this optimal coloring, we have $v \in S_c$. Let choice $i \geq 0$ (of colors) be the $(i + 1)$-st possibility where a node could have chosen a color, i.e., iteration i of the for-loop of Algorithm *FastRandColoring*. Let the set C^i_S be all distinct colors that have been obtained by a set of nodes S for any choice $j \leq i$, i.e., $C^i_S := \{c | \exists u \in S,$ s.t. $col(u) = c$ after iteration $i\}$. We do not use multisets here, i.e., a color c can only occur once in C^i_S. Let P^i_S be all nodes in S that make a choice in iteration i, i.e., $P^i_S := \{c | \exists u \in S,$ s.t. $choice(u) = c$ in iteration $i\}$. Let CP^i_S be all colors that have been chosen (but not yet obtained) by a set of nodes S in iteration i, i.e., $CP^i_S := \{c | \exists u \in P^i_S,$ s.t. $choice(u) = c$ in iteration $i\}$. By definition, $|CP^i_S| \leq |P^i_S|$.

To deal with the interdependence of nodes we follow the idea of *stochastic domination*. If X is a sum of random binary variables $X_i \in \{0, 1\}$, i.e., $X := \sum_i X_i$, with probability distributions A, B and $Pr_A(X_i = 1 | X_0 = x_0, X_1 = x_1, ..., X_{i-1} = x_{i-1}) \geq Pr_B(X_i = 1 | X_0 = x_0, X_1 = x_1, ..., X_{i-1} = x_{i-1}) = p$ for any values of $x_0, x_1, ..., x_{i-1}$, we can apply a Chernoff bound to lower bound $Pr_A(X \geq x)$ by a sum of independent random variables X_i, where $X_i = 1$ with probability p.

Theorem 24. *After choice $i \in [0, 1]$ for every node v holds w.h.p.: The colored nodes C^1_S of any set $S \in \{S_c \cap N_0(v) | c \in [0, \chi - 1]\}$ or $S \in \{N(v), N_0(v)\}$*

with $|S| \geq c_2 \log n$ *fulfill* $|C_S^1| \in [|S|/(16c_1), 3|S|/c_1]$ *with* $c_1 > 32$. *The number of nodes* $|P_S^1|$ *making a choice is at least* $|S|/(4c_1)$ *and at most* $3|S|/(2c_1)$.

Proof. Consider such a set S of nodes for some node v. For i possibilities to make a choice we expect (up to) $i|S|/c_1$ nodes to actually make a choice. Using the Chernoff bound from Corollary 23 the number of nodes that choose a color deviates by no more than one half of the expectation with probability $1 - 2^{i/8|S|/c_1} \geq 1 - 2^{i/8c_2 \log n/c_1} = 1 - 1/n^{c_3}$ for a constant $c_3 := ic_2/(8c_1)$. Thus, at most $3i|S|/(2c_1)$ neighbors of v make a choice and potentially get colored with probability $1 - 1/n^{c_3}$. Using Theorem 2 (Chapter 3) this holds for all nodes with probability $1 - 1/n^{c_3-3}$, which yields the bounds $|P_S^1| \leq 3|S|/(2c_1)$ and $|C_S^1| \leq 3|S|/c_1$.

For choice i w.h.p. the number of nodes that make a choice is therefore in $[a, b] := [1/2 \cdot (1 - 3i/(2c_1)) \cdot |S|/c_1, 3|S|/(2c_1)]$. The lower bound, i.e., $a \leq |P_S^i|$, follows if we assume that for each choice $j < i$ at most $3|S|/(2c_1)$ nodes get colored, which happens w.h.p.. Thus, after $i - 1$ choices at least $(1 - 3i/(2c_1)) \cdot |S|$ nodes can make a choice, i.e., are uncolored. We expect a fraction of $1/c_1$ to choose a color. Using Corollary 23 the nodes that make a choice is at least half the expected number w.h.p.. Thus, for choice 1 we have for $c_1 > 32$ and $a := (1 - 3/(2c_1))/(2c_1) \cdot |S|$ the following: $|S|/(4c_1) \leq a \leq |P_S^1|$.

Consider an arbitrary order $w_0, w_1, ..., w_{|S|-1}$ of nodes S. We compute the probability that node $w_k \in S$ obtains a distinct color for choice i from all previous nodes $w_0, w_1, ..., w_{k-1} \in S$. The probability is minimized, if all $k - 1$ nodes have distinct colors and k is large. Since $k \leq b = 3|S|/(2c_1)$ we have $p(col(w_k) \in [0, \Delta_0/2] \setminus S_{w_0,w_1,...,w_{k-1}}) \geq p(col(w_k) \in [0, \Delta_0/2] \setminus S_{w_0,w_1,...,w_{b-1}}) \geq 1/c_1 \cdot (1 - b/(\Delta_0/2)) \geq (1 - 3/\Delta_0/(2c_1)/(\Delta_0/2))/c_1 = 1/c_4$ with constant $c_4 := 1/c_1 \cdot (1 - 3/c_1)$. The lower bound for the probability of $1/c_4$ holds for any $k \in [0, b - 1]$ and any outcome for nodes $S_{w_0,w_1,...,w_{k-1}}$. Thus, to lower bound the number of distinct colors $|C_S|$ that are obtained by nodes in S we assume that the number of nodes that make a choice is only a and that each node that makes a choice gets a color with probability $1/c_4$ (independent of the choices of all other nodes). Using the Chernoff bound from Corollary 23 gives the desired result for a set S. In total there are n nodes and we have to consider at most $1 + \chi \leq n + 2$ sets per node. Using Theorem 2(Chapter 3) for $n \cdot (n + 1)$ events each occurring w.h.p. completes the proof. □

Next we consider a node v and prove that for the second attempt of all uncolored nodes $u \in S_c \cap N(v)$ a constant fraction of colors taken by independent nodes $w \in S_c \cap N(v) \setminus \{u\}$ from u are not taken (or chosen) by its neighbors $y \in N(u)$.

Theorem 25. *For the second choice let $E(c)$ be the event that for a node v for each uncolored node $u \in N_0(v) \cap S_c$ holds $|(CP^1_{N_0(u)} \cup C^1_{N_0(u)}) \cap C^1_{N_0(v) \cap S_c}| \leq 3/4 |C_{N_0(v) \cap S_c}|$ for $|N(v) \cap S_c| \geq c_2 \log n$. Event $E(c)$ occurs w.h.p.*

Proof. Consider a colored node $w \in S_c \cap N_0(v)$ for some node v. We compute an upper bound on the probability that an uncolored node $y \in N(u)$ gets (or chooses) color $col(w)$, i.e., $p(\exists y \in N(u), col(y) = col(w) \vee choice(y) = col(w)) = p(\vee_{y \in N(u)} col(y) = col(w) \vee choice(y) = col(w)) \leq \sum_{y \in N(u)} p(col(y) = col(w) \vee choice(y) = col(w))$. The latter inequality follows from the inclusion-exclusion principle: For two events A, B we have $p(A \cup B) = p(A) + p(B) - p(A \cap B) \leq p(A) + p(B)$. We consider the worst case topology and worst case order in which nodes make their choices to maximize $\sum_{y \in N(u)} p(col(y) = col(w) \vee choice(y) = col(w))$. Due to Theorem 24 for every uncolored node $y \in N(u)$ at most $|P^0_{N_0(y)}| + |P^1_{N_0(y)}| \leq 3d(y)/c_1 \leq 3\Delta_0/c_1$ neighbors $z \in N_0(y)$ make a choice during the first two attempts $i \in [0, 1]$. To maximize the chance that some node y obtains (or chooses) color $col(w)$, we can minimize the number of available colors for y and the probability that some neighbor $z \in N_0(y)$ chooses color $col(w)$, since when making choice i we have $p(choice(y) = col(w)) \leq 1/(c_1 |J(y)|)$ because each available color in $J(y)$ is chosen with the same probability. To minimize $|J(y)|$ the number of colored nodes $z \in N_0(y)$ should be maximized and at the same time each node $z \in N_0(y)$ should have a neighbor itself with color $col(w)$. The latter holds, if $z \in N_0(y)$ is adjacent to node w. Thus, to upper bound $p(col(y) = col(w))$ we assume that node w and each node $y \in N(u)$ share the same neighborhood (except u), i.e., $N_0(y) \setminus \{u\} = N_0(w)$, and the maximal number of nodes in $N_0(y)$ given our initial assumption are colored or make a choice, i.e., $3d_0(y)/c_1 \leq 3\Delta_0/c_1$. This, yields $p(col(y) = col(w)) \leq 1/(c_1 |J(y)|) \leq 1/(c_1(\Delta_0/2 - 3\Delta_0/c_1)) \leq 8/(c_1 \Delta_0)$ (for $c_1 > 32$) and therefore $p(\exists y \in N(u), col(y) = col(w)) = p(\vee_{y \in N(u)} col(y) = col(w)) \leq \sum_{y \in N(u)} p(col(y) = col(w)) \leq 3\Delta_0/c_1 \cdot 8/(c_1 \Delta_0) \leq 1/c_1$ (for $c_1 > 32$). In other words, the probability that some uncolored node $y \in N(u)$ has obtained color $col(w)$ or chooses $col(w)$ is bounded by $1/c_1$.

Let us estimate the probability that some neighbor $y \in N_0(u)$ gets the same color as a node $w_1 \in N_0(v) \cap S_c$ given that some nodes $z \in N_0(u)$ have chosen or obtained $col(w_0)$ for some node $w_0 \in C_{N_0(v) \cap S_c} \setminus \{w_1\}$. To minimize $|J(y)|$ we assume that $|J(y)|$ is reduced by 1 for every colored node $w_0 \in C_{N_0(v) \cap S_c} \setminus \{w_1\}$. Since at most $3/2 d_0(y)/c_1 \leq 3/2 \Delta_0/c_1$ neighbors make a choice concurrently, the event reduces the size of $|J(y)|$ by at most $3/2 \Delta_0/c_1$. Using the same calculations as above with $|J(y)| \leq \Delta_0/2 - 9/2 \Delta_0/c_1$, the probability that some node $y \in N_0(u)$ has obtained color $col(w)$ or chooses $col(w)$ given the outcome for any set of colored nodes $W \subseteq N_0(v) \cap S_c$ is at most $1/2$. Thus, we expect at most $|C_{N_0(v) \cap S_c}|/2$ colors

from $C_{N_0(v)\cap S_c}$ to occur in node u's neighborhood. Using the Chernoff bound from Corollary 23, we get that the deviation is at most $1/2$ the expectation with probability $1-2^{-|C_{N_0(v)\cap S_c}|/8}$ for node u, i.e., the probability $p(E(u,c))$ of the event $E(u,c)$ that for a node $u \in N(v)$ at most $3|C_{N_0(v)\cap S_c}|/4$ colors from $N_0(v) \cap S_c$ are also taken or chosen by its neighbors $y \in N_0(u)$ is at least $1 - 2^{-|C_{N(v)\cap S_c}|/8}$. Using Theorem 24 for $S = N_0(v) \cap S_c$ we have $|C_{N_0(v)\cap S_c}| \geq |S|/(16c_1) = |N_0(v) \cap S_c|/(16c_1) \geq c_2 \log n/(16c_1)$. Therefore, $p(E(u,c)) \geq 1 - 1/n^{c_2/(16c_1)}$. Due to Theorem 2(Chapter 3) the event $E(c) := \bigwedge_{u\in N_0(v)} E(u,c)$ occurs w.h.p.. □

Using Theorem 25 together with Theorem 2(Chapter 3) yields the following corollary.

Corollary 26. *An event $E(c)$ (as defined in Theorem 25) occurs w.h.p. given $\bigwedge_{c_1\in X\subseteq[0,\chi],|N(v)\cap S_{c_1}|\geq c_2 \log n} E(c_1)$ for an arbitrary set $X \subseteq [0,\chi]$*

Theorem 27. *After the first two choices for a node v with initial degree $d_0(v) \geq \Delta_0/2$ there exists a subset $N_c \subseteq N_0(v)$ with $|N_c| \geq (\Delta+1)/(c_5\chi)$ that has been colored with $(\Delta+1)/(2c_5\chi)$ colors for a constant c_5 w.h.p. for $\Delta \in \Omega(\log^{1+1/\log^* n} n)$ and $\chi \in O(\Delta/\log n)$.*

Proof. By assumption $\chi \in O(\Delta/\log n)$, i.e., $\chi < 1/(4c_3)\Delta/\log n$. At least half of all neighbors $u \in N_0(v)$ with $u \in S_c \cap N_0(v)$ must be in sets $|S_c \cap N_0(v)| \geq c_3 \log n$. This follows, since the maximum number of nodes in sets $|S_c \cap N_0(v)| < c_3 \log n$ is bounded by $\chi \cdot c_3 \log n \leq \Delta_0/4$. Assume that all statements of Theorem 24 that happen w.h.p. have actually taken place. Consider a node v and a set $N_0(v) \cap S_c$ with $|S_c \cap N_0(v)| \geq c_3 \log n$ given there are at most $3/4d_0(v) \leq 3/4\Delta_0$ colored neighbors $u \in N_0(v)$. For a node $u \in N_0(v) \cap S_c$ the probability that it obtains the same color of another node $N_0(v) \setminus \{u\} \cap S_c$ is given by the probability that it chooses a color $col(w)$ taken by node $w \in N_0(v) \setminus \{u\} \cap S_c$ that is not chosen by any of u's neighbors $x \in N_0(u)$. Due to Corollary 26 $|C_{N_0(v)\cap S_c}|/4$ colors exist that are taken by some node $w \in N_0(v) \cap S_c$ but not taken (or chosen for the second choice) by a neighbor $x \in N_0(u)$. Due to Theorem 24 we have $|C_{N_0(v)\cap S_c}|/4 \geq |N_0(v) \cap S_c|/(64c_1)$. Additionally, the theorem yields $|P^1_{N_0(v)\cap S_c}| \geq |N_0(v) \cap S_c|/(4c_1)$.

The probability for a node $u \in P^1_{N_0(v)\cap S_c}$ to obtain (not only choose) a color in $C_{N_0(v)\cap S_c}$ becomes the number of "good" colors, i.e., $|N_0(v) \cap S_c|/(64c_1)$, divided by the total number of available colors, i.e., $1/(\Delta_0/2)$, yielding $|N_0(v) \cap S_c|/(32c_1 \cdot \Delta_0)$. This holds irrespectively of the behavior of other nodes $w \in P^1_{N_0(v)\cap S_c}$ and $w \in N_0(v) \cap S_d$ with $d \in [0,\chi-1] \setminus \{c\}$. The reason is that a node u makes its decision what color to choose independently of its neighbors $y \in N_0(u)$ and Theorem 24 and Corollary 26 already account

for the worst case behavior of neighbors $y \in N_0(u)$ to bound the probability that node u gets a chosen color.

Thus, for a set of $|P^1_{N_0(v)\cap S_c}| \geq |N_0(v) \cap S_c|/(4c_1)$ nodes we expect that for at least $|N_0(v) \cap S_c|^2/(128c_1^2 \cdot \Delta_0)$ nodes u there exists another node $w \in (N_0(v) \cap S_c) \setminus \{u\}$ with the same color. The expectation $|N_0(v) \cap S_c|^2/(128c_1^2 \cdot \Delta_0)$ is minimized if all sets $|N_0(v) \cap S_c| \geq c_3 \log n$ are of equal size and as small as possible, i.e., $\Delta_0/(4\chi)$ since at least $\Delta_0/4$ nodes are in sets $|N_0(v) \cap S_c| \geq c_3 \log n$ for some c. This gives $\sum_{c\in[0,\chi-1]} |N_0(v) \cap S_c|^2/(128c_1^2 \cdot \Delta_0) \geq \sum_{c\in[0,\chi-1]} (\Delta_0)^2/(2048c_1^2 \cdot \Delta_0 \cdot \chi^2) = \Delta_0/(c_5 \cdot \chi)$ for $c_5 := 2048c_1^2$. Since by assumption $\chi \in O(\Delta_0/\log n)$ using Corollary 23 the actual number deviates by at most $1/2$ of its expectation w.h.p.. Therefore, for at least $\Delta_0/(c_5 \cdot \chi)$ nodes $u \in N_0(v) \cap S_c$ there exists another node $w \in N_0(v)\setminus\{u\}\cap S_c$ with the same color. Thus, to color all of these $\Delta_0/(c_5 \cdot \chi)$ nodes only $\Delta_0/(2c_5 \cdot \chi)$ colors are used. □

Theorem 28. *If* $\Delta \in \Omega(\log^{1+1/\log^* n} n)$ *and* $\chi \in O(\Delta/\log^{1+1/\log^* n} n)$ *then Algorithm* FastRandColoring *computes a* $(1 - 1/O(\chi))\Delta$ *coloring in time* $O(\log \chi + \log^* n)$ *w.h.p..*

Proof. Extending Theorem 27 to all nodes using Theorem 2(Chapter 3) we have w.h.p. that each node v with $d_0(v) \geq \Delta_0/2$ has at most $(\Delta_0 + 1) \cdot (1 - 1/(c_5\chi))$ uncolored neighbors after the first two choices. However, node v is allowed to use $d(v) + 1$ colors and, additionally, half of the conserved colors, i.e., $s(v)/2 = \Delta_0/(8c_2\chi) \geq \log^{1+1/\log^* n} n/(4c_5)$ (see Theorem 27), to get a color itself. When executing Algorithm *DeltaPlus1Coloring* the maximum degree is reduced by a factor 2 in $O(1)$ rounds as long as it is larger than $\Omega(\log n)$ due to Theorem 9 (Chapter 3). The time until the maximum degree Δ is less than $s(v)/4$ is given by $O(\log \Delta_0 - \log s(v))= O(\log \Delta_0 - \log(\Delta_0/(32c_2\chi))) = O(\log \chi)$. Thus, we have at least 2Δ colors available, i.e., at least $\log^{1+1/\log^* n} n/(4c_5)$ additional colors, when calling Algorithm *ConstDeltaColoring*. Therefore, the remaining nodes are colored in time $O(\log^* n)$ using Corollary 15(Chapter 3). Nodes with initial degree $d_0(v) < \Delta_0/2$ can use $|H(v)| := \Delta_0+1-s(v)/2 \geq \Delta_0+1-d_0(v)/2 \geq \Delta_0/2 \in \Omega(\log^{1+1/\log^* n} n)$ (since one cannot save more colors $s(v)$ than there are neighbors $d_0(v)$, i.e $s(v) \leq d_0(v)$) colors to color $d(v)+1$ nodes. After the first two attempts to get a color we have $d(v) \leq d_0(v) - 2s(v) < \Delta_0/2 - 2s(v) \leq (\Delta_0 + 1 - s(v)/2)/2 = |H(v)|/2$. The first inequality follows since any saved color c implies that at least two neighbors $u, w \in N_0(v)$ are colored with the same color c. Thus, as for the case $d_0(v) \geq \Delta_0/2$ there are at least $2 \cdot d(v) \leq |H(v)|$ colors with $|H(v)| \in \Omega(\log^{1+1/\log^* n} n)$ available to color $d(v)$ nodes. □

Chapter 5

(Extended) Deterministic Coin Tossing

5.1 Introduction

Minimum dominating sets (MDS) and connected dominating sets (CDS) are well-studied theoretical problems in wireless multi-hop networks, such as ad hoc, mesh, or sensor networks. In hundreds of papers they have been identified as key to efficient routing, media access control, or coverage, to just name three popular examples of usage. Consequently, the networking community has suggested a great number of algorithms towards computing CDS et al.; almost all of these algorithms are distributed, as wireless networks tend to be unreliable and dynamic, and conventional global algorithms seem too slow to cope with this constant churn. However, most algorithms are also heuristic in nature, and have been shown to perform poorly in efficacy and/or efficiency, when analyzed rigorously. However, there are exceptions; we discuss them in detail in Section 2.2.

Given the huge impetus from the application side, recent research mostly concentrated on special graph classes that represent the geometric nature of wireless networks well. The classic theoretical model for wireless networks is the so-called unit disk graph (UDG) model, where the nodes are points in the plane, and two nodes are neighbors in the graph if and only if their Euclidean distance is at most 1. However, wireless radios will never transmit in perfect circles, and hence the UDG model has recently gotten a lot of stick. Instead the community started looking into generalized models, e.g. the quasi unit disk graph (QUDG) model, or the unit ball graph (UBG) model. In Section 2.1 we adopt the so-called bounded-independence graph (BIG) model (also called growth-bounded graph (GBG) model [74]); the BIG

model only restricts the number of independent nodes in each neighborhood and is therefore a generalization of the UDG, QUDG and UBG models.

In Section 5.2 we present a novel distributed algorithm for the maximal independent set (MIS) problem. On bounded-independence graphs our algorithm finishes in $O(\log^* n)$ time, n being the number of nodes. As we discuss in more detail in Section 2.2, our algorithm beats all existing algorithms for geometric models such as UDG or BIG by an exponential factor. Indeed, thanks to a lower bound argument by Linial [88], our algorithm is asymptotically optimal. In the BIG model a MIS is a constant approximation of a MDS, hence our algorithm gives the fastest constant MDS approximation. In Section 5.3 we also quickly mention how to compute a CDS, how to obtain a polynomial time approximation scheme (PTAS), and an asymptotically optimal algorithm for computing a Δ+1 coloring, as well as a distance two coloring. Finally, we analyze our algorithm from a theoretical and experimental perspective.

5.2 MIS Algorithm

Let us start by giving an informal description of our deterministic MIS algorithm. Each node performs a series of competitions against neighbors, such that more and more nodes drop out until only nodes joining the MIS remain. For all nodes not in the MIS or not adjacent to a node in the MIS, the process is repeated.

To get a deeper understanding of a competition we take a closer look at the very first competition. A node v competes against the neighbor u with minimum ID. If ID_u is larger than ID_v, i.e. node v has the smallest ID among all neighbors, the result is 0. If ID_v is not the smallest of all neighbors, the result of the competition is the maximum position for which v's ID has a bit equal to 1 and u's ID has a bit equal to 0. For ID_v being 11101 and ID_u being 10001, the two differing positions are 3 and 4 and thus the result of the competition for v is 4, i.e. $r_v = 4$.

The result r_v of the first competition forms the basis for the next competition, the result of that competition in turn is used for the following competition and so forth.

A node can be in one of five states, which it might alter after each competition (see Algorithm Update State). Initially each node is a competitor. If the result of the competition for node v is (strictly) smaller than that of all its competing neighbors, node v becomes a dominator and joins the MIS. All adjacent nodes of a dominator become dominated. Both dominators and dominated nodes are not involved in further competitions. In case the result of a node is as small as that of all its neighbors and at least one neighbor has the same result, the node becomes a ruler. A neighbor of a ruler gets ruled

(if not dominated). A ruler immediately ends the current phase of becoming a dominator (lines 4 to 14 in Algorithm MIS), becomes a competitor again and proceeds to the next phase. After a node has advanced to the c^{th} phase (where c is some number depending on the function $f()$ - see Definition 1) by changing back and forth between competitor and ruler only, i.e. without ever being ruled, it must be dominated or a dominator. A ruled node, ends the phase and stays quiet until all neighbors are ruled (or dominated). Then it starts the algorithm again as competitor, i.e. it is again in its first phase (of becoming a dominator). During a phase a node executes at most $\log^* n + 2$ recursive competitions (as be shown in Section 5.4.1) and every competing node must change its state. In the subsequent competition (the first one of a new phase) all competitors compete again by using IDs. For some more examples including updates of states consider Figure 5.1.

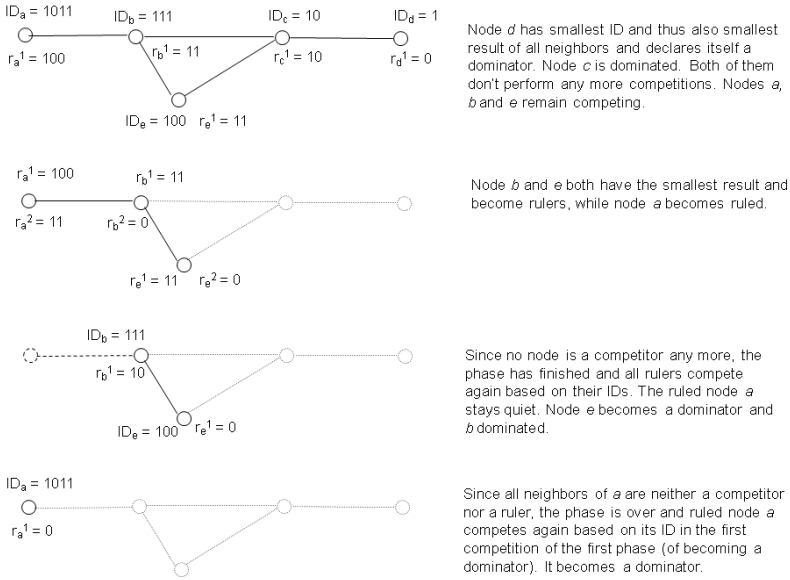


Figure 5.1: Graph showing a complete execution of Algorithm MIS. Dominators and dominated nodes are shown with a dotted line. Ruled nodes with a dashed line.

Next, we give a more formal definition of a competition to clarify our

notation. Let r_v^j denote the result of the j^{th} (recursive) competition for node v. The first competition is always based on the IDs. Thus we define $r_v^0 := ID_v$. Any number r_v^{j-1} consists of l bits ($l \leq \log^{(j)} n$ as shown in Lemma 31) and has the form $r_v^{j-1} = y_v^t, y_v^{t-1}, ..., y_v^1$. A competitor only competes against nodes that are also competitors, i.e. the results of ruled or dominated nodes are not considered. In order to perform the j^{th} (recursive) competition with $j \geq 1$ node v chooses a competitor $u \in N(v)$, s.t. $r_u^{j-1} = \min_{w \in N(v)} r_w^{j-1}$. In case the length of r_u^{j-1} and r_v^{j-1} differ, we make them equal by prepending zeros to the smaller number r^{j-1}. The result r_v^j for node v gives the maximum position, s.t. the $(r_v^j)^{th}$ bit of number r_v^{j-1} is 1 (i.e. $y_v^{r_v^j} = 1$) and the $(r_v^j)^{th}$ bit of r_u^{j-1} is 0 (i.e. $y_u^{r_v^j} = 0$). If r_v^{j-1} is a minimum of all r^{j-1} (i.e. $r_v^{j-1} \leq r_u^{j-1}$), then we set $r_v^j = 0$. Taking into account both cases yields: $r_v^j := \max(\{k | (y_v^k > y_u^k) \wedge (r_v^{j-1} > r_u^{j-1})\} \cup \{0\})$. Observe that by definition all bits higher than the $(r_v^j)^{th}$ bit are the same (i.e. $y_u^i = y_v^i$ for $r_v^j < i \leq \log^{(j)} n$) if $r_v^{j-1} \geq r_u^{j-1}$.

Fast termination of Algorithm MIS is shown in Section 5.4.1 for bounded-independence graphs. However, the algorithm is robust in the sense that it is correct for general graphs as well (see Section 5.4.2).

A node executes phases in a synchronous manner with its neighbors (see also Lemma 32). For a reader not familiar with distributed computing this might seem a too strong assumption. A simple way to solve the problem is that we let all nodes know an upper bound of n. With that all nodes can execute all steps of the algorithm in lock-step, even if some of the nodes are not participating in some of the steps (because they are not competing anymore, for instance). On the one hand this guarantees global synchronization, on the other hand our algorithm is not uniform anymore.

A better solution is to use a local synchronizer (i.e. synchronizer α). With that, all messages can be exchanged completely asynchronously; the only constraint is that nodes need to wait until their neighbors have signaled that they are okay with executing the next step of the algorithm. Using a synchronizer it may happen that some nodes already are two rounds ahead of others, however, locally all nodes are always within one step.

5.3 Applications of MIS

Our MIS algorithm serves as a key building block to tackle many other problems for bounded-independence graphs.

5.3.1 CDS and MDS

In order to obtain a CDS given a MIS S, each node $v \in S$ chooses a shortest path to every node $u \in (N^3(v) \cap S)$ with $ID_u < ID_v$ and adds all nodes from

Algorithm MIS

For each node $v \in V$

repeat
  state s_v := competitor
  repeat
    {Phase start} $r_v^0 := ID_v$
    if s_v = ruler **then** s_v := competitor **end if**
    j:=0
    repeat
      {Competition start} j:= j+1
      if s_v = competitor **then**
        Select competitor $u \in N(v)$ with $r_u^{j-1} = \min_{w \in N(v)} r_w^{j-1}$
        $r_v^j := \max(\{k | (y_v^k > y_u^k) \wedge (r_v^{j-1} > r_u^{j-1})\} \cup \{0\})$
      end if
      Execute **Update State** s_v {Competition end}
    until $\nexists u \in (N(v) \cup v)$ with s_u = competitor {Phase end}
  until $\nexists u \in (N(v) \cup v)$ with s_u = ruler
until $s_v \in$ {dominator,dominated}

the path to the CDS. Because the size of the set $N^3(v) \cap S$ is at most $f(3)$ and the length of any chosen path is at most 3, at most $3 \cdot f(3) \cdot |S| + |S|$ nodes form the CDS. Since S is a constant approximation of the MDS in a bounded-independence graph, we get a constant approximation of the Minimum CDS (MCDS) in $O(\log^* n)$ time. See also [3]. Due to a lower bound [86, 26] of $\Omega(\log^* n)$ to get a constant approximation of an MDS, our algorithm has asymptotically optimal time complexity for the MDS and MCDS problem. (Observe that the lower bound is also valid for the MCDS problem, since a MCDS is a constant approximation of an MDS in a bounded-independence graph.)

5.3.2 PTAS for MDS and MaxIS

By using the clustering technique from [74] together with our MIS algorithm, a $(1+\epsilon)$-approximation for the MDS and MaxIS problem is computed in $O(\log^* n / \epsilon^{O(1)})$ time. More precisely, we use Theorem 5.8 from [74]:

Theorem 29 (Theorem 5.8 [74])**.** *Let $G = (V, E)$ be a polynomially bounded-independence graph. Then, there exist local, distributed $(1+\epsilon)$- approximation algorithms, $\epsilon > 0$, for the MaxIS and MDS problems on G. The number of communication rounds needed for the respective construction of the subsets is $O(T_{MIS} + \log^* n / \epsilon^{O(1)})$, where T_{MIS} is the time to compute a MIS in G.*

Algorithm Update State

1: **if** s_v = competitor **then**
2: Exchange r^j with competing neighbors $T \subseteq N(v)$
3: **if** $\forall t \in T$ holds $r_t^j > r_v^j$ **then**
4: s_v := dominator
5: **else if** $\forall t \in T$ holds $r_t^j \geq r_v^j$ **then**
6: s_v := ruler
7: **end if**
8: **end if**
9: Exchange state s with all neighbors $t \in N(v)$
10: **if** $\exists\ t \in N(v)$ with s_t = dominator **then**
11: s_v := dominated
12: **else if** ($s_v \neq$ ruler) $\wedge$ ($\exists\ t \in N(v)$ with s_t=ruler) **then**
13: s_v := ruled
14: **end if**

5.4 Theoretical Analysis

We analyze Algorithm MIS for bounded-independence graphs as well as for general graphs.

5.4.1 Bounded-independence graphs

The proof for Algorithm MIS is done by showing correctness of the computed MIS first, i.e. dominators are independent and every node has a dominator as a neighbor in case the algorithm finishes. Then we focus on termination and give evidence that a node ends a phase after at most $\log^* n + 2$ competitions. In other words no node can be a competitor for more than $\log^* n + 2$ consecutive competitions without ever changing its state. In addition we prove that after every phase some nodes near a competitor v stop competing with v (at least) until v becomes ruled. As a next step, we prove that after the $f(2)^{th}$ phase every competing node must end up in exactly one clique of rulers and thus in the $(f(2)+1)^{st}$ phase the node in the clique with smallest ID becomes a dominator. Then we show that every non-dominated node and non-dominator has another dominator within hop distance $f(2)+3$ after $O(\log^* n)$ rounds of communication. Since dominators are independent and the number of independent nodes within distance $f(2) + 3$ is constant, it follows that after $O(\log^* n)$ the MIS is computed.

Lemma 30. *No dominators can be adjacent. On termination of Algorithm MIS every node is either a dominator or must have at least one dominator*

as a neighbor.

Proof. When a node v becomes a dominator, no neighbor $u \in N(v)$ turns into a dominator in the same competition, since result r_v is smallest for all neighbors.

When a node v becomes a dominator, no neighbor can become a dominator or a ruler in a later competition. This follows from the facts that a dominator does not alter its state and that all neighbors $u \in N(v)$ have s_u = dominated after executing Algorithm Update State. They will remain in that state, as long as they are adjacent to a dominator.

The property that every node gets dominated or is a dominator after the execution of Algorithm MIS follows directly from the condition in line 16. □

The upcoming lemma bounds the number of competitions per phase and furthermore says that all competitors must change their states during a phase. Since nodes that have become rulers immediately start the next one, the following lemma also bounds the time until rulers progress to the next phase.

Lemma 31. *After a phase, i.e. after at most* $\log^* n + 2$ *recursive competitions, no node is a competitor.*

Proof. The first competition is based on the IDs, which have at most $\log n$ bits. The result of the first competition r^1 gives an index of a bit of the ID and thus requires at most $\lceil \log \log n \rceil$ bits. The result r^2 of the second competition is a number less than $\lceil \log \log n \rceil$ and uses at most $\lceil \log \log \log n \rceil$ bits etc. In general r^j needs up to $\lceil \log^{(j+1)} n \rceil$ bits. By definition after $\log^* n + 1$ competitions the result is a single bit, i.e. 0 or 1. If $(r_v^{\log^* n+1} = 0) \vee \big(\forall u \in N(v)$ holds $r_u^{\log^* n+1} = 1\big)$ then $s_v \in$ {dominator,ruler} else $s_v \in$ {dominated,ruled}. Thus every node becomes a non-competitor once. Since within the loop (lines 7 to 14) no node turns from a non-competitor into a competitor, the lemma follows. □

The next Lemma 32 essentially guarantees that phases are started and executed locally synchronously. Recall that once a node has become ruled, it stays quiet until all its neighbors are ruled or dominated and then starts the algorithm again as competitor in the first phase (of becoming a dominator).

Lemma 32. *If node v is a competitor in the i^{th} phase (of becoming a dominator) all competing neighbors must also be in the i^{th} phase. Moreover, for a node v executing the j^{th} competition, all competing neighbors must also execute the j^{th} competition.*

Proof. Since we assume synchronous wake-up the very first competition and phase is started by all nodes in parallel. So assume all nodes execute the same phase and the same competition. As long as node v is a competitor all neighbors must be competing in the same competition or be dominated. If node v has been ruled and starts again with the first competition of the first phase (of becoming a ruler), then all neighbors must be ruled or dominated as well and thus if a neighbor starts a new phase it must also be the first phase and the first competition. If a node becomes a ruler in the j^{th} competition of phase i, then all neighbors $u \in N(v)$ that have become rulers in the same competition, will start phase $i+1$ concurrently. All other neighbors must be ruled or dominated and thus cannot start a new phase until all neighbors become ruled or dominated. □

Definition 2. *Let the set $U \subseteq V$ be a connected set of competitors of maximal size, s.t. no competitor $w \notin U$ is a neighbor of a node $v \in U$. For any vertex v, let U_v^i denote a connected set of competitors of maximal size that contains v in the beginning of the i^{th} phase of node v.*

Lemma 33 ensures that all nodes of a set of connected rulers have the same result r (and the results in the previous competition have the same prefix). Afterwards, Lemma 34 shows that in each phase some nodes near every ruler stop competing with it.

Definition 3. *A node $u \in V$ can be reached by a path p of rulers from v in competition j of phase i, if $\exists p = (v = t_0, t_1, ..., u = t_q)$, s.t. $\forall (0 \leq k < q)$ holds that node t_k has become a ruler in competition j of phase i.*

Lemma 33. *If nodes U_v^i with $i > 0$ became rulers in competition j in phase $i-1$ of node v then for any node $u \in U_v^i$, r_u^j is the same as r_v^j, and the prefixes of r_v^{j-1} are the same as those of r_u^{j-1}, i.e. $y_v^i = y_u^i$ for $r_j^v < i \leq \log n$*

Proof. Assume u was reachable by the path $p = (v = t_0, t_1, ..., u = t_q)$ of rulers from v and $r_v^j \neq r_u^j$. Due to the maximality of U_v^i (see Definition 2) all rulers t_i are in U_v^i, i.e. $t_i \in U_v^i$ for $0 \leq i < q$. By assumption there would have to exist two neighboring rulers $t_l, t_{l+1} \in U_v^i$ with $r_{t_l}^j \neq r_{t_{l+1}}^j$. Since either $r_{t_l}^j > r_{t_{l+1}}^j$ or the other way round, they could not both have become rulers in the same competition (This would contradict Lemma 32). Assume their prefixes differed, i.e. $y_v^i \neq y_u^i$ for $r_v^j < i \leq \log n$. Then r_v^j could not be equal to r_u^j. □

The next lemma gives evidence that for a ruler v after every phase one node w at hop distance two and all its neighbors $N(w)$ will not compete with v (at least) until it gets ruled. Since there are at most $f(2)$ of such nodes w (see Lemma 37) only $f(2)+1$ phases are needed until a ruler has no two

hop neighbors and thus must be in a clique. After the first competition of the next phase a dominator is chosen in every clique (see Lemma 38) and all other nodes in the clique are dominated.

Definition 4. *Let the set $W_v^i \subseteq U_v^i$ be the set of nodes at distance two from node v that compete with v in phase i, i.e. $W_v^i := (N^2(v) \setminus N(v)) \cap U_v^i$.*

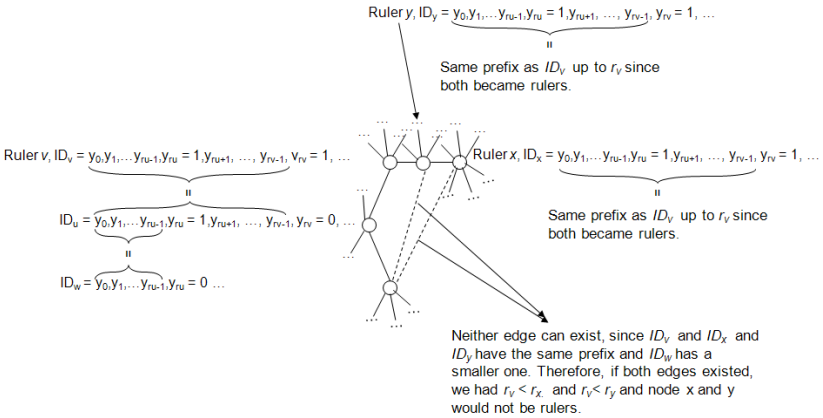


Figure 5.2: Graph of some nodes that participated with node v in the first competition. Assume nodes v, x, y became rulers and furthermore $ID_v > ID_u > ID_w$. In this case w cannot be reachable by a path P of rulers, such as $P = (v, y, w)$ or $P = (v, y, x, w)$.

Lemma 34. *Consider a node v in phase i, which has become a ruler in the j^{th} competition for any $j \geq 0$. If $|W_v^i| > 0$ then $\exists w \in W_v^i$, which cannot be reached by a path of rulers from v.*

Proof. Case $j = 0$, i.e. let us investigate the first competition, which is based on IDs. Consider the value r_v^1 of node v. If $r_v^1 = 0$, then v has a smaller ID than all its neighbors and thus is a dominator. If $r_v^1 = \log n$, then by definition $y_v^{\log n} = 1$ and node v must have had a neighbor u with $y_u^{\log n} = 0$. This neighbor u must have $r_u^1 < \log n$ and thus v cannot be a ruler. So assume $r_v^1 \in [1, \log n - 1]$.

Due to Lemma 33 all rulers s reachable by a path of rulers from v must have $r_s^j = r_v^j$ and their values r^{j-1} must have the same prefix as v. By definition of r_v^j there must exist a node $u \in N(v)$, s.t. $y_v^i = y_u^i$ for $r_v^j < i \leq \log n$ and $1 = y_v^{r_v^j} > y_u^{r_v^j} = 0$. Thus $ID_v > ID_u$. Since nodes u and v differ in

position r_v^1, we have $r_u^1 \neq r_v^1$. Since v is a ruler, $r_v^1 < r_u^1$. Because $r_v^1 < r_u^1$ and $ID_v > ID_u$, this neighbor u must itself have a neighbor w with $ID_w < ID_u$ and $y_w^i = y_u^i$ for $r_u^1 < i \leq \log n$ and $1 = y_u^{r_u^1} > y_w^{r_u^1} = 0$. Apart from that, v and u have an ID with the same prefix from bit (at most) $\log n$ down to bit $r_v^1 + 1$. The node w cannot be a neighbor of any ruler $x \in U_v^i$ with value $r_x^1 = r_v^1$, since otherwise $r_v^1 \geq r_u^1$ because $1 = y_v^{r_u^1} = y_u^{r_u^1} > y_w^{r_u^1} = 0$, i.e. the prefix of w is smaller than that of v. See Figure 5.2.

Case $j > 0$, i.e. let us look at the j^{th} competition for $j > 0$. Assume node v was competing in all previous competitions and was in particular not a ruler after the $(j-1)^{st}$ one. The arguments are similar to those of the first competition.

Assume $0 < r_v^j \leq \log^{(j)} n$ then the same reasoning applies as for the first competition – only the value for r_v^1 has to be substituted by r_v^j, ID_v by r_v^{j-1} and $\log n$ by $\log^{(j)} n$.

Assume $r_v^j = 0$. Since v was not a ruler (or dominator) in competition $j-1$, there exists a neighbor $u \in N(v)$ with $r_u^{j-1} < r_v^{j-1}$. Neighbor u cannot participate in competition j, since otherwise by definition $r_v^j > 0$. Since v is a ruler, u must have become dominated or ruled in competition $j-1$ by a neighbor $w \in N(u)$. If w became a dominator in round $j-1$, all neighbors $s \in N(w)$ became dominated in round $j-1$ as well. Thus w cannot be reached by a path of rulers from v. If w turned into a ruler in competition $j-1$ and v in competition j, then due to Lemma 32 nodes w, v cannot be in the same connected set of rulers and thus node w cannot be reached by a path of rulers from v. □

Figure 5.3 illustrates that no edge can exist between a node w having been a ruler in competition $j-1$ and a ruler v in current competition j, since otherwise node v would have been already ruled (or dominated).

Lemma 35 shows that every connected set of competitors, containing node v in phase j, must be a subset of a previous connected set of competitors, containing node v in phase i with $i < j$. This will be used by Lemma 36 to show that if a set of arbitrary nodes does not have a common node with a set of connected competitors in some phase, then this will hold for all proceeding phases.

Lemma 35. *If node v has been either a ruler or a competitor during j phases, then $U_v^j \subseteq U_v^i$ for $i < j$.*

Proof. Let the neighborhood $N(T)$ of a set of nodes $T \subseteq V$ be the set $\{s \mid \exists u \in T : s \in N(u) \setminus T\}$.

Let $w \in U_v^i$ be the (or one of the) first node(s) that become a ruler or dominator in phase i (say in competition k). By definition all nodes $N(U_v^i)$ have been ruled or dominated in competition k and thus cannot become

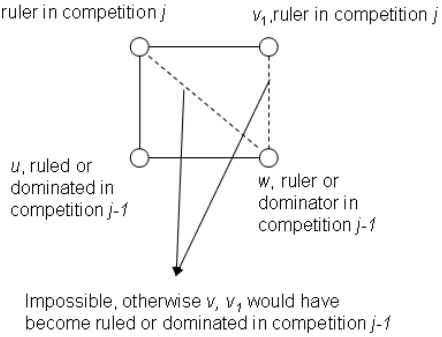


Figure 5.3: Graph of some nodes that participated with node v in competition $j-1$ and j

rulers. Thus $U_w^{i+1} \subseteq U_v^i$ and all neighboring nodes $N(U_w^{i+1})$ of U_w^{i+1} are ruled or dominated as well. Consider the next competition l with $l > k$, where some node $t \in (U_v^i \setminus (U_w^{i+1} \cup N(U_w^{i+1}))$ becomes a dominator or ruler. Ten we have that $U_v^{i+1} \subseteq (U_v^i \setminus (U_w^{i+1} \cup N(U_w^{i+1})) \subseteq U_v^i$. The argument proceeds in the same manner. Thus we have that $U_v^{i+1} \subseteq U_v^i$. Analogously, it follows that $U_v^{i+2} \subseteq U_v^{i+1} \subseteq U_v^i$ a.s.o. □

Lemma 36. *For a set $T \subset V$, s.t. $U^i \cap T = \emptyset$ holds that $U_v^j \cap T = \emptyset$ with $i < j$.*

Proof. Due to Lemma 35, we have that $U_v^j \subseteq U_v^i$. Due to the disjointedness of U_v^i and T, U_v^j and T must also be disjoint. □

The next two lemmas together give an upper bound of the number of (consecutive) phases until a node becomes a dominator.

Lemma 37. *If a node v has become a ruler in the $f(2)^{th}$ phase, then it is in a clique of competitors in phase $f(2)+1$.*

Proof. Let a node w, as defined in Lemma 34, for phase i be denoted by $w_i \in W_v^i$. Lemma 34 implies that no neighbor $t \in N(w_i)$ can be a ruler reachable by a path of rulers from v. Thus by definition $U_v^{i+1} \cap N(w_i) = \{\}$. Due to Lemma 36, no node $t \in N(w_i) \cup w_i$ will be reachable by a path of competitors from v until (at least) v has become ruled. Since by definition $W_v^i \subseteq U_v^i$, this implies that $W_v^{i+1} \subseteq (W_v^i \setminus (N(w_i) \cup w_i))$. As a consequence nodes $w_i \in W_v^i$ and $w_k \in W_v^k$ with $i \neq k$ (i.e. from different phases) are

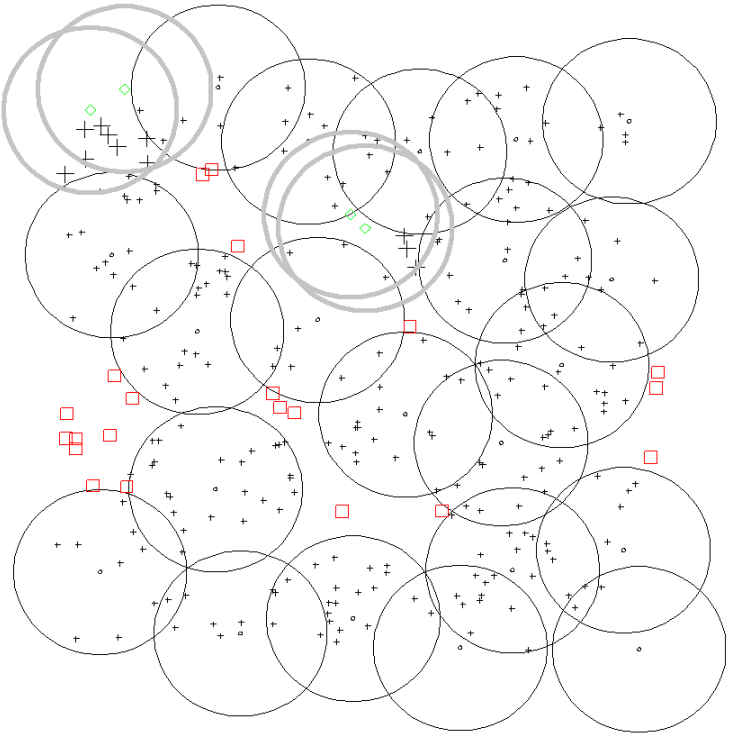


Figure 5.4: Algorithm MIS on an instance of a UDG. It shows the state of each node after the very first competition. For dominators and rulers a circle indicating the transmission range is shown. Rulers are depicted by diamonds, ruled nodes by big crosses, dominated nodes by small crosses, dominators by small circles and competitors by boxes. As can easily be seen, most nodes are already dominated after the first competition. After the second there are no competitors left and after the first competition of the next phase the algorithm is done.

independent. The size of a maximum independent set in $N^2(v)$ is upper bounded by $f(2)$. In every phase i, at least one node $w_i \in W_v^i$ at distance two from v is removed. Thus after at most $f(2)$ phases, node v cannot reach any competitor at hop distance at least 2 by a path of competitors, i.e. $W_v^{f(2)+1} = \{\}$ and the nodes $U_v^{f(2)+1}$ form a clique. □

Lemma 38. *If a node v is still competing in the $(f(2)+1)^{st}$ phase then either v or a neighbor of v will become a dominator in that phase.*

Proof. Using Lemma 37, we have that each competitor v is in a clique in the beginning of the $(f(2)+1)^{st}$ phase. Thus in the first competition the node with the smallest ID of the clique will become a dominator. □

Next we show that for every non-dominated node v a dominator is chosen within constant distance from v. Essentially this is because a node cannot end a phase as a ruled node without having a ruler advancing to the next phase in its neighborhood (see Algorithm Update State). Due to Lemma 38 after a constant number of phases a node must become a dominator or dominated.

Since dominators are independent (see Lemma 30) and in a bounded-independence graph the number of independent nodes within constant distance is also a constant, only $O(\log^* n)$ rounds of communication are needed (see Theorem 40).

Lemma 39. *After $O(\log^* n)$ rounds of communication each node v either becomes a dominator or there exists an additional node that has become a dominator within hop distance $f(2)+3$.*

Proof. We will show that the distance between a ruler in phase i and node v is at most i. After the first phase, every node v is a ruler itself or adjacent to a ruler.

Assume the distance was at most $i-1$ after the $(i-1)^{st}$ phase. In the i^{th} phase only rulers become competitors again (line 5) and participate in the competitions. Thus after the i^{th} phase, every competitor will become a ruler or a dominator or have at least one of the two in its neighborhood. Thus the distance between a ruler and a ruled node grows at most by 1 per phase.

Due to Lemma 38 every competitor or one of its neighbors must become a dominator in the $(f(2)+1)^{st}$ phase. Assume node $u \in U_v^0$ started competing with v. Since a phase has at most $\log^* n + 2$ competitions (see Lemma 31) and to perform a competition the algorithm requires three communication rounds, after at most $3 \cdot (f(2)+1) \cdot (\log^* n + 2) \in O(\log^* n)$ rounds, node u must have got a dominator within distance $f(2)+2$. Assume node u is ruled, then at least one of its neighbor must be competing or it starts competing

again itself. Thus after at most $O(\log^* n)$ rounds of communication it must have got an additional dominator within hop distance $f(2)+3$. □

Theorem 40. *The total time to compute a MIS is in $O(f(f(2)+3)\log^* n)$ and each message is of size $O(\log n)$.*

Proof. Due to Lemma 39 within $O(\log^* n)$ every node gets a dominator within distance $f(2)+3$. Since dominators are independent (Lemma 66), the number of dominators within distance $f(2)+3$ is upper bounded by the size of a maximum independent set in $N^{f(2)+3}(v)$, which is $f(f(2)+3)$. This yields that $f(f(2)+3)\cdot O(\log^* n)$ rounds of communication are needed.

For initialization all IDs (of maximum length $\log n$) have to be exchanged among neighbors requiring one communication round for the first competition to execute. The following update of the state needs every message to be of size $O(\log\log n)$ and takes three rounds of communication. Namely, exchanging the result of the prior competition which also serves as input for the next. Additionally, a request and possibly delayed reply of the current state of all neighbors. Apart from that no communication has to take place for initialization and the first competition. In an analogous derivation the second competition requires only messages of size $O(\log\log\log n)$ etc. and three communication rounds. Thus each message is of size at most $O(\log n)$. □

5.4.2 General Graphs

For a general graph the size of a MaxIS in the neighborhood $N^r(v)$ of a node v can be bounded by a function $g(|N^r(v)|)$, since the size of a MaxIS including nodes up to distance r cannot be larger than the size of the neighborhood $N^r(v)$.

Theorem 41. *Let the function $g(\Delta^r)$ be such that for each node $v \in V$, the size of a MaxIS in the neighborhood $N^r(v)$ is at most $g(\Delta^r)$, $\forall r \geq 0$. Then algorithm MIS needs at most $O(\min(n, g(g(\Delta^2)+3))$ time.*

Proof. To see that the running time is in $O(g(g(\Delta^2)+3))$ the same analysis as in Section 5.4.1 can be used. Thus let us focus on the case that the number of rounds is also in $O(n)$.

While not all nodes are dominated or dominators, there always exists at least one set U of connected competitors (or rulers that become competitors immediately). Consider the connected set of competitors U_v^i (see Definition 2) for phase i. For a competition we can distinguish two outcomes: First, all nodes $u \in U_v^i$ become rulers, ruled, dominated or dominators. Second, at least one node $w \in U_v^i$ remains a competitor and at least one node $u \in U_v^i$ becomes a ruler or a dominator. (Observe, that it is not possible that all nodes remain competing, since the node with minimum result will become a

ruler or dominator.) In case, some nodes $W \subset U_v^i$ became rulers, say node $s \in U_v^i$ for instance, then due to Lemma 34 in phase $i+1$ there exists a node $w \in W \setminus U_s^{i+1}$. Therefore, there exist two independent non-empty sets U_s^{i+1} and $U_v^i \setminus U_s^{i+1}$ (at least $w \in U_v^i \setminus U_s^{i+1}$). Apart from that the set $U_v^i \setminus U_s^{i+1}$ either contains a dominator or a ruler. In the next competition (the first of phase $i+1$) the node with minimum ID in U_s^{i+1} becomes a dominator and if the set $U_v^i \setminus U_s^{i+1}$ contained a ruler also the node with minimum ID in this set. Assume for phase i it takes $j > 1$ competitions until all nodes in U_v^i become non-competitors. The number of dominators $u \in U_v^i$ obtained during the last $j-1$ competitions in phase i plus the number of dominators in the first competition of phase $i+1$ of nodes $u \in U_v^i$, is at least $\max\{j-1, 1\}$. Thus, in the worst case all nodes $u \in U_v^i$ become non-competitors after two competitions and only one dominator can be accredited to these two competitions which yields at most $2 \cdot n$ competitions. To perform a competition the algorithm requires three communication rounds. □

The running time can indeed be $O(n)$ as shown by the following graph: Each node v with $0 < ID_v < n-1$ has edges to nodes u, w with $ID_u = ID_v - 1$ and $ID_w = ID_v + 1$. A node v having its last two bits equal to 1 (i.e. $ID_v \bmod 100 = 11$) is additionally connected to all nodes with higher ID and to the node u with $ID_u = ID_v - 2$ (see Figure 5.5). The states of the nodes during the execution of the algorithm follow a pattern which repeats every three rounds of communication. In the first round the node with smallest ID (with the last two bits being 00) becomes a dominator and its neighbor becomes dominated (its ID ends with 01). All other nodes remain competitors. In the second round the two nodes with smallest IDs (having the last two bits 10 and 11) change to rulers and all others become ruled. In the third round the node with smallest ID (ending with 10) turns into a dominator and its neighbor becomes dominated. All ruled nodes switch back to competitors again.

5.4.3 $\Delta+1$ Coloring

We state two methods for computing a $\Delta+1$ coloring, both relying on the same observation that a node v can color all its neighbors if no other node $u \in N^3(v)$ does so at the same time. See Figure 5.6.

In the first procedure a node v competes against a neighbor $u \in N^3(v)$, i.e. we compute a MIS S on the graph $G' = (V, E')$ with $E' = E \cup \{(u,v) | \{(u,s),(s,v)\} \subseteq E\} \cup (\{(u,v) | \{(u,s),(s,t),(t,v)\} \subseteq E\})$. All MIS nodes $v \in S$ color all their neighbors in G and themselves, taking into account already used colors of colored nodes $w \in N^2(v)$. This procedure is repeated for all uncolored nodes. In each iteration i a MIS S_i is computed

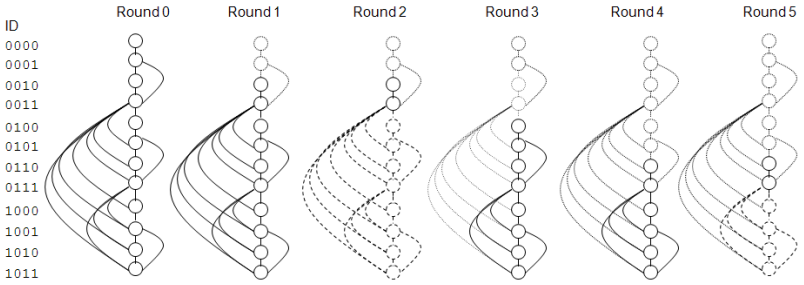


Figure 5.5: Algorithm MIS running on a unbounded-independence graph, taking time $O(n)$. The solid lines show competitors. The dotted lines are dominators and dominated nodes. The dashed lines indicate ruled nodes.

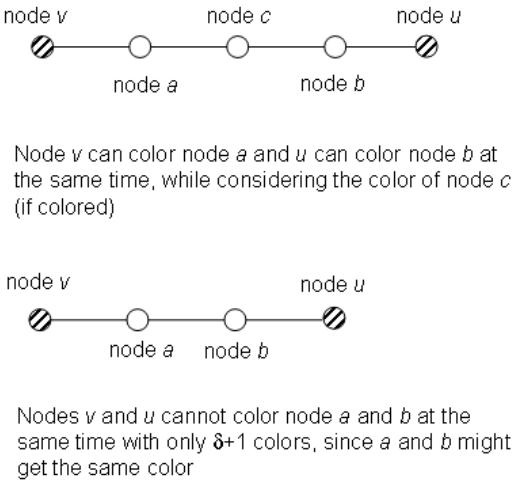


Figure 5.6: Illustration showing that the distance between two nodes concurrently coloring their neighbors must be at least 4

and an uncolored node $u \in V$ either gets colored or has distance at most three in G to a node $v \in S_i$. The union of MIS S_i and S_j with $i \neq j$ forms an independent set in G. The number of independent nodes for node v at distance at most three is bounded by $f(3)$. Thus after at most $f(3)$ computations of a MIS in G' node v gets colored. The graphs G' is also bounded-independence with $f'(r) \leq f(3 \cdot r)$, yielding an overall running time of $O(\log^* n)$. Due to Linial's $\Omega(\log^* n)$ [88] lower bound our algorithm is asymptotically optimal. Observe that in order to compete against all nodes in $N^k(v)$ messages of size $O(\log n)$ are sufficient, since a node only needs to know the minimum result of a competition. At first a node broadcasts its own result to all neighbors and from then on forwards the smallest result it received so far for $k-1$ rounds. A distance two coloring with the same message and time complexity can be obtained, when every node v competes against all nodes $u \in N^6(v)$ instead of $N^3(v)$ and colors its two hop neighborhood.

Alternatively to the above method a MIS S on $G = (V, E)$ can be computed first. Next we consider the graph G' defined by nodes S and edges between two nodes $u, v \in S$, if they can reach each other by a path of length at most three, i.e. $G' = (S, E')$ with $E' = \{(u, v) | u, v \in S \wedge \big((\exists s \in (V \setminus S)(\{(u, s), (s, v)\} \subseteq E) \vee (\exists s, t \in (V \setminus S)(\{(u, s), (s, t), (t, v)\} \subseteq E)\big)\}$. We compute a MIS $S' \subseteq S$ on G'. Every node $v \in S'$ colors all its neighbors, respecting already used colors. The process is repeated for all uncolored nodes. Thus in an iteration a node v either gets colored or a neighbor $u \in N^4(v) \cap S'$ colors itself and all its neighbors. Since colored nodes are not considered any more, for a node v there are at most $f(4)$ such neighbors $u \in N^4(v)$. Therefore in total at most $2 \cdot f(4)$ MIS computations are required, giving an overall running time of $O(\log^* n)$.

5.4.4 Maximal matching

We can use the same idea as for the coloring in Section 5.4.3. We compute a MIS S such that all nodes in the MIS have distance at least 6. This allows a node v in the MIS S to compute a maximal matching for all nodes $u \in N(v) \cup v$, i.e. it can pick any edge $e = (u, w)$ with $w \in N^2(v)$ and add it to the matching. All nodes adjacent to an edge in the matching and those that have no unmatched neighbors are removed and the process repeats, i.e. again a MIS S' is computed for the remaining nodes, such that all nodes in the MIS have distance at least 6.

Whenever a MIS S is computed, any node is either matched and stops or a node within distance 6 matches all its neighbors. Since the number of nodes that can match all their neighbors within distance 6 corresponds to the size of a maximum independent set, we need to compute at most $O(f(6))$ MIS S, giving an overall running time of $O(\log^* n)$.

5.4.5 Others

Our algorithm MIS has been used in the area of self-assembling systems [116], to compute a PTAS for minimum clique partition in the UDG model [100], to approximate the robot assignment [20] and the facility location problem [98] just to name a few.

5.5 Experimental Results

We compared three algorithms namely Algorithm MIS, Luby's [90] and Algorithm MAX, where a node joins the MIS if its ID is maximum among all neighbors. In fact, we slightly adapted algorithm MIS, such that before it performs the first competition, it updates the state based on the IDs, i.e. all nodes become dominators that have minimum ID among their neighbors and dominated nodes inform their neighbors. For a competition our algorithm needs three communication rounds: One to exchange the result of a competition and two more to let all neighbors know about the new state. Algorithm MAX needs two rounds to inform all neighbors about the new states. Both algorithms MIS and MAX need an initial round to exchange the IDs among the neighboring nodes. For Luby's algorithm the update of the states takes two rounds and we assumed that one initial round is needed to get to know the degree of the nodes.

We considered random graphs, where an edge between two nodes was added with probability p. Our experiments were conducted for UDG and random graphs with 1500 nodes. They indicate that Luby's algorithm performs worst in general. Algorithm MIS and MAX perform relatively similar (see Figures 5.7 and 5.8), however Algorithm MAX lacks good worst case guarantees for UDG graphs. In particular for loosely connected networks, the chance that there are long chains of nodes with increasing IDs increases with the network size and thus algorithm MAX performs worse in such scenarios.

Due to the random arrangement of nodes the first selection of dominators in Algorithm MIS and MAX is similar to the one of Luby, except for the fact that Algorithm MIS and MAX do not face the problems of multiple neighbors joining the MIS at the same time and that no nodes join at all. If there are many small (unconnected) subgraphs of nodes not covered by the MIS, Luby's algorithm has a high chance that at least for some of them, no node decides to join the MIS, whereas Algorithm MIS and MAX both will select at least one.

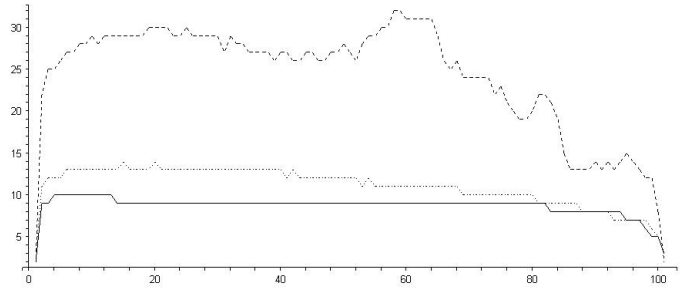


Figure 5.7: Simulations on Erdős Rényi graphs of 1500 nodes; The probability of an edge between two nodes is indicated on the x-axis. The y-axis shows the number of communication rounds. The dashed line shows the rounds needed for Luby's algorithm, the dotted one for Algorithm MAX and the solid one for our Algorithm MIS.

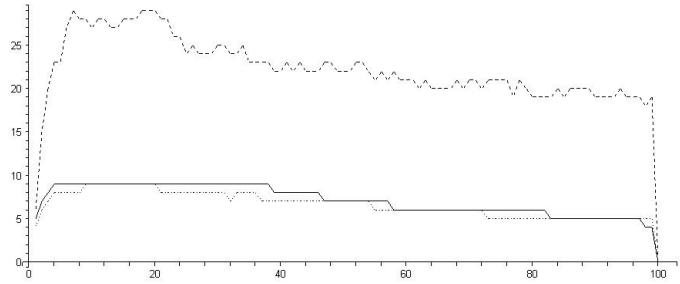


Figure 5.8: Simulations on UDGs of 1500 nodes. The probability of an edge between two nodes is indicated on the x-axis. The y-axis shows the number of communication rounds. The dashed line shows the rounds needed for Luby's algorithm, the dotted one for Algorithm MAX and the solid one for our Algorithm MIS.

Chapter 6

Trading among Complexity Measures

6.1 Introduction

The efficiency of a distributed algorithm is assessed with at least one out of three classic distributed complexity measures: time complexity (number of rounds for synchronous algorithms), communication or bit complexity (total number of bits transmitted), and message complexity (total number of messages transmitted). Depending on the application, one or another measure might be more relevant. Generally speaking, time complexity has received most attention; but communication complexity (bandwidth constraints) or message complexity (accounting for message overhead) play a vital role as well. One cannot just ignore one of the measures, as there are tradeoffs: One may for instance sometimes cut down on time by exchanging larger messages. Alternatively, one may save messages and bits by communicating "silently". Two parties may for instance communicate for free by telephone by simply never picking up the phone, and instead letting the phone ring for a long time when transmitting a binary 1, and just a short time for a binary 0. A more sophisticated example for silent communication employs time-coding to communicate information through time. As illustration consider pulse-position modulation, as used in wireless and optical communication. A k-bit message can be dispersed over time by encoding the message with a single pulse in one of 2^k possible slots. Employing a single pulse within time t allows to communicate at most $\log t$ bits.[1] Reducing message complexity is harder in general, and in some cases impossible as there are dependencies

[1] The amount of information that can be communicated follows directly from our bound.

between messages. We identify mechanisms for symmetry breaking that cut both message and bit complexity by a factor of $\log t$, even though multiple messages cannot be combined into a single message through time-coding.

Although it is well-known that these dependencies exist, to the best of our knowledge the tradeoffs are not completely understood. A considerable amount of work deals with both message size and time complexity. These two measures give a (rough) bound on the bit complexity, e.g. time multiplied by message size as an upper bound. However, we show that for a given fixed bit complexity allowing many arbitrary small messages (i.e. consisting of 1 bit) compared to allowing only one large message might cause a drastic (up to exponential) gain in time. For some examples both the time and overall message complexity, i.e. the messages transmitted by all nodes, have been optimized, but the tradeoffs between all three have not been investigated to the best of our knowledge.

After stating the model and related work in Section 6.4 we answer questions like "If we can prolong an algorithm by a factor t in time and can increase the number of messages by a factor m, what is the effect on the bit complexity b?" We give a tight bound on the amount of information exchangeable between two nodes of $\Theta(m \log(tb/m^2) + b)$ bits for $m < \sqrt{bt}$ and $\Theta(b)$ for larger m. A bound on the (communicable) information together with a bound on the minimum required information that has to be exchanged to solve a problem yields bounds on the time-complexity. We derive such bounds for typical symmetry breaking problems, such as coloring and maximal independent sets. We show that for $t \in [2, n]$ any MIS and $O(\Delta)$ coloring algorithms using up to $c_0 \log n / \log t$ bits and messages for a constant c_0 require time t. In light of the state of the art upper bounds for unrestricted communication of $\mathrm{O}(\log n)$ using $O(\log n)$ bits for the MIS problem, and $O(\log^* n)$ for the $O(\Delta)$ coloring problem, our lower bound indicates that even a logarithmic factor of $\log t$ in the amount of transmittable bits can make more than an exponential difference in time.

In Section 6.5 we identify two coding schemes, i.e. transformations, to gain a factor $\log t$ in bit as well as message complexity by a time increase of t^c that cannot be achieved with traditional time coding. We employ them to deterministic and randomized coloring and maximal independent set algorithms. Our techniques are applicable beyond these problems, e.g. for certain divide-and-conquer algorithms.

6.2 Related Work

Related work related to coloring and MIS can be found in Section 2.2. In [69] the notion of "bit rounds" was introduced in the context of a coloring algorithm, where a node must transmit either 0 or 1 in one bit round. This *bit*

round complexity is an interesting hybrid between time and bit complexity, particularly useful in systems where sending a single bit does not incorporate a significant protocol overhead. The paper [69] also states a lower bound of $\Omega(\log n)$ bit rounds to compute a coloring on a ring. In contrast, our bit-time complexity model is motivated by the fact that in practice time is frequently divided into slots, i.e. rounds, and nodes might not transmit at all in a slot or they might transmit more than a single bit. In some sense, the bit-time complexity model unifies algorithm complexity focusing on running time and communication complexity. For a survey on communication complexity see [83]. In a common set-up two (or more) parties A, B want to compute a function f that depends on values held by A and B and the goal is to derive bounds on the needed information. For example, [96] shows that for some network topologies (involving more than two nodes) reducing the number of allowed message exchanges by 1 can exponentially increase the time complexity. The amount of exchangeable information between two parties given a certain number of messages and time can be found in e.g. [103]. To the best of our knowledge, we are the first to extend the tradeoff to allow for a variable number of bits per message.

In [32] the communication complexity of breaking symmetry in rings and chains is investigated. In [39] the running time of MIS algorithms is investigated depending on the amount of advice (measured in bits) that nodes are given before the start of the algorithm. For a ring graph it is shown that $\Omega(n/\log^{(k)} n)$ bits of information are needed for any constant k to break the lower bound of $\Omega(\log^* n)$ [88]. At first, asking a global instance, knowing the exact topology of the graph, for advice seems to contradict the distributed approach. But the question regarding the amount of needed information to solve a task is interesting and valuable for enhancing the understanding of distributed algorithms. In particular, since some (aggregate) knowledge of the graph is often necessary for an efficient computation, e.g. the type of the graph.

There is a myriad of papers for different problems that consider two complexity measures. However, whereas many papers concentrate on time complexity and merely mention message size [90, 72, 14, 15], others derive explicit tradeoffs [40, 91, 41].

In a paper by Métivier et al. [91] an algorithm for the MIS problem was stated running in time $O(\log n)$ with bit complexity $O(\log n)$ for general graphs. It improves on the bit complexity of the fastest algorithm [90]. Essentially, each node draws a random number in $[0, n]$ and is joined the MIS, if its number is the smallest. Our MIS algorithm trading time for bit/message complexity improves on [90] through a different technique. For the MIS problem arbitrary large messages do not allow for an arbitrary fast algorithm, i.e. in general graphs every algorithm requires at least $\Omega(\sqrt{\log n/\log\log n})$

or $\Omega(\log \Delta / \log\log \Delta)$ communication rounds for computing a MIS [77]. Interestingly, the opposite is true: An arbitrarily slow algorithm allows for constant message and bit complexity. The lower bound is achieved with a pseudo-symmetric graph, such that a node needs to get to know its neighborhood up to distance $\Omega(\sqrt{\log n / \log\log n})$ or $\Omega(\log \Delta / \log\log \Delta)$.

6.3 Model and Definitions

Here we only state definitions and modeling assumptions in addition to the ones given in Section 2.1. The term "with high probability" abbreviated by w.h.p. denotes the number $1 - 1/n^c$ for an arbitrary constant c. The maximum degree is denoted by Δ and $\Delta_{N_+(v)}$ denotes the maximum degree of a node in $N_+(v)$, i.e. $\Delta_{N_+(v)} := \max_{u \in N_+(v)} d(u)$.

The bit complexity denotes the maximum sum of the number of bits transmitted over any edge during the execution of the algorithm, i.e. if an algorithm has time complexity t then the bit complexity is $\max_{e \in E} \sum_{r=0}^{t-1} b_e(r)$, where $b_e(r)$ denotes the number of bits transmitted over edge e in round r. Analogously, the message complexity denotes the maximum number of messages transmitted over any edge.

The time complexity of a distributed algorithm is traditionally defined as the number of communication rounds until the *last* node completes the algorithm. Somewhat inconsistently, message respectively bit complexity often measure the *total* of all exchanged messages respectively bits (of all nodes) or the expectation of message and bits exchanges of a single node during the execution. Analogous to the definition of time complexity, we consider the worst node only; in other words, the message respectively bit complexity is given by the number of messages or bits exchanged by the most loaded node. Both views have their validity, are commonly used and sometimes even coincide. The focus on a more local "maximum" measure is motivated by the observation that for distributed systems, an individual node might often form a bottleneck, and delay an algorithm, although overall the constraints on bandwidth, energy etc. are fulfilled. For example, if a single node in a battery powered sensor network must transmit much more often than other nodes, it will become non-operational much more quickly. This might have devastating effects on the network topology, e.g. disconnecting it and thereby preventing further data aggregation.

6.4 Bounds on Transmittable Information

In all (reasonable) distributed algorithms nodes must exchange a certain minimum of information with their neighbors. The amount of exchanged information is not only given by the total amount of bits contained in the

messages, but also by the times, when the messages were sent and the size of the messages. In other words, the number of different (observable) behaviors of a node v by a neighbor u, i.e. the number of different ways v can communicate with u, determines the total amount of exchanged information between two nodes. We first bound the number of exchangeable information between two communication parties. By using a lower bound for the minimum needed amount of exchanged information for any kind of problem one therefore gets a lower bound on the time complexity of any algorithm depending on the bits and messages exchanged. We illustrate this general technique by deriving lower bounds for the MIS and coloring problem.

Theorem 42. *If a node executes t rounds using up to $m \leq t$ messages (at most one message per round) with a total of $b \geq m$ bits within messages (i.e. at least one bit per message), it can communicate in total $\Theta(m \log(tb/m^2)+b)$ bits for $m < \sqrt{bt}$ and $\Theta(b)$ bits for $m \geq \sqrt{bt}$.*

Proof. A node can decide not to transmit at all or it can transmit in $n_r \in [1, m]$ rounds $\{r_1, r_2, ..., r_{n_r}\}$ with $r_i \in [0, t-1]$ for $1 \leq i \leq n_r$. In each chosen round r_i the node transmits at least one bit. The total number of choices of rounds is given by $\binom{t}{n_r}$. Say a node wants to transmit n_{r_i} bits in round r_i then the sum of all bits transmitted n_t in all rounds must be at least n_r and at most b, i.e. $n_r \leq n_t := \sum_{i=1}^{n_r} n_{r_i} \leq b$. Thus the number of all possible sequences $(n_{r_1}, n_{r_2}, ..., n_{r_{n_r}})$ with $n_{r_i} \in [1, b - n_r + 1]$ is given by the composition of n_t into exactly n_r parts, i.e. the number of ways we can write n_t as a sum of n_r terms, i.e. $\binom{n_t-1}{n_r-1} \leq \binom{n_t}{n_r}$. Each of the at most n_t transmitted bits can either be 0 or 1, yielding 2^{n_t} combinations. Multiplying, these three terms and adding one for the case that a node does not transmit, i.e. $\binom{t}{n_r} \cdot \binom{n_t-1}{n_r-1} \cdot 2^{n_t} + 1$, gives a bound on the different behaviors of a node for a fixed n_r. Thus, overall the number of behaviors is upper bounded by:

$$1 + \sum_{n_r=1}^{m} \sum_{n_t=n_r}^{b} \binom{t}{n_r} \cdot \binom{n_t}{n_r} \cdot 2^{n_t} \leq 1 + mb \cdot \max_{1 \leq n_r \leq m, 0 \leq n_t \leq b} \binom{t}{n_r} \cdot \binom{n_t}{n_r} \cdot 2^{n_t}$$

$$\leq 1 + mb \cdot \max_{1 \leq n_r \leq m} \binom{t}{n_r} \cdot \binom{b}{n_r} \cdot 2^b$$

The last inequality follows due to $n_r \leq n_t \leq b$. We have $\binom{n}{k} \leq (ne/k)^k$, thus $\binom{b}{n_r} \leq (eb/n_r)^{n_r}$. Continuing the derivation we get:

$$\leq 1 + mb \cdot \max_{1 \leq n_r \leq m} (et/n_r)^{n_r} \cdot (eb/n_r)^{n_r} \cdot 2^b \leq 1 + mb \cdot 2^b \max_{1 \leq n_r \leq m} (e^2 bt/n_r^2)^{n_r}$$

Next we compute the maximum:

$$\frac{d}{dn_r}(e^2bt/n_r^2)^{n_r} = (e^2bt/n_r^2)^{n_r} \cdot (\ln(e^2bt/n_r^2) - 2) = 0$$

$$\Leftrightarrow \ln(e^2bt/n_r^2) - 2 = 0 \Leftrightarrow e^2bt/n_r^2 = e^2 \Leftrightarrow n_r = \sqrt{bt}$$

For $m \geq \sqrt{bt}$ we get: $1+mb\cdot 2^b \max_{1\leq n_r\leq m}(e^2bt/n_r^2)^{n_r} \leq (m+1)\cdot 2^b\cdot(e^2)^{\sqrt{bt}}$
For $m < \sqrt{bt}$: $1 + mb \cdot 2^b \max_{1\leq n_r\leq m}(e^2bt/n_r^2)^{n_r} \leq (m+1)(b+1)\cdot 2^b(e^2bt/m^2)^m$

Taking the logarithm yields the amount of transmittable information being asymptotically equal to $O(m\log(tb/m^2)+b)$ for $m < \sqrt{bt}$, since $t \geq m$ and $b \geq m$ and $O(b+\sqrt{bt}) = O(b)$ for $b \geq m \geq \sqrt{bt}$.

With the same reasoning as before a lower bound can be computed. We use $\binom{n}{k} \geq (n/k)^k$ we have $\binom{b-1}{n_r-1} \geq ((b-1)/(n_r-1))^{n_r-1}$.

$$1+\sum_{n_r=1}^{m}\sum_{n_t=n_r}^{b}\binom{t}{n_r}\cdot\binom{n_t-1}{n_r-1}\cdot 2^{n_t} \geq \max_{1\leq n_r\leq m, 0\leq n_t\leq b}\binom{t}{n_r}\cdot\binom{n_t-1}{n_r-1}\cdot 2^{n_t}$$

$$\geq \max_{1\leq n_r\leq m}\binom{t}{n_r}\cdot\binom{b-1}{n_r-1}\cdot 2^b \geq 2^b \max_{1\leq n_r\leq m}(t/n_r)^{n_r}\cdot((b-1)/(n_r-1))^{n_r-1}$$

$$\geq 2^b \max_{1\leq n_r\leq m}((b-1)t/((n_r-1)n_r))^{n_r-1} \geq 2^b \max_{1\leq n_r\leq m}((b-1)t/n_r^2)^{n_r-1}$$

Next we compute the maximum:

$$\frac{d}{dn_r}((b-1)t/n_r^2)^{n_r-1} = ((b-1)t/n_r^2)^{n_r-1}\cdot(\ln(((b-1)t/n_r^2)) - 2 + 1/n_r) = 0$$

$$\Leftrightarrow \ln((b-1)t/n_r^2) - 2 + 1/n_r = 0 \Leftrightarrow (b-1)t/n_r^2 = e^{2-1/n_r} \Leftrightarrow n_r = \sqrt{(b-1)t}/e^{1+1/(2n_r)}$$

For $m \geq \sqrt{(b-1)t}/e^{1+1/(2n_r)}$ we have: $2^b \max_{1\leq n_r\leq m}((b-1)t/n_r^2)^{n_r-1} \geq 2^b(e^2)^{\sqrt{(b-1)t/e^2-1}}$
For $m < \sqrt{(b-1)t}/e^{1+1/(2n_r)}$: $2^b \max_{1\leq n_r\leq m}((b-1)t/n_r^2)^{n_r-1} \geq 2^b(e^2(b-1)t/m^2)^{m-1}$

This, yields $\Omega(m\log(tb/m^2)+b)$ for $m < \sqrt{(b-1)t}/e^{1+1/(2n_r)}$, since $t \geq m$ and $b \geq m$ and $\Omega(b+\sqrt{bt}) \geq \Omega(b)$ for $m \geq \sqrt{(b-1)t}/e^{1+1/(2n_r)}$.

Overall, the bounds become $\Theta(m\log(tb/m^2)+b)$ for $m < \sqrt{bt}$ and $\Theta(b)$ otherwise.

□

Corollary 43. *The amount of information that k parties can exchange within t rounds, where each party can communicate with each other party directly and uses up to $m \leq t$ messages (at most one message per round) with a total of $b \geq m$ bits (i.e. at least one bit per message) is $\Theta(k \cdot (m \log(tb/m^2) + b))$ for $m < \sqrt{bt}$ and $\Theta(kb)$ bits for $m \geq \sqrt{bt}$.*

Proof. Compared to Theorem 42, where one node A only transmits data to another node B, the number of observable behaviors if k nodes are allowed to transmit is raised to the power of k, i.e. if one node can communicate in x different ways then the total number of observable behaviors becomes x^k. Taking the logarithm gives the amount of exchangeable information for k parties, i.e. $\log(x^k) = k \log x$, where $\log x$ is the amount of information a single node can transmit as stated in Theorem 42. □

6.4.1 Lower Bound on the Time Complexity Depending on the Bit (and Message) Complexity

We first bound the amount of information that must be exchanged to solve the MIS and coloring problem. Then we give a lower bound on the time complexity depending on the bit complexity for any MIS and coloring algorithm where a message consists of at least one bit.

Theorem 44. *Any algorithm computing a MIS (in a randomized manner) in a constant degree graph, where each node can communicate less than $c_0 \log n$ bits for some constant c_0 fails (w.h.p.).*

The intuition of the so called "fooling set" argument proof is as follows: If a node cannot figure out what its neighbors are doing, i.e. it is unaware of the IDs of its neighbors, its chances to make a wrong choice are high.

Proof. Let us look at a graph being a disjoint union of cliques of size 2, i.e. every node has only one neighbor. A node can communicate in up to $2^{c_0 \log n} = n^{c_0}$ distinct ways. In the deterministic case, let $B_u \in [0, n^{c_0} - 1]$ be the behavior that a node u decides on given that it sent and received the same information throughout the execution of the algorithm. Clearly, before the first transmission no information exchange has occurred and each node u fixes some value B_u. Since there are only n^{c_0} distinct values for n nodes, there exists a behavior $B \in [0, n^{c_0} - 1]$, which is chosen by at least n^{1-2c_0} nodes given that they sent and received the same information.

Consider four arbitrary nodes $U = \{u, v, w, x\}$ that receive and transmit the same information. Consider the graph with $G' = (U, \{(u, v), (w, x)\})$ where u, v and also w, x are incident, $G'' = (U, \{(v, w), (u, x)\})$ and $G''' = (U, \{(u, w), (v, x)\})$. Note that u, v, w, x have no knowledge about the identity of their neighbors (They only know that their degrees are 1). Assume a

deterministic algorithm correctly computes a MIS for G' and v is joined the MIS, then u is not joined the MIS in G' but also not in G''', since it cannot distinguish G' from G'''. Thus w must join the MIS to correctly compute a MIS for G'''. Therefore, both v, w are joined the MIS S in G'' and thus S violates the independence condition of a MIS.

For the randomized case the argument is similar. Before the first transmission all nodes have received the same information. Since there are only n^{c_0} distinct behavior for n nodes at least a set S of nodes of cardinality $|S| \geq n^{1-c_0}$ will decide to transmit the same value B with probability at least $1/n^{2c_0}$ (given a node sent and received the same information). Now, assume we create a graph by iteratively removing two randomly chosen nodes $u, v \in S$ and adding an edge (u, v) between them until S is empty. For each node $v \in S$ must specify some probability to be joined the MIS. Assume the algorithm sets at least $|S|/2$ nodes to join with probability at least $1/2$. Given that at most $|S|/4$ nodes have been chosen from S to form pairs, the probability that two nodes u, v out of the remaining nodes joining the MIS with probability $1/2$, i.e. $\geq |S|/2 - |S|/4 = |S|/4$ nodes, are paired up is at least $1/16$ independently of which nodes have been paired up before. The probability that for a pair u, v behaving identically both nodes u, v join (and thus the computation of the MIS fails) is at least $1/4$. Since we have $|S|/2 = n^{1-2c_0}/2$ pairs, we expect a set $S' \subset S$ of at least $n^{1-6c_0}/2 \cdot 1/16 \cdot 1/4$ pairs to behave identically and join the MIS. Using a Chernoff bound for any constant $c_0 < 1/6$ at least $|S'| \geq n^{1-6c_0}/1024$ nodes behave identically with probability at least $1 - 1/n^c$ for an arbitrary constant c.

An analogous argument holds if less $|S|/2$ nodes are joined with probability more than $1/2$. In this case for some pairs u, v w.h.p. no node will join the MIS.

□

Theorem 45. *Any algorithm computing a MIS or coloring deterministically (or in a randomized manner) transmitting only $b \leq c_1 \frac{\log n}{\log(t/\log n)}$ bits per edge with $t \in [2 \log n, n^{c_2}]$ for constants c_1, c_2 requires at least t time (w.h.p.). For $t < 2 \log n$ and $b \leq c_1 \log n$ bits no algorithm can compute a MIS (w.h.p.).*[2]

Proof. If $m \geq \sqrt{tb}$ using the bound of $\Theta(b)$ of Theorem 42, a node can communicate at most $c_{thm} c_1 \log n$ bits for a constant c_{thm}. We have $c_{thm} c_1 \log n \leq c_0 \log n$ for a suitable constant c_1. Due to Theorem 44 at least $c_0 \log n$ bits are needed. For $t < 2 \log n$ and $b \leq c_1 \log n$, we have $\sqrt{tb} \leq 2 c_1 \log n$. If $m < \sqrt{tb}$, then the amount of transmittable information

[2] Note that the theorem does not follow directly from Theorem 44, since the number of bits that can be communicated using time-coding is generally larger than b, i.e. see Theorem 42.

becomes (neglecting c_{thm} for now) $(m\log(tb/m^2)+b) \leq \sqrt{tb}\log 2 + c_1 \log n \leq 3c_1 \log n \leq c_0 \log n$ for a suitable constant c_1.

For $t \geq 2\log n$ and $m \leq b \leq \sqrt{tb}$ the amount of transmittable information becomes $O(m\log(tb/m^2)+b)$. We have $\max_{m\leq b} m\log(tb/m^2) \leq b\log(t/b)$. The maximum is attained for $m = b$. Using the assumption $b \leq c_1 \frac{\log n}{\log(t/\log n)}$, we get further: $b\log(t/b) \leq c_1 \frac{\log n}{\log(t/\log n)} \cdot \log(t\log(t/\log n)/\log n) = c_1 \frac{\log n}{\log(t/\log n)} \cdot (\log(t/\log n) + \log(\log(t/\log n))) = c_1 \log n \left(1 + \frac{\log(\log(t/\log n))}{\log(t/\log n)}\right) \leq 2c_1 \log n (\text{since } t \leq n)$. Thus, we have $m\log(tb/m^2)+b \leq 2c_1 \log n + b \leq 3c_1 \log n$.

Due to Theorem 44 at least $c_0 \log n$ bits required, thus for $3c_1 c_{thm} < c_0$ at least time t is required. The lower bound for the MIS also implies a lower bound for $O(\Delta)$ coloring, since a MIS for constant degree graphs can be computed from a coloring in constant time, i.e. in round i nodes with color i are joined the MIS, if no neighbor is already in the MIS. □

In a later section, we give an algorithm running in time $O(t\log n)$ using $O(\log n/\log t)$ messages and bits. Thus, there exists an algorithm running in $O(\log n)$ time transmitting only $\log n/c$ messages containing one bit for any constant c. On the other hand, due to our lower bound any algorithm that transmits only one message containing $\log n/c$ bits for a sufficiently large constant c requires time n^{1/c_1} for some constant c_1 and is thus exponentially slower.

6.5 Algorithms

We look at various deterministic and randomized algorithms for the coloring and the maximal independent set problem as case studies for trading among bit, message, and time Complexity. Before showing the tradeoffs we reduce the bit complexity of the algorithms without altering the time complexity. Then we show two mechanisms how prolonging an algorithm can be used to reduce the bit and – at the same time – the message complexity.

The first mechanism is straight forward and useful for randomized algorithms for symmetry breaking tasks, e.g. for MAC protocols where nodes try to acquire a certain resource. Assume a node tries to be distinct from its neighbors or unique among them. For example, for the coloring problem, it tries to choose a distinct color from its neighbors. For the MIS problem it tries to mark itself, and joins the MIS, if no neighbor is marked as well. Thus, if two neighbors get marked or pick the same color, we can call this a collision. We can reduce the probability of collisions by reducing the probability of a node to pick a color or get marked in a round. Thus, if a node

only transmits if it has chosen a color or got marked, this causes less bits to be transmitted.

The second mechanism is beneficial for certain distributed algorithms that solve problems by iteratively solving subproblems and combining the solutions. Often the size (or number) of subproblems determines the number of iterations required, e.g. for divide and conquer algorithms. Assume that a distributed algorithm requires the same amount of communication to solve a subproblem independent of the size of the subproblem. In this case, by enlarging the size of the subproblem, the total number of iterations and thus the total amount of information to be transmitted can be reduced.

Apart from that there are also general mechanisms that work for any algorithm. Encoding information using the traditional *time coding* approach for k rounds works as follows: To transmit a value x we transmit x div k in round $x \mod k$.[3] Thus, in case $k \geq 2x - 1$ a single message of one bit is sufficient. Otherwise $\log x - \log k$ bits are needed. Our lower bound (Theorem 42) shows that a value of $\log x$ bits can be communicated by transmitting less than $\log x$ bits using more than one message and more than one communication round.

6.5.1 Randomized $O(\Delta)$ Coloring in Time t^c using $O(\log n / \log t)$ Bits

One could use the previously described algorithm and traditional time coding to save on the bit complexity maintaining the same number of transmitted messages. For readability and to illustrate both concepts quantitatively we focus on the case $t^c \geq \log^{2+\epsilon} n$ (for an arbitrary small constant ϵ), where one can save on both: the message complexity by a factor of $\log t$ and the bit complexity by a factor of $\log \log n \log t$.[4] A node initially chooses an interval consisting of $(1 + 1/2^{\log^* n - 2}) \log^{1+1/\log^* n} n$ colors. Then the node iteratively transmits a single bit in a random round out of every $t_p = t^c / (c_1 \log n^{1+1/\log^* n})$ rounds. If it is the only one transmitting, it chooses a color, informs its neighbors about the obtained color and ends the algorithm.

Theorem 46. *The algorithm computes an $O(\Delta + \log^{1+1/\log^* n} n)$ coloring with bit complexity $O(\log n / \log t)$ in time $t^c + O(\log^* n)$ for any parameter c and t such that $t^c \geq \log^{2+\epsilon} n$ and $t \leq n$ for an arbitrary constant $\epsilon > 0$ w.h.p.*

[3] The division operation x div k returns an integer value that states how often number k is contained in x.

[4] For small $t \leq 2 \log n$ it is not possible to achieve bit complexity $c_1 \log n / \log t$ for a fixed constant c_1 due to the lower bound given in Theorem 45.

Algorithm FewBitsDeltaColoring, i.e. $(1+\epsilon)\Delta$ for $\epsilon > 1/2^{\log^* n - 2}$ and parameter $t > \log^{2+\epsilon} n$

1: $s(v) := none$; $ind_{I(v)} := none$; $C(v) := \{0, 1, ..., (1+\epsilon)\log^{1+1/\log^* n} n - 1\}$
2: $I(v) :=$ random integer $r \in [0, (1+\epsilon)\Delta_{N_+(v)}/\log^{1+1/\log^* n} n + 1]$
3: Transmit $I(v)$ to all neighbors $u \in N(v)$ using time $t^c/2$ and $\log n/\log t$ bits and messages
4: $N_{I(v)}(v) := \{u \in N(v) | I(v) = I(u)\}$ {Only consider nodes in the same interval}
5: $i := 0$; $t_p := t^c/(c_1 \log^{1+1/\log^* n} n)$ {with constant c_1}
6: **repeat**
7: **if** $i \mod t_p = 0$ **then** $t_s(v) :=$ Random odd number in $[0, t_p]$ **end if**
8: **if** $t_s(v) = i$ **then**
9: Transmit 1
10: **if** nothing received **then**
11: $ind_{I(v)} :=$ arbitrary available color
12: Transmit $ind_{I(v)}$
13: **end if**
14: **end if**
15: $N(v) := \{u | u \in N_{I(v)}(v) \wedge color_I(u) = none\}$
16: $C(v) := C(v) \setminus \{ind_I(u) | u \in N(v)\}$
17: $i := i + 1$
18: **until** $ind_{I(v)} \neq none$
19: $color(v) := ind_{I(v)} + I(v) \cdot (1+\epsilon)\log^{1+1/\log^* n} n$

Proof. The initial transmission of the interval requires less than $\log n$ bits, i.e. $\log\Delta - \log\log n$. We can use Theorem 42 with $b = m = O(\log n/\log t)$ messages m and bits b and at least $t^c/2 \geq (\log^{2+\epsilon} n)/2$ rounds. Since $\sqrt{bt} > m$ the amount of information that can be communicated is $\Theta(m\log(tb/m^2) + b) = O(\log n/\log t \log(t/(\log n/\log t)) + \log n/\log t) \geq O(\log n/\log t \log(t\log t/\log n)) \geq O(\log n/\log t(\log t + \log(\log t/\log n))) = O(\log n + \log n(\log\log t/\log t - \log\log n/\log t) = O(\log n)$ bits, since $\log\log n/\log t^c < 1/2$ because $t^c \geq \log^{2+\epsilon} n$.

Due to Lemma 17 each node v has at most $\Delta_0 := \log^{1+1/\log^* n} n$ neighbors competing for the $(1 + 1/2^{\log^* n - 2})\log^{1+1/\log^* n} n$ colors of v's chosen interval. A node v transmits one bit for each interval of length t_p. Since nodes make their choices independently, the probability that node v is the only node transmitting is at least $1 - \Delta_0/t_p$, corresponding to the worst case that all neighbors transmit in different rounds. We have $\Delta_0/t_p = \Delta_0/(t^c/(c_2\log^{1+1/\log^* n} n) = \Delta_0 \cdot c_2 \log^{1+1/\log^* n} n/t^c \leq$

$c_2 \log^{2+2/\log^* n} n/t^c$(due to Lemma 17) $\leq 1/t^{c \cdot 1/c_3}$ for some constant c_3 since $t^c \geq \log^{2+\epsilon} n$.Thus the chance $O(\log n/\log t) = c_4 \log n/\log t$ trials fail is $(1/t^{c/c_3})^{c_4 \log n/\log t} = 1/n^{c_1}$ for an arbitrary constant c_1 and a suitable constant c_4. □

6.5.2 Deterministic $\Delta + 1$ Coloring

We adapt an algorithm [80] to turn a Δ^k coloring for any constant k into a $\Delta + 1$ coloring in time $O(t^c \Delta \log \Delta)$ using $O(\log \Delta/\log t)$ messages of size $O(\log \Delta)$ and for an arbitrary parameter t and arbitrary constant c. The algorithm reduces the number of used colors in an iterative manner by splitting up the whole range of colors into sequences of colors of size at least $2t^c(\Delta + 1)$. Consider all nodes S_I that have a color in some sequence I consisting of $2t^c(\Delta+1)$ colors, e.g. $I = \{0, 1, ..., 2t^c(\Delta+1)-1\}$. To compress the range of used colors to $[0, \Delta]$, we can sequentially go through all $2t^c(\Delta + 1)$ colors and let node $v \in S_I$ choose the smallest available color, i.e. in round i a node having the ith color in the interval I can pick a new color from I not taken by any of its neighbors. After going through all colors, we combine $2t^c$ intervals $\{I_0, I_1, ..., I_{2t^c-1}\}$ to get a new interval I' of the same size, i.e. $2t^c(\Delta + 1) - 1$. A node v with color i from I_j with $i \in [0, \Delta]$ gets color $c(v) = j \cdot (\Delta + 1) + i$ in I'. Then we (recursively) apply the procedure again on all intervals I'.

Theorem 47. *The deterministic $\Delta + 1$ coloring terminates in time $O(t^c \Delta \log \Delta)$ having bit complexity $O(\log^2 \Delta/\log t)$ and message complexity $O(\log \Delta/\log t)$ for any parameter $1 < t \leq \Delta$ and any constant c.*

Proof. We start from a correct Δ^k coloring for some constant k. A node gets to pick a color out of a sequence $(c_1, c_1 + 1, ..., c_1 + 2t^c(\Delta + 1) - 1)$ of $2t^c(\Delta + 1)$ colors for $c_1 := c_0 2t^c(\Delta + 1)$ and an arbitrary integer c_0. Thus it can always pick a color being at most $c_1 + \Delta$ since it has at most Δ neighbors. After every combination of intervals requiring time $2t^c(\Delta+1)$, the number of colors is reduced by a factor of $2t^c$. We require at most $x = k \log \Delta/\log(2t^c)$ combinations since $(2t^c)^x = \Delta^k$. Therefore, the overall time complexity is $O(t^c \Delta \log \Delta)$. In each iteration a node has to transmit one color out of $2t^c(\Delta + 1)$ many, i.e. a message of $\log(2t^c(\Delta + 1)) = O(\log \Delta)$ bits (since $t^c \leq \Delta$) giving $O(\log^2 \Delta/\log t)$ bit complexity. □

6.5.3 MIS Algorithm

Our randomized Algorithm LowBitAndFast is a variant of algorithm [90]. It proceeds in an iterative manner. A node v marks itself with probability $1/d(v)$. In case two or more neighboring nodes are marked, the choice which

of them is joined the MIS is based on their degrees, i.e. nodes with higher degree get higher priority. Since degrees change over time due to nodes becoming adjacent to nodes in the MIS, the degree has to be retransmitted whenever there is a conflict. Our algorithm improves Luby's algorithm by using the fact that the degree $d(u)$ of a neighboring node is not needed precisely, but an approximation $\tilde{d}(u)$ is sufficient. Originally, each node maintains a power of two approximation of the degrees of its neighbors, i.e. the approximation is simply the index of the highest order bit equal to 1. For example, for $d(v)$ having binary value 10110, it is 4. The initial approximate degree consists of $\log\log n$ bits. It is transmitted using (well known) time coding for $x = \log n$ rounds, i.e. to transmit a value k we transmit k div x in round $k \mod x$. When increasing the time complexity by a factor of t^{c_0} for an arbitrary constant c_0, a node marks itself with probability $1/(t^{c_0}\tilde{d}(v))$ for t^{c_0} rounds, where the approximation is only updated after the t^{c_0} rounds. Afterwards, a node only informs its neighbors if the degree changed by a factor of at least two. For updating the approximation we use time message coding for t^{c_0} rounds and a constant number of messages and bits. Whenever a node is joined the MIS or has a neighbor that is joined, it ends the algorithm and informs its neighbors.

The analysis is analogous to [90] and differs only in the constants.

Lemma 48. *A node v maintains an approximation $\tilde{d}(u)$ of the degrees $d(u)$ of all neighbors $u \in N(v)$, such that $d(u) \leq \tilde{d}(u) \leq 2 \cdot d(u)$, when node v begins to join the MIS for t^{c_0} rounds (i.e. enters the for loop).*

Proof. Since initially a node transmits a two approximation of its degree to all neighbors, the lemma holds at the beginning of the algorithm, i.e. before the first execution of the for loop (lines 4 to 7). Whenever for a node v, the approximation $\tilde{d}(v)$ stored by its neighbors (and itself) is larger than $i \cdot d(v)$ with integer $i > 2$ before an execution of the for loop (line 4 to 7), node v adapts $\tilde{d}(v)$, i.e. sets $\tilde{d}(v) := \tilde{d}(v)$ div $2^{\lceil \log i \rceil}$, such that $d(v) \leq \tilde{d}(v) \leq 2 \cdot d(v)$ and transmits i to its neighbors $u \in N(v)$ that also adapt their approximation $\tilde{d}(v)$ for node v's degree. □

Lemma 49 (Joining MIS)**.** *A node v is joined the MIS in t^{c_0} rounds with probability $p \geq \frac{1}{8t^{c_0} d_0(v)}$, where $d_0(v)$ denotes the degree of node v before the execution of the first out of t^{c_0} rounds.*

Proof: Let M be the set of marked nodes, i.e. with their bits equal to 1. Let $H(v)$ be the set of neighbors of v with higher or equal degree approximation. Using independence of v and $H(v)$ we get

Algorithm LowBitAndFast for arbitrary value $t^{c_0} \geq 16$

For each node $v \in V$:

1: $\tilde{d}(v) :=$ index of highest order bit of $2|N(v)|$ {2 approximation of $d(v)$}
2: Transmit $\tilde{d}(v)$ to all neighbors $u \in N(v)$ using time coding for $\log n$ rounds
3: **loop**
4: **for** $i = 1..t^{c_0}$ **do**
5: Choose a random bit $b(v)$, such that $b(v) = 1$ with probability $\frac{1}{4t^{c_0} \cdot \tilde{d}(v)}$
6: Transmit $b(v)$ to all nodes $u \in N(v)$
if $b(v) = 1 \wedge \nexists u \in N(v), b(u) = 1 \wedge \tilde{d}(u) \geq \tilde{d}(v)$ **then** Join MIS
end if
7: **end for**
8: $k(v) := \max\{\lceil \log i \rceil | \text{integer } i, \frac{\tilde{d}(v)}{i} \geq d(v)\}$
9: **if** $k(v) > c_0/2 \log t$ **then**
10: Transmit $k(v)$ using time message coding for t^{c_0} rounds using c_2 messages of size 1 bit
11: $\tilde{d}(v) := \tilde{d}(v) \text{ div } 2^{k(v)} + \tilde{d}(v) \mod 2^{k(v)}$
12: **end if**
13: **for all** received messages $k(u)$ **do**
14: $\tilde{d}(u) := \tilde{d}(u) \text{ div } 2^{k(u)} + \tilde{d}(u) \mod 2^{k(u)}$
15: **end for**
16: **end loop**

$$
\begin{aligned}
Pr\left[v \notin \text{MIS} | v \in M\right] &= Pr\left[\exists w \in H(v), w \in M | v \in M\right] \\
&= Pr\left[\exists w \in H(v), w \in M\right] \\
&\leq \sum_{w \in H(v)} Pr\left[w \in M\right] = \sum_{w \in H(v)} \frac{1}{4t^{c_0} \tilde{d}(w)} \\
&\leq \frac{d_0(v)}{4\tilde{d}(v) t^{c_0}} \leq \frac{1}{2t^{c_0}}
\end{aligned}
$$

- The first equality is correct because $H(v)$ and v are independently marking themselves.
- The second inequality is because $w \in H(v)$, if $\tilde{d}(v) \leq d(\tilde{w})$
- The third inequality is because $w \in H(v)$, if $d_0(v) \leq 2d(\tilde{v})$

Then

$$Pr\left[v \in \text{MIS}\right] = Pr\left[v \in \text{MIS} | v \in M\right] \cdot Pr\left[v \in M\right] \geq (1 - \frac{1}{2t^{c_0}}) \cdot \frac{1}{4t^{c_0}\tilde{d}(v)} \geq \frac{1}{8t^{c_0}d_0(v)}$$

□

Lemma 50 (Good Nodes)**.** *A node v is called* good *if*

$$\sum_{w \in N(v)} \frac{1}{4d(w)} \geq \frac{1}{12}.$$

Otherwise we call v a bad *node. A good node will be removed with probability $p \geq c_1$ for some constant $c_1 > 0$ in t^{c_0} rounds.*

Proof: Let node v be good. If there is a neighbor $w \in N(v)$ with degree at most 8 we are done: With Lemma 49 the probability that node w is joined the MIS is at least $\frac{1}{64t^{c_0}}$ for a single round and thus $1 - (1 - \frac{1}{64t^{c_0}})^{t^{c_0}} \geq c_1$ for t^{c_0} rounds, and our good node will be removed.

So all we are worried about is that all neighbors have at least degree 9: For any neighbor w of v we have $\frac{1}{4d(w)} \leq \frac{1}{12}$. Since $\sum_{w \in N(v)} \frac{1}{4d(w)} \geq \frac{1}{12}$ there is a subset of neighbors $S \subseteq N(v)$ such that $\frac{1}{12} \leq \sum_{w \in S} \frac{1}{4d(w)} \leq \frac{1}{3}$

We can now bound the probability that node v will be removed. Let therefore R be the event of v being removed. Again, if a neighbor of v is joined the MIS, node v will be removed. We have

$$\begin{aligned} Pr\left[R\right] &\geq Pr\left[\exists u \in S, u \in \text{MIS}\right] \\ &\geq \sum_{u \in S} Pr\left[u \in \text{MIS}\right] - \sum_{u,w \in S; u \neq w} Pr\left[u \in \text{MIS } \textbf{and } w \in \text{MIS}\right]. \end{aligned}$$

For the last inequality we used the inclusion-exclusion principle truncated after the second order terms. Let M again be the set of marked nodes after step 1. Using $Pr\left[u \in M\right] \geq Pr\left[u \in \text{MIS}\right]$ we get

$$\begin{aligned} Pr[R] &\geq \sum_{u \in S} Pr[u \in \text{MIS}] - \sum_{u,w \in S; u \neq w} Pr[u \in M \textbf{ and } w \in M] \\ &\geq \sum_{u \in S} Pr[u \in \text{MIS}] - \sum_{u \in S} \sum_{w \in S} Pr[u \in M] \cdot Pr[w \in M] \\ &\geq \sum_{u \in S} \frac{1}{8t^{c_0} d(u)} - \sum_{u \in S} \sum_{w \in S} \frac{1}{4t^{c_0} d(u)} \frac{1}{4t^{c_0} d(w)} \\ &\geq \sum_{u \in S} \frac{1}{4t^{c_0} d(u)} \left(\frac{1}{2} - \sum_{w \in S} \frac{1}{2t^{c_0} d(w)} \right) \geq \frac{1}{48t^{c_0}} \left(\frac{1}{2} - \frac{1}{6t^{c_0}} \right) = \frac{1}{c_2 t^{} \end{aligned}$$

(for some constant c_2).

Thus, the probability that a node is not removed for t^{c_0} rounds becomes $(1 - \frac{1}{c_2 t^{c_0}})^{t^{c_0}} \leq c_3$ for some constant $c_3 < 1$. Therefore, the probability p that a good node is removed is a constant $c_1 := 1 - c_3 > 0$.

Lemma 51 (Good Edges)**.** *An edge $e = (u, v)$ is called* bad *if both u and v are bad; else the edge is called* good*. The following holds: At any time at least half of the edges are good.*

Proof: For the proof we construct a directed auxiliary graph: Direct each edge towards the higher degree node (if both nodes have the same degree direct it towards the higher identifier). Now we need a little helper lemma before we can continue with the proof. The indegree is defined as the number of edges pointed toward a node. The outdegree is defined as the number of edges incident to a node that point away from it.

Lemma 52. *A bad node has outdegree at least twice its indegree.*

Proof: For the sake of contradiction, assume that a bad node v does not have outdegree at least twice its indegree. In other words, at least one third of the neighbor nodes (let us call them S) have degree at most $d(v)$. But then

$$\sum_{w \in N(v)} \frac{1}{4d(w)} \geq \sum_{w \in S} \frac{1}{4d(w)} \geq \sum_{w \in S} \frac{1}{4d(v)} \geq \frac{d(v)}{3} \frac{1}{4d(v)} = \frac{1}{12}$$

which means v is good, a contradiction. □

Continuing the proof of Lemma 51: According to Lemma 52 the number of edges directed into bad nodes is at most half the number of edges directed out of bad nodes. Thus, the number of edges directed into bad nodes is at most half the number of edges. Thus, at least half of the edges are directed into good nodes. Since these edges are not bad, they must be good.

Theorem 53. *Algorithm* LowBitAndFast *terminates in time* $O(t^{c_0} \log n)$ *w.h.p. having bit and message complexity* $O(\log n / \log t)$.

Proof: With Lemma 50 a good node (and therefore a good edge) will be deleted with constant probability within t^{c_0} rounds. Since at least half of the edges are good (Lemma 51) a constant number of edges will be deleted in each phase.

More formally: With Lemmas 50 and 51 we know that an edge will be removed with probability at least $c_1/2$. Let R be the number of edges to be removed. Using linearity of expectation we know that $E[R] \geq m \cdot (c_1/2)$, m being the total number of edges at the start of a phase. Now let $p := Pr[R \leq E[R]/2]$. Bounding the expectation yields

$$E[R] = \sum_r Pr[R = r] \cdot r \leq p \cdot E[R]/2 + (1-p) \cdot m.$$

Solving for p we get

$$p \leq \frac{m - E[R]}{m - E[R]/2} < \frac{m - E[R]/2}{m} \leq 1 - c_1/2.$$

In other words, with probability at least $c_1/4$ for at least $m \cdot (c_1/4)$ edges are removed in a phase. We call a phase successful if this is the case. Since phases are independent, we can bound the probability that half of $c_2 \cdot \log n$ phases are successful using a Chernoff bound by $1 - e^{c_2/8 \log n}$. Thus, with probability $1 - 1/n^{c_2/8}$ there are at least $c_2/2 \log n$ successful phases. Since the number of edges $m \leq n^2$, the theorem follows for a sufficiently large constant c_2. □

Initially a node transmits its two approximation, which corresponds to an index in the binary representation of its degree. The degree can be encoded using $\log n$ bits and thus the two approximation requires at most $\log \log n$ bits.

Using time coding for $\log n$ rounds these bits are transmitted using a message of 1 bit. From then on the degree is only updated if a node's approximation differs from its correct value by more than a factor of 2 after t^{c_0} rounds. To perform an update, a node v transmits a value $k(v) = \log \max\{i | \text{integer } i, \frac{\tilde{d}(v)}{i} \geq |N(v)|\}$ using time coding for t^{c_0} rounds. Say a node transmits $r \leq O(\log n / \log t)$ factors $k_0(v), k_1(v), ..., k_{r-1}(v)$. We must have that $\prod_{i=0}^{r-1} 2^{k_i(v)} \leq d(v) \leq n$ or equivalently $\log \prod_{i=0}^{r-1} 2^{k_i(v)} \leq \log n = \sum_{i=0}^{r-1} k_i(v) \leq \log n$. Thus, the maximal amount of bits that has to be transmitted within messages when using time coding for t^{c_0} rounds for each message is given by $\max_{r, k_i(v)} \sum_{i=0}^{r-1} (\max \log k_i(v) - c_0 \log t, 1)$ given $\sum_{i=0}^{r-1} k_i(v) \leq \log n$. The maximum is $O(\log n / \log t)$ for $k_i(v) = 1$ and $r = O(\log n / \log t)$ yielding an overall bit complexity of $O(\log n / \log t)$.

Chapter 7

Conclusions

We have looked at different techniques for symmetry breaking. We proposed a new technique called *Multi-Trials* and applied it to the coloring and ruling set problems. For coloring, it improves on previous work dramatically in most cases. Only if the maximum degree Δ of a graph is small, deterministic techniques are preferable (from a worst case perspective). Simulations also show good average case performance. We have presented a novel deterministic coin tossing technique which enables us to achieve an asymptotically optimal distributed algorithm for computing a MIS in bounded-independence graphs. Although the hidden constants in the analysis are significant, our simulations indicate that the constants will be small in practice. Our algorithm has been applied in various settings beyond the MIS problem, for instance for computing a Δ+1 coloring, an asymptotically optimal CDS in $O(\log^* n)$ or for solving the facility location problem [98].

We believe that our work strikes a chord with symmetry breaking. Randomized symmetry breaking has the problem of only producing results "in expectation". But often symmetry breaking algorithms are used as building blocks, and then they need to work "with high probability" which regularly causes a logarithmic overhead. In other words, when it comes to "ultra-fast" distributed algorithms, determinism may have an advantage over randomization.

Though it is hard in a distributed setting – and sometimes not even possible – to use less than $\Delta + 1$ colors, we feel that one should also keep an eye on the original definition of the coloring problem in a distributed environment: Color a graph with as little colors as possible. To strive for a $\Delta+1$ coloring is of much appeal and gives interesting insights but as we have shown (in many cases) better bounds regarding the number of used colors and the required time complexity can be achieved by taking the chromatic

number of the graph into account.

We also discussed different mechanisms to trade among complexity measures of distributed algorithms and illustrated them by applying them to various symmetry breaking algorithms. Our belief is that these trade-offs are very important, since different application domains have different requirements, i.e., have different views on the importance of a complexity measure.

Finally, as promised in Section 2.1, let us discuss the communication model in more detail. We presented our algorithm in the classic local model, where each node can talk to each neighbor in each round. In some application domains, in particular in wireless networks, this assumption is too demanding, as it asks for a perfect (and hence non-existent!) media access control (MAC) protocol. In reality MAC protocols are quite unreliable, with messages not being received because of wireless channel fluctuations, or message collisions. One way to address this is to study the problem in a model that includes the MAC layer, in the sense that the algorithm designer has to exactly specify at what time the nodes transmit or receive. This is a cumbersome job, especially since the result is often related to the algorithm in the clean local model, e.g., the without collision detection model [108] uses the local algorithm presented in this work as a building block.

What about situations where the software engineer has no control over the MAC layer? How should our algorithm be implemented? We argue that one should simply simulate our algorithm in the "rollback compiler" self-stabilization framework [10]. Note that our algorithm does not use all the flexibility provided by the local model, instead the result messages r_v^i can be broadcast in the neighborhood. In the self-stabilization framework our protocol then boils down to repeatedly transmitting the same single message, containing information about the relevant result values computed in the individual phases/competitions. This message is of logarithmic size, and resilient to a whole array of failures and dynamics, e.g., message collisions, message failures, even nodes rebooting and topology changes due to mobility or late node wake-ups. For our MIS algorithm nodes will always have a correct MIS at most $O(t \log^* n)$ time after the last failure, assuming that each node can successfully transmit a message at least once in time t!

Part II

Wireless Networks

Chapter 8

Introduction

Wireless networks have become omnipresent; there are different architectures available, e.g. GSM-driven mobile phones, or laptops equipped with 802.11 WLAN. A challenging network architecture are so-called wireless multi-hop networks, e.g., ad hoc, sensor, or mesh networks. In these networks, the participating nodes themselves provide the infrastructure. This lack of already available infrastructure expresses itself primarily during the start-up phase of the wireless multi-hop network, as nodes need to establish their own infrastructure before being able to communicate. For instance, upon deployment the nodes act independently, as they are unaware of each other. In this stage communication is unreliable and slow. A major problem is that messages might be lost due to collisions, i.e. if nodes u and w concurrently transmit a message, a node v being in the transmission range of both u and w might not receive any message correctly, or v might not even detect that there was a transmission. Coordinating the nodes in a distributed manner to achieve efficient and reliable communication is a time consuming task – in particular in multi-hop networks, where the exchange of messages among nodes requires the help of intermediate nodes.

Once a wireless network is fully operational, it runs a reliable Media Access Control (MAC) scheme, for instance, Time Division Multiple Access (TDMA). In TDMA, time is slotted, and individual time slots are assigned to different nodes, such that potentially interfering nodes do not share a common time slot. Algorithmically speaking this boils down to a coloring problem: Assign colors to nodes such that "close-by" nodes do not share the same color. Depending on the application the term "close-by" may have a different meaning: direct neighbors, or also two-hop neighbors (since two-hop neighbors u and w share a common neighbor v that will be affected by a concurrent transmission of u and w), or more generally k-hop neighbors

for some constant k. Strictly speaking even that is a simplification because physical radio signals do not comply to graph-based models; however, in practice graph-based models are used, and do usually work alright. We will discuss the modeling issues in the conclusions. To achieve a high throughput in the wireless network, we must aim for a coloring that minimizes the number of colors. As the number of colors makes a significant impact in practice, constants do matter.

When studying distributed algorithms for wireless networks, there are two extreme models. The popular *(unstructured) radio network model* buys into worst-case thinking: Nodes wake-up asynchronously in a worst case manner and concurrent transmissions cancel each other because of interference, usually to a degree such that a potential receiver cannot even sense that there has been a message collision. On the other hand, the *local model* is used to abstract away from media access issues, allowing the nodes to concurrently communicate with all neighbors and generally assumes synchronized wake-up of all nodes.

Clearly, the local model is too optimistic. The radio network model, however, often is rather pessimistic. Most wireless devices can distinguish at least four states: (i) either the wireless node is transmitting itself and is therefore not capable of noticing any other communication, or it is silently listening, usually allowing it to differentiate between the other three states: (ii) the media is free because nobody is transmitting, (iii) at least one node is transmitting and the message can be decoded, and (iv) more than one node is transmitting but no message can be decoded. In the last case the listening node can sense that there are transmissions happening, e.g. there is energy on the channel in a wireless network. This model is called the *collision detection model*.

In Chapter 9 we study different problems from a theoretical point of view, such as the coloring and broadcasting problems in the network model and the collision detection model by deriving worst-case upper and lower bounds in both models.

In the practical part (Chapter 10), we derive an ultra-low power MAC protocol that was implemented and tested on actual hardware. We also investigated the reception properties of a listening node if several nodes in its neighborhood transmit concurrently through experiments. This not only deepened our understanding of wireless networks but also gave rise to our belief that the collision detection model is generally underrated within the wireless community dealing with theory mainly. These studies also led to a localization scheme for wireless nodes.

Chapter 9

Theory

9.1 Introduction

We derive several results for different problems to relate the radio network and the collision detection models. We make the same assumptions about the graph, wake-up, topology etc. in both models. In particular, we assume that an estimate of n is known. Without an estimate of n a transmission takes $\Omega(\frac{n}{\log n})$ in the radio network model, yielding a clear advantage for algorithms employing collision detection. For an overview of lower and upper bounds see Table 9.1. All in all, an advantage of collision detection is that it allows to design fast deterministic algorithms giving reliable bounds on the time complexity. For example, our MIS algorithm is asymptotically optimal, and also considerably faster (i.e. a factor of $\log n / \log\log n$) than the best possible MIS algorithm for the radio network model. For broadcasting, our deterministic algorithm can be exponentially faster than the best deterministic counterpart in the radio network model. For coloring, the current lower and upper bound show that there cannot be an asymptotic gain for randomized algorithms for graphs of maximal degree $\Delta \in \Omega(\log^2 n)$.

Furthermore, many algorithms for wireless networks are designed for general graphs. This model does not capture the nature of (somewhat) circular transmission ranges of wireless devices. Therefore, within the wireless computing community the so-called *Unit Disk Graph (UDG)* and variations of it, e.g. the *Quasi Unit Disk Graph*, have been widely adopted. In the UDG two nodes are adjacent if their distance is at most 1. We use a generalized model of these geometric graphs, i.e. *Growth-Bounded Graphs (GBG)*, which restrict the size of an independent set in the neighborhood of a node.

Interestingly, we show that the lower bound for general graphs without collision detection for deterministic broadcasting can be adapted to GBG

without any asymptotic change. The lower bound for randomized algorithms can be adapted as well yielding no asymptotic change already for graphs of polylogarithmic diameter (in the number of nodes n). Thus, the choice of the GBG model does not seem to render the problem more simple.

On the road to our results we also encounter a special type of broadcasting problem, which we call the *distance-d broadcasting problem*: A message should be delivered from an originator to all nodes within some (hop) distance d but (ideally) to no node further away. We present a solution that is time-optimal in the radio networks model for constant distances such that only nodes at distance $d + 1$ might receive the message. The current state of the art coloring algorithm for radio networks by Moscibroda et al. [93] needs with high probability $O(\Delta \cdot \log n)$ time and uses $O(\Delta)$ colors in a unit disk graph, where n and Δ are the number of nodes in the network and the maximum degree, respectively; the algorithm requires a linear bound on n and Δ. Similarly to the journal version of [93] we generalize the network topology from unit disk graph to bounded-independence graph (sometimes also called growth-bounded graphs[1]) [74]. Bounded-independence graphs restrict for any node the maximum size of a set of independent nodes in its neighborhood. In a unit disk graph nodes are points in the plane, with two nodes u, v being neighbors if and only if their Euclidean distance is at most 1. We improve the result of [93] in three ways:

1. We reduce the time complexity, instead of the logarithmic *factor* we just need a polylogarithmic *additive term*; more specifically, our time complexity is $O(\Delta + \log \Delta \cdot \log n)$ given an estimate of n and Δ, and $O(\Delta + \log^2 n)$ without any knowledge of Δ.

2. For a simple (one-hop) coloring, our algorithm needs $\Delta + 1$ colors only. As in prior work, the number of actual colors used depends only on the local density of the nodes.

3. Our algorithm can be adapted to compute a distance-d coloring with $O(\Delta)$ colors for any constant d.

After giving the necessary model assumptions and discussing related work, we give a coloring algorithm (also usable for computing TDMA schedules) in the radio networks model in Section 9.4. Several problems, i.e. broadcasting, coloring and MIS, for the collision detection model are elaborated on in Section 9.5.

[1] In some publications growth-bounded refers to the growth of the number of neighbors and in other publications it denotes the growth of the number of (maximum) possible independent nodes.

9.2 Model and Definitions

Communication among nodes is done in synchronized rounds. In every round a node v can either listen or transmit. A listening node v can successfully receive a message in round i, if exactly one neighbor $u \in N(v)$ was transmitting in round i. If collision detection is available, we say a node v *detected transmission (dT)* in round i, if the node was listening in round i, if it has at least one transmitter in its neighborhood $N(v)$.

We assume that n is known and all nodes have unique IDs from the interval $[1, n]$ using the same number of bits, i.e. small IDs have a prefix with 0s such that all IDs have equal length.[2] All our algorithms are shown to work in case of asynchronous wake-up, i.e. each node wakes-up at an unknown point in time. Only after its wake-up it is able to follow ongoing communication. The time complexity of an algorithm denotes the number of rounds until a solution has been computed for all nodes, i.e. it denotes the time from the wake-up of the last node until all nodes have computed a solution. For broadcasting we focus on conditional (or non-spontaneous) wake-up, where nodes wake-up and can perform computations (and transmissions) only after they detected transmission for the first time.

A set S is a maximal independent set (MIS), if any two nodes $u, v \in S$ have hop distance at least 2 and every node $v \in V \setminus S$ is adjacent to a node $u \in S$. A MIS S of maximum cardinality is called a maximum independent set. We model the communication network using undirected growth-bounded (also known as bounded-independence) graphs(GBG):

Definition 5. *A graph $G = (V, E)$ is growth-bounded if there is a polynomial bounding function $f(r)$ such that for each node $v \in V$, the size of a MaxIS in the neighborhood $N^r(v)$ is at most $f(r)$, $\forall r \geq 0$.*

In particular, this means that for a constant c the value $f(c)$ is also a constant. A subclass of GBGs are (Quasi)UDGs, which have $f(r) \in \mathrm{O}(r^2)$.

We denote by $\log^{(j)} n$ the binary logarithm taken j times recursively. Thus $\log^{(1)} n = \log n$, $\log^{(2)} n = \log \log n$, etc. To improve readability we assume that $\log^{(j)} n$ is an integer for any j. The term $\log^* n$ denotes how often one has to take the logarithm to get down to 1, i.e. $\log^{(\log^* n)} n \leq 1$.

9.3 Related Work

A lot of work work deals with unstructured radio networks (interfering transmissions, asynchronous wake-up) [75]. In this harsh model algorithms without at least a little topology information, i.e. the number of nodes n, are

[2] A polynomial bound n^c of the number of nodes n and IDs chosen from the range $[1, n^c]$, would yield the same asymptotic run time for all our algorithms.

Upper and Lower Bounds		
Problem	With Collision Detection	Without
MIS	$O(\log n)$ det. [This work]	$O(\log^2 n)$ ra. [94]
	$\Omega(\log n)$ [This work]	$\Omega(\log^2 n / \log\log n)$[64]
$\Delta + 1$ Col.	$O(\Delta + \log^2 n)$ ra. [108]	$O(\Delta + \log^2 n)$ ra. [108]
	$\Omega(\Delta + \log n)$ [This work]	$\Omega(\Delta + \log n)$ [This work]
Broadcast	$O(D \log n)$ det. [This work]	$O(n \log n)$ [70] det.
	$\Omega(D + \log n)$ [This work]	$\Omega(n \log_{n/D} n)$ det. [70][This work]

Table 9.1: Comparison of deterministic (det.) and randomized (ra.) algorithms with/without collision detection for various problems in GBG

inherently slow [65]. Efficient deterministic algorithms are often hard or impossible to design. For instance, for the broadcasting problem there is an exponential gap in the time complexity between any deterministic and randomized algorithm [70]. Still, an algorithm achieving an $O(\Delta)$-coloring in time $O(\Delta \cdot \log n)$ with probability $1 - \frac{1}{n}$ for unit disk graphs was presented by Moscibroda et al. [93]. This algorithm improved on [99] which ran in time $O(\Delta \log^2 n)$. In that algorithm knowledge about the network topology is limited to a bound on n and Δ. The algorithm computes a set of leaders which grant a color range for all other nodes (so called slaves). This general idea has also been followed previously, e.g. [59], however in a simpler model allowing for powerful communication.[3] Likewise, distributed coloring for TDMA has been studied several times, however, always in different settings (single-hop network, stronger communication models, etc.); for a detailed treatment we refer to the discussion of related work in [93].

There is also an ample body of work on asynchronous wake-up. In the so-called wake-up problem [65], the goal is to wake up all nodes in the graph as quickly as possible. The assumption is that a node is woken up by an incoming message. While the algorithmic problems resulting from this assumption are interesting, multi-hop wake-up radios are currently technically infeasible.

The MIS problem has been studied in many types of graphs using many different models, e.g. the UDG and its generalization the GBG [73] or geometric radio networks (GRN), e.g. [31]. In the weaker GRN model nodes are positioned in the plane and each node knows its coordinates by a GPS device or some other means (and sometimes also the coordinates of its neighbors or a bound on the distances). A node v is connected to all other nodes within some distance $dist(v)$. Often the distance is equal for all nodes, e.g.

[3] Specifically: Synchronous start, and each node can sense the presence of a signal. If the medium is free, it has a constant chance of transmitting without collision.

[31], and thus connectivity is the same (up to a scaling factor) as for UDG. In the message passing model, where all nodes can exchange messages at the same time (without collisions), an asymptotically optimal MIS algorithm was stated in [106] needing $O(\log^* n)$ communication rounds for GBG. We extend this algorithm in several ways. If collisions can occur, but may not be detected, in [94] a randomized algorithm taking time $O(\log^2 n)$ was given, which is optimal up to a factor of $O(\log \log n)$[64]. It even works for arbitrary wake-up, i.e. nodes do not share global time.

For the well-studied broadcasting problem under the assumption of unknown topology and conditional (also called non-spontaneous) wake-up, i.e. a node can perform any computation only after detecting some activity (e.g. receiving the message or detecting energy on the channel) in [12] a broadcasting scheme was proposed performing a broadcast in time $O(D \cdot \log n + \log^2 n)$. Essentially each node broadcasts $O(\log n)$ times with the same probability $\frac{1}{2^i}$ for $1 \leq i \leq \log n$, starting with probability $\frac{1}{2}$, heading towards $\frac{1}{n}$, and then restarting again with $\frac{1}{2}$. This broadcasting algorithm was improved to $\Theta(D \cdot \log \frac{n}{D} + \log^2 n)$ in [25] assuming collisions (but no detection) in general (undirected) graphs. This is optimal due to lower bounds in [2, 81]. The key idea is to choose high sending probabilities more often. In the deterministic case in [70] an algorithm is described requiring $O(n \cdot \log^2 D)$ steps, which is optimal up to factor of $O(\log D)$ [23]. We extend the lower bound for deterministic algorithms[70] as well as the $\Omega(D \cdot \log(n/D))$ bound for randomized algorithms [82] to GBG. The $\Omega(\log^2 n)$ lower bound[2] cannot be extended in the same manner as discussed in Section 9.5.2.

Both algorithms[25] rely on a global clock. Furthermore, these algorithms are not straightforward to adjust such that they solve the *distance-d* broadcasting problem, where a message should be broadcast to all nodes up to hop distance d but not further. For instance, just adding a time-to-live variable to a message does not solve the problem. Our broadcasting algorithm in Section 9.4.1 uses the same transmission probabilities as [12, 25], but is independent of a global clock.

In [22] it was shown how to broadcast a message of size $O(k)$ in time $O(k \cdot D)$ by using collision detection to forward a message bit by bit in arbitrary graphs. In the same paper the currently fastest deterministic algorithm for arbitrary message size also using collision detection was given taking time $O(n \cdot D)$. Thus for the crucial class of GBG in the area of wireless networks our algorithm is an exponential improvement for graphs of polylogarithmic diameter. In [61] broadcasting is discussed with and without collision detection using "advice", i.e. each node is given some number of bits containing arbitrary information about the network. It is shown that for graphs, where constant broadcasting time is possible, $O(n)$ bits of advice suffice to achieve optimal broadcasting time without collision detection, whereas $o(n)$ bits are

not enough (even with collision detection at hand). In case of GRN only a constant number of bits is sufficient.

In [31] every node can detect collisions and knows its position within a GRN. This allows to assign nodes into grid cells, which is the key to achieve asymptotically optimal broadcasting time of $\Theta(D + \log n)$.

In [71] an $O(n)$ time deterministic algorithm for the problem of leader election with collision detection for arbitrary networks was given. In [117] a randomized leader election protocol is given for single-hop networks running in expected time $O(\log \log n)$. In [27] deterministic algorithms for consensus and leader election were studied for single-hop networks, i.e. the underlying graph forms a clique. With collision detection both tasks can be performed in $\Theta(\log n)$, whereas without collision detection time $\Omega(n)$ is required. As in this work (see Algorithm *Asynchronous MIS*), round coding was used to synchronize rounds. For single-hop networks time $\Omega(k(\log n)/\log k)$ [48] is needed by any deterministic algorithm until k stations out of n transmit using collision detection.

9.4 Coloring Radio Networks

9.4.1 Algorithm Broadcast

Algorithm Broadcast is an essential part of the coloring algorithm given in the next section. It performs a broadcast of the content msg_w created by node w, which is delivered to all nodes $u \in N^h(w)$ up to distance h with probability at least $1 - \epsilon$. The message might also be received by (some) nodes $u \in N^{h+1}(w) \setminus N^h(w)$ at distance $h + 1$, but not by nodes at distance more than $h + 1$. The algorithm still works correctly if up to $5 \cdot f(h)$ (see Definition 5) nodes $u \in N^h(w)$ perform a broadcast concurrently.

Nodes forward a message in a synchronized manner, meaning that all nodes having received a message by w, transmit (sufficiently often) with the same probabilities. To achieve a synchronized behavior of the broadcasting nodes, each forwarding node u determines the same schedule r_w for a message originator w.[4] The value $r_w[i]$ denotes the transmission probability of the i^{th} time-slot in the schedule. Only a fraction $\frac{1}{c}$ with $c := 5 \cdot f(h)$ of all transmission probabilities are larger than 0. More precisely, for every interval $[c \cdot i, c \cdot (i + 1) - 1]$ of length c, one time slot with non-zero probability is chosen uniformly at random. For illustration consider the following example with $c = 2$: $[0, > 0, 0, > 0, > 0, 0, 0, > 0, ...]$.[5] The non-zero probabilities

[4] It would also be possible to let w determine the schedule r_w and append it to the message. Furthermore, the schedule could be changed for every broadcast initiated by w.

[5] Since the schedule r_w must be the same for all nodes, each node creating r_w must make the same random choices. From a practical point of view, this could be achieved

themselves can be chosen such as in [12] or [25]. For our purposes it suffices to stick to the simpler sequence [12], which starts with probability $\frac{1}{2}$, then $\frac{1}{4}$, followed by $\frac{1}{8}$ a.s.o. until $\frac{1}{\Delta}$.[6] Then the sequence repeats starting with probability $\frac{1}{2}$. An example of a schedule with $\Delta = 8$ and $c = 2$ is $[0, \frac{1}{2}, 0, \frac{1}{4}, \frac{1}{8}, 0, 0, \frac{1}{2}, 0, \frac{1}{4}, ...]$. For a schedule r_w the sequence $\frac{1}{2}, \frac{1}{4}, .., \frac{1}{\Delta}$ is repeated $e^7 \cdot f(h) \cdot h \cdot \log \frac{1}{\epsilon}$ times. Since only one slot out of $c = 5 \cdot f(h)$ slots is non-zero the length $|r_w|$ of the schedule r_w becomes $e^7 \cdot 5 \cdot f(h)^2 \cdot h \cdot \log \Delta \cdot \log \frac{1}{\epsilon}$, where e denotes the Euler constant.

A node u having received a message originated at w must transmit according to the schedule r_w. In our coloring algorithm (Section 9.4.2) a node u might have to forward several messages originating from up to $5 \cdot f(h)$ different nodes $w \in N^h(u)$. Therefore it is supposed to transmit according to multiple schedules at the same time. Unfortunately, it might be that two (or more) schedules overlap, i.e. there is a time slot where at least two schedules have non-zero transmission probability. In such a case a node transmits according to an arbitrary non-zero transmission probability. Furthermore, if a node u received a message from node w then it cannot just start transmitting according to the schedule upon reception of a message. Due to collisions neighbors of w might receive the message at different times and thus the schedule would not be followed synchronously. Thus, a node v broadcasting according to r_w appends the current position within the schedule, i.e. if it transmits with probability $r_w[i]$, it appends i. A node receiving a message waits until the sender completed the schedule (i.e. for $|r_w| - i$ time slots) before starting to transmit according to schedule r_w. Summing up, an actual message used in the protocol consists of three elements: a message msg, a time to live h stating how many hops a message will still be forwarded and finally the current slot in r. A node v keeps track of all messages it still has to broadcast using the set M_v.

9.4.2 Algorithm FastColoring

We present the main ideas of our coloring approach in Section 9.4.2 assuming synchronous wake-up and a given estimate of the maximal degree Δ. This simplifies the algorithm, allows to focus on the main ideas, and permits to directly relate our time complexity to previous work. In Section 9.4.2 we remedy these assumptions. Finally, in Section 9.4.2 we show how to get a distance-d coloring.

---

by initializing a pseudo random number generator with the ID of w.

[6] In case Δ is not known, it is approximated using n.

Algorithm Broadcast(hops h, message msg_w, failure probability ϵ)

For each node $v \in V$
1: $|r| := \mathrm{e}^7 \cdot 5 \cdot f(h)^2 \cdot h \cdot \log \Delta \cdot \log \frac{1}{\epsilon}$
2: **if** $v = w$ **then** $M_v := \{(msg_w, h+1, 0, Started)\}$ **else** $M_v := \{\}$ **end if**
3: **for** $i = 0..h \cdot |r|$ **do**
4: $m_C := (msg_u, h-1, t)$ for some entry $(msg_u, h, t, Started) \in M_v$ with $t < |r_u|$ and transmission probability > 0 (i.e. $r_u[t] > 0$)
5: **if** $m_C \neq \{\}$ **then**
6: Transmit m_C with probability $r_u[t]$
7: **else**
8: Listen for message m_L of the form (msg_u, h, t)
9: **if** (received m_L) $\wedge$ $(\nexists (msg_u, ., .) \in M_v) \wedge (h > 0)$ **then**
10: $M_v := M_v \cup (msg_u, h, t, NotStarted)$
11: **end if**
12: **end if**
13: Replace all entries $(msg_u, ., t, NotStarted) \in M_v$ with $t = |r_u|$ by $(msg_u, ., 0, Started)$ and all others by $(msg_u, ., t+1, Started)$
14: **end for**

Synchronous wake-up, Δ known

Algorithm FastColoring roughly iterates two main steps. First, a set of (independent) leaders is elected that control all other nodes within distance 6. Second, these leaders assign colors to their (direct) neighbors.

In order to get the set of leaders S^6 we first compute a MIS S^1 using algorithm [95] and then extend it using [106]. After the MIS S^1 is computed, all nodes are synchronized, i.e. nodes in the MIS wait until all nodes within 6 hops have completed algorithm [95] and tell (by broadcasting message $InMIS$) all nodes $N^6(v) \setminus S^1$ to stop and wait until the computation of S^6 is over. Next, a node $v \in S^1$ gets to know all nodes $u \in N^6(v) \cap S^1 \subseteq IS^6(v)$ within 6 hops in the independent set S^1. The algorithm presented in [106] requires $IS^6(v)$ and computes S^6. All leaders in S^6 ensure that they are not disturbed during their color assignment by broadcasting messages $DoNotTransmit$ and $DoNotTryMIS$. Then a leader $v \in S^6$ and its neighbors $u \in N(v)$ repeatedly execute 3 synchronized time slots. In the first slot every uncolored node u might apply for the permission to choose a color. If leader v received a request, it grants the request. Node u chooses an available color c. In the third slot node u informs its neighbors that color c is used.

Let us go through the algorithm in more detail. The broadcast of message $InMIS$ following the computation of MIS S^1 serves the purpose of telling nodes $v \notin S$ that the computation of the MIS S^1 is over and that they must wait before (re)starting algorithm TryMIS until all nodes in S^1 have finished the computation of S^6 (Lines 21 and 22). Additionally, using the $InMIS$ messages a node v gets to know all neighbors $u \in (N^6(v) \cap S)$, i.e. set $IS^6(v)$, (Lines 3 to 6) that joined S^1 in parallel with v, i.e. before having received a message $InMIS$. For communication nodes in the MIS S^1 use algorithm Broadcast (Section 9.4.1). Unfortunately, if a node u received a message msg_v broadcast by v, then v might not receive a message msg_u even if both broadcast up to the same hop distance h, i.e. call Broadcast(.,h,.,.). Thus per se the set $IS^6(v)$ is not symmetric, i.e. if $u \in IS^6(v)$ then not necessarily $v \in IS^6(u)$. However, symmetry is required in [106] to get set S^6 and therefore guaranteed (with high probability) in lines 7 to 8. The set S^6 is computed on the (undirected) graph $G' = (V' = \{u | u \in S\} = S, E' = \{\{u, v\} | u, v \in V', v \in IS^6(u)\})$ using algorithm [106]. The algorithm presented in [106] assumes that a node $v \in V'$ knows all nodes $u \in V'$ with $\{v, u\} \in E'$, i.e. v knows the set $IS^6(v)$ and that communication among nodes is reliable, i.e. no message is lost. Therefore, in case a node v misses some information of a node $u \in IS^6(v)$, e.g. due to a collision, it continuously sends a request to u asking for a retransmission. Furthermore, communication between two nodes $u, v \in V'$ is not just a simple transmission between direct neighbors in G, since their hop distance in G is up to 7. Thus we use Algorithm Broadcast with parameters $h = 7$ and $\epsilon = \frac{1}{\sqrt{2}}$. Once a node starts executing algorithm [106], it will execute it for a fixed number of steps, i.e. $c_5 \cdot \log \Delta \cdot \log n$ for some constant c_5. With high probability the computation of S^6 has finished by then. In case a node terminates the algorithm prior to the allowed number of steps it simple waits for the remaining time slots.

In the second step every node $v \in S^6$ assigns colors to all uncolored neighbors $u \in N(v)$. To begin with, nodes $v \in S^6$ ensure that all nodes within 3 hops remain quiet, while they are busy assigning colors. A node receiving a notification $DoNotTransmit(v)$ forwards the message $DoNotTransmit(v)$ and then must not transmit until it received a message $TransmitAndTryMIS(v)$. All messages $InMIS$ or $MIS6$ are ignored by a node while it is not allowed to transmit, in particular it will also not forward them once it is allowed to transmit again. Any interrupted broadcast of a message $DoNotTryMIS$ or $TransmitAndTryMIS$ originated at some node w is restarted from scratch, i.e. the schedule r_w is executed from slot 0 onwards such that the forwarding nodes still transmit synchronized. More precisely, assume a node u received message $DoNotTransmit(v)$ at time t_u^{DoNot}, when executing slot j of r_w and $TransmitAndTryMIS(v)$ at time t_u^{Can}. Node u will transmit again at time $(t_u^{DoNot} - j) + i \cdot |r_w|$, such that i

is the smallest positive integer with $(t_u^{DoNot} - j) + i \cdot |r_w| > t_u^{Can}$. If a node u received a broadcast message while not being able to transmit and was supposed to start the broadcast at time t_u, it will start it at time $t_u + i \cdot |r_w|$ such that i is the smallest positive integer with $t_u + i \cdot |r_w| > t_u^{Can}$. The message $DoNotTryMIS(v)$ ensures that only nodes in S^6 within 8 hops from v execute algorithm TryMIS while node v is assigning colors. In particular, the broadcast of messages $DoNotTryMIS$ and $DoNotTransmit$ ensures that if a node v assigns colors to its neighbors all neighbors are allowed to transmit and will get a color. A node u keeps track of all received messages $DoNotTryMIS$ and $DoNotTransmit$ using the set $WaitFor(u)$.

To obtain a color, nodes $v \in S^6$ and its uncolored neighbors $u \in N(v)$ begin executing algorithm *GetColor* concurrently. Essentially, algorithm *GetColor* repeats a schedule consisting of 3 synchronized time slots, i.e. all nodes $N(v) \cup v$ execute the same time slot concurrently. In the first slot every uncolored node u might apply for a color by sending a request to color itself. If the node $v \in S^6$ performing the assignment, received a message $Request(u)$, it grants the request by transmitting $Grant(u)$ in the second slot. Upon reception of message $Grant(u)$, node u chooses an available color c, transmits $Taken(c)$ in the third slot to all its neighbors and exits the algorithm. Every uncolored node u keeps track of the used colors in its neighborhood, i.e. $TakenColors(u)$. It also maintains an estimate of the number of uncolored neighbors $\tilde{n}(u)$ (initially Δ)that might concurrently apply for a color. The probability that a node transmits a request is given by $\frac{1}{\tilde{n}(u)}$. Since more and more neighbors get colored over time and the estimate might overshoot the true number of uncolored nodes, the estimate is updated from time to time, i.e. if out of a sequence of $128 \cdot e^2 \cdot \log n$ time slots less than $32 \cdot \log n$ *Grant* messages were received the estimate $\tilde{n}(u)$ is divided by 2.

Arbitrary wake-up, Δ unknown

To bound Δ we simply use n. If all nodes are known to wake up within a time span of length t, any newly woken-up node listens for t time slots and forwards broadcasts but does not initiate any broadcasts, i.e. algorithm TryMIS is not started. This way, a node will be fully aware of the state of the computation; as for synchronous wake-up algorithm FastColoring can be used.

For arbitrary wake-up we only sketch the algorithm. The main difficulty is that a woken-up node is unaware of the state of its neighbors. For instance, a node u might wake up and have multiple leaders as neighbors that all assign colors. If u transmits, it might disturb the color assignment. Thus a leader v assigning colors, interrupts algorithm GetColor from time to time (say after $O(\log^2 n)$ slots) to broadcast messages $DoNotTransmit(v)$ and

node $v \in S$	uncolored node $u \in N(v)$
$color_v := 0$	$\tilde{n}(v) := \Delta$
Listen	Transmit $Request(u)$ with prob. $\frac{1}{\tilde{n}(v)}$
if received $Request(u)$ **then** Transmit $Grant(u)$	Listen
Sleep	**if** received $Grant(u)$ **then** $color_v := \min\{1, ..., \Delta\} \setminus TakenCo.(v)$ Transmit $Taken(c)$ and exit **else** Listen **end**
if not recv. msg $Request$ for $384e^2 \log n(\log \Delta + 1)$ slots **then** Exit **end**	**if** received $Taken(c)$ **then** $TakenCo.(v) := TakenCo.(v) \cup c$ **end**
	if recv. $< 32 \log n$ $Grant$ msgs and $\tilde{n}(v)$ unchanged for last $384e^2 \log n$ slots **then** $\tilde{n}(v) := \frac{\tilde{n}(v)}{2}$ **end**

Table 9.2: Algorithm *GetColor*: After initialization a node repeats a schedule of three (synchronized) time-slots

$DoNotTryMIS(v)$ again and all nodes that are only prevented from transmitting by v forward the message. However, more needs to be done: Assume a node u has received $DoNotTransmit$ from two or more different leaders. In that case, it cannot forward any message. If a node $w \in N(u)$ wakes-up and node u is its only neighbor, i.e. $N(w) = u$, then it does not receive any message and has to wait very long (i.e. $O(n)$) to be sure that it can transmit safely. To remedy this problem, we introduce some empty slots during the color assignment. More precisely, out of $f(8)$ slots we only use 1 slot for the color assignment. The other slots are used for forwarding broadcasts. The leader determines which slots are used and broadcasts the chosen slots together with the message $DoNotTransmit$, such that a node is not allowed to transmit in the chosen slots. Thus, there are enough free slots to forward broadcasts. However, there are two more issues: First, a leader does not necessarily color all its uncolored neighbors. Second, a (newly) woken-up node is unaware of the previously chosen colors, but still must avoid deciding on an already chosen color.

We address these two problems by splitting algorithm GetColor into a sequence of four repeated phases, each of duration $O(\log^2 n)$. Any woken-up node can try to get a color, whenever the first phase is (re)started. If a node chose a color, already chosen by node u, then u can place a veto. More precisely, during the first phase the number of uncolored neighbors of the leader is estimated and used to set the transmission probability. During the second phase, uncolored nodes apply for a color as before. During the

third and fourth phase all nodes u that faced a color collision, i.e. a node v chose a color previously selected by a node u, transmit a veto, forcing v to choose another color. During the third phase, a node u estimates the number of nodes that also faced a color collision and then uses this estimate to determine the transmission probability in the veto phase.

The estimate of the number of nodes having a color collision can be computed as follows. A node $w \in S^6$ broadcasts $StartGettingDensity$ for 2 hops and all nodes in $N^2(w)$ that faced a color collision execute the following protocol synchronously for $O(\log^2 n)$ time steps: Transmit $Prob(\frac{1}{2^i})$ with probability $\frac{1}{2^i}$ and for all i starting from 1 to $\log n$ for $O(\log n)$ time slots for each i. The estimate $|\tilde{N}(u)|$ is given by $\max\{2^i | u$ received $\mathrm{O}(\log n)$ messages $Prob(\frac{1}{2^i})\}$. The number of uncolored nodes $u \in N(w)$ can be estimated in an analogous way.

Distance-d coloring

We focus on synchronous wake-up. To compute a distance-d coloring (for constant d) with $\mathrm{O}(\Delta)$ colors, algorithm FastColoring can be used with minor modifications. It must be ensured that the distance among nodes assigning colors to their direct neighbors is at least $\min\{d+3, 6\}$, such that nodes with distance at least $d+1$ get the same color. Furthermore, leaders having assigned colors broadcast all (newly) assigned colors up to distance d. The assigned colors can be broadcast together with message $TransmitAndTryMIS$. Finally, we have to alter the distances of the various broadcasts. The broadcast of message $InMIS$ is performed with $h = \max\{d+2, 6\}$[7], of $MIS6$ with $h = \max\{d+2, 6\} + 1$, of $DoNotTryMIS$ with $h = \max\{d+2, 6\} + 3$ and of $TransmitAndTryMIS$ with $h = \max\{d+2, 6\} + 4$. The broadcast of $DoNotTransmit$ remains unchanged.

9.4.3 Analysis Algorithm Broadcast

Throughout the proof we assume that a broadcast is only performed up to a constant distance d and for constant $f(d)$. We use the notion of a subschedule $r_w^k \subseteq r_w$. The k^{th} subschedule r_w^k denotes the transmission probabilities of the slots $r_w[j]$ with $k \cdot d < j \leq k \cdot (d+1)$ with $d := \mathrm{e}^7 \cdot 5 \cdot f(h)^2 \cdot h \cdot \log \Delta$ and $0 \leq k \leq \log \frac{1}{\epsilon}$.

The first lemma states that all nodes broadcasting a message according to schedule r_w follow the schedule synchronously, i.e. transmit with the same probability $r_w[t]$ for some $0 \leq t \leq |r_w| - 1$.

[7] Distance $d+2$ suffices, since the distance between nodes in the MIS S^{d+2} is $d+3$.

Lemma 54. *If two nodes $u, v \in V$ transmit concurrently according to schedule r_w then both transmit the same message (msg_w, h, t) with probability $r_w[t]$ for some $0 \leq t \leq |r_w| - 1$ at the same time.*

Proof. Assume node u received the message from node s and node v from node t and both s and t execute the schedule r_w synchronously. More precisely, say node v received message (msg_w, h, t_v) at local time t_s and node u received message (msg_w, h, t_u) x slots later at time $t_s + x$. Since s and t transmit the schedule synchronously (i.e. are at the same position in the schedule) and t_u gives the current position, we have $t_u = t_v + x$. Due to the algorithm node v will start executing schedule r_w from slot 0 onwards at time $t_s + |r_w| - t_v$ (of node s) and node u at time $t_s + x + |r_w| - t_u = t_s + x + |r_w| - t_v - x = t_s + |r_w| - t_v$. Thus both nodes will start in parallel.

The case when both nodes u, v received the message from the same node is analogous. □

The next lemma shows that if some neighbors of node v follow (concurrently) one subschedule r_w^k then the chance that node v receives a message is constant.

Lemma 55. *Assume neighbors $T \subseteq N(v)$ of node v transmit according to schedule r_w. After an execution of some subschedule r_w^k with $0 \leq k \leq \log \frac{1}{\epsilon}$, node v has received msg_w with probability $1 - \frac{1}{e^2 \cdot f(h) \cdot h}$.*

Proof. By definition (Section 9.4.1) within distance h of v at most $5 \cdot f(h)$ schedules are allowed to be executed concurrently and each schedule only contains one entry with value larger than 0 out of $5 \cdot f(h)$ entries. The chance $p_{r_w^k}$ that all nodes $u \in N(v)$ transmit either according to $r_w^k[i]$ for some entry $r_w^k[i] > 0$ or not at all, is at least the probability that the (current) entry of all other concurrently executed schedules (at most $5 \cdot f(h) - 1$) is 0:

$$p_{r_w^k} \geq (1 - \frac{1}{5 \cdot f(h)})^{5 \cdot f(h) - 1} \geq \frac{1}{\mathrm{e}}$$

The probability $p_{exactly1}$ that exactly one node $u \in T$ transmits if all nodes in T transmit with probability $\frac{1}{2 \cdot |T|} \leq r_w^k[i] \leq \frac{2}{|T|}$ is

$$p_{exactly1} \geq |T| \cdot \frac{1}{r_w^k[i]} \cdot (1 - r_w^k[i])^{|T|-1}$$

$$\geq |T| \cdot \frac{1}{2 \cdot |T|} \cdot (1 - \frac{2}{|T|})^{|T|-1} \geq \frac{1}{2} \cdot (1 - \frac{2}{|T|})^{|T|-1} \geq \frac{1}{2 \cdot \mathrm{e}^2}$$

Thus the total probability that node v receives the message within $5 \cdot f(h)$ slots is $p_{r_w^k} \cdot p_{exactly1} \geq \frac{1}{\mathrm{e}^4}$.

Due to construction schedule r_w^k with $|r_w^k| = \mathrm{e}^7 \cdot 5 \cdot f(h)^2 \cdot h \cdot \log \Delta$ contains one entry $r_w^k[i]$ out of $5 \cdot f(h) \cdot \log \Delta$ many with $\frac{1}{2 \cdot |T|} \leq r_w^k[i] \leq \frac{2}{|T|}$ and $0 < |T| \leq \Delta$. Thus when executing r_w^k a node transmits $\mathrm{e}^7 \cdot f(h) \cdot h$ times with probability p for $\frac{1}{2 \cdot |T|} \leq p \leq \frac{2}{|T|}$. The chance that all attempt fail is: $(1 - \frac{1}{\mathrm{e}^4})^{\mathrm{e}^7 \cdot f(h) \cdot h} \leq \frac{1}{\mathrm{e}^{2 \cdot f(h) \cdot h}}$. □

Theorem 56. *The probability that a message originated at w reaches all nodes $u \in N^h(w)$ and no node $u \in N^{h+2}(w) \setminus N^{h+1}(w)$ within time $O(\log \Delta \cdot \log \frac{1}{\epsilon})$ is at least $1 - \epsilon$.*

Proof. When a node receives a message $(msg_w, d, .)$ it will transmit $(msg_w, d-1, .)$ as long as $d > 0$. Since the originator transmits $(msg_w, h, .)$, a message will only reach nodes up to distance $h + 1$.

The neighborhood $N^h(w)$ can be covered by a MIS S^1 with $|S^1| \leq f(h)$ (see Definition 5). Clearly, any node $v \in S^1$ is reachable by a path $P = \{w = u_0, u_1, ..., u_h\}$ of length at most h nodes. Since $|S^1| \leq f(h)$ we need at most $f(h)$ (distinct) paths to get from w to all nodes $v \in S^1$. To ensure that the message reaches node v, a node u_i with $0 \leq i < h$ must have/receive a message $(msg_w, d, .)$ with $d \geq h - i$. Clearly, this holds for $u_0 = w$. Assume that for a node u_i all neighbors forward a message $(msg_w, d, .)$ with $d \geq h-i$. Then the chance p_k that u_i receives the message after the execution of a subschedule r_w^k is $p_k := 1 - \frac{1}{\mathrm{e} \cdot f(h) \cdot h}$ (Lemma 55). The chance that a message is forwarded along a path of length h (i.e. h consecutive times) due to r_w^k is $(p_)k^h$. The chance that this happens for at most $f(h)$ paths is $(p_k)^{f(h) \cdot h}$. Once all nodes $v \in S^1$ have the message, they must transmit it to their neighbors (By definition every node $u \in N^h(w)$ is adjacent to a node $v \in S^1$). Thus the total chance that a message is forwarded to all nodes $v \in S^1$ and they transmit it to their neighbors if a single subschedule r_w^k is executed is $(p_k)^{f(h) \cdot (h+1)} = (1 - \frac{1}{\mathrm{e}^2 \cdot f(h) \cdot h})^{f(h) \cdot (h+1)} \geq \frac{1}{2}$. Since a node having received a message starts the execution after its neighbors have completed all subschedules, we get that the failure probability after having executed $\log \frac{1}{\epsilon}$ subschedules is $\frac{1}{2}^{\log \frac{1}{\epsilon}} = \epsilon$ □

9.4.4 Analysis Algorithm FastColoring

Synchronous wake-up, Δ known

After the first three lemmas, we prove the correctness and time complexity of all steps of the algorithm one after the other. We assume that $n > \mathrm{e}^{(\log^* n)^{c_3 - 1}}$ for some constant

c_3.[8] By M^{Broad} we denote the set of all broadcasts of any message $msg \in \{InMIS, MIS6, DoNotTryMIS, DoNotTransmit, TransmitAndTryMIS\}$ performed during algorithm FastColoring.

The upcoming lemma is an extension of Lemma 54 and shows that despite a delayed start or interruption of a (broadcast) schedule due to messages $DoNotTransmit$ and $TransmitAndTryMIS$ nodes still execute any broadcast in a synchronized manner.

Lemma 57. *Lemma 54 holds even if schedule r_w got interrupted or its start got delayed.*

Proof. Due to Lemma 62 we can assume that,initially, all nodes $u, v \in V$ execute schedule r_w synchronously. Assume a node u received message $(DoNotTransmit(s), ., y_u^{DoNot})$ at time t_u^{DoNot} and $TransmitAndTryMIS(s)$ at time t_u^{Can}. Say, x slots later, node v got $(DoNotTransmit(z), ., y_v^{DoNot})$ and $TransmitAndTryMIS(z)$ at time t_v^{Can}. Since y_v^{DoNot} gives the current position in the schedule, which advanced by x slots since $(DoNotTransmit(w), ., y_u^{DoNot})$ has been transmitted, we have $y_v^{DoNot} = y_u^{DoNot} + x$.

Node u starts r_w at time $(t_u^{DoNot} - y_u^{DoNot}) + i_u \cdot |r_w|$, such that i_u is the smallest positive integer with $(t_u^{DoNot} - y_u^{DoNot}) + i_u \cdot |r_w| > t_u^{Can}$. Node v restarts at time $(t_u^{DoNot} + x - y_v^{DoNot}) + i_v \cdot |r_w| = (t_u^{DoNot} - y_u^{DoNot}) + i_v \cdot |r_w| > t_v^{Can}$, such that for integer i_v holds $(t_u^{DoNot} - y_u^{DoNot}) + i_v \cdot |r_w| > t_v^{Can}$. Thus either i_v equals i_u and both nodes start synchronously or in case they are distinct they execute schedule r_w sequentially.

The proof for a delayed start is analogous. □

The next two lemmas show that all nodes assigning colors have distance at least 7. The first lemma proves the claim if all nodes are allowed to transmit. The second lemma shows that the claim holds even if some nodes are not permitted to transmit.

Lemma 58. *Assuming that no broadcast in M^{Broad} fails and all nodes are allowed to transmit, S^6 and S^1 are computed (correctly) in time $O(\log \Delta \cdot \log n)$, then no nodes with distance less than 7 will ever assign colors at the same time.*

Proof. Assume that two nodes u, v with $v \in N^6(u)$ have joined the MIS S^6 while all nodes are allowed to transmit (i.e. $\nexists DoNotTransmit \in WaitFor(s)$ for all $s \in V$). Assume u has joined first. Node v must have joined S^1 before the broadcast of $InMIS(u)$ could have reached v. If $InMIS(u)$ reached v before it joined S^1, it would have stopped executing algorithm

[8] The assumption can be dropped changing the overall time complexity from $O(\Delta + \log \Delta \cdot \log n)$ to $O(\Delta + \log \Delta \cdot \log n \cdot \log^* n)$.

TryMIS upon reception of a message $InMIS$ and would have waited with the restart for at least $constant \cdot \log \Delta \cdot \log n$ slots. Since S^6 is computed in time $O(\log \Delta \cdot \log n)$ (Lemma 61) and a broadcast is also in $O(\log \Delta \cdot \log n)$ (Lemma 56), the *constant* can be chosen such that for any node u that joined S^6 the message $DoNotTryMIS(u)$ will have reached all nodes within distance 8 of u before they (re)start algorithm TryMIS. Clearly, after the reception of $DoNotTryMIS(u)$ a node must wait until it received a message $TransmitAndTryMIS$ and thus no node $v \in N^6(u)$ is able to join S^6, while u assigns colors. Thus node v must have joined S^1 before it received $DoNotTryMIS(u)$ or $InMIS(u)$. Since u broadcasts $InMIS(u)$ directly after it joined, node v must have joined while the broadcast was still going on. Thus, node v will receive the broadcast of $InMIS(u)$ and add u to its set $IS^6(v)$. But also u must receive $InMIS(v)$ since u waits for the duration of a broadcast $InMIS$ after it completed the broadcast $InMIS(u)$. With the same argument node v will receive $IS^6(u)$ and node u receives $IS^6(v)$ before it starts the computation of S^6. Therefore, we have that $u \in IS^6(v)$ and $v \in IS^6(u)$ and by assumption S^6 is computed correctly given a correct input to algorithm [106] and thus all nodes have distance 7. □

Lemma 59. *Assuming that no broadcast in M^{Broad} fails, S^6 and S^1 are computed (correctly) in time $O(\log \Delta \cdot \log n)$, then no nodes with distance less than 7 will ever assign colors at the same time.*

Proof. Due to Lemma 58 the statement is true, if all nodes are allowed to transmit. For the proof we require that a message $DoNotTryMIS(v)$ is received by some node u before (or concurrently) it received $DoNotTransmit(v)$. We refrain from the lengthy proof and remark that the two separate broadcasts of $DoNotTryMIS$ and $DoNotTransmit$ can be replaced by a single $Broadcast(v, \{DoNotTransmit(v), DoNotTryMIS(v)\}, 8, \frac{1}{n^5})$. If a node has received $(\{DoNotTransmit(v), DoNotTryMIS(v)\}, 5, \frac{1}{n^5})$, it forwards the message without $DoNotTransmit(v)$.

Consider a node $v \in S^1$ (possibly also in S^6) for which some nodes in $T \subseteq N^6(v)$ did not receive a broadcast initiated by v, i.e. the broadcast got interrupted due to nodes that are not allowed to transmit. Let set $W \subseteq S^6$ be the set of nodes which caused the nodes not to transmit, i.e. each node $w \in W$ broadcast $DoNotTransmit$ and some node $s \in N^6(v) \cap N^4(w)$ received the message.

For every node $u \in T \subseteq N^6(v)$ and every path from v to u (of length at most 6) there must be a node s that received a message $DoNotTransmit(w)$ by a node $w \in W$. Otherwise u would have received the broadcast by v. Let s be the closest node to u. Since both s and u are on a path of length at most 6, we have that the distance from s to u is at most 5. Since s

is assumed to be the closest non-transmitting node to u and node s will forward $DoNotTryMIS(w)$ for 5 more hops (see above: $DoNotTryMIS$ is forwarded 5 hops more than $DoNotTransmit$), node u must have received $DoNotTryMIS(w)$. Since a node u will not start TryMIS for *constant* $\log\Delta\cdot\log n$ steps, node u will not execute algorithm TryMIS before node v completed its broadcasts $DoNotTryMIS(v)$ and $DoNotTransmit(v)$ (Same argument as in proof of Lemma 58).

As long as node u has not received $DoNotTryMIS(v)$, such a non-transmitting node s must exist and node u will not execute TryMIS because of a message $DoNotTryMIS(w)$ with $w \in W$. □

The following lemma deals with the time complexity of algorithm [95]. In the paper the running time of the algorithm is $O(\log^2 n)$ with probability $1-\frac{1}{n}$ without an estimate of Δ. We briefly show how to increase the success probability (using more time slots though) and how to make use of the estimate Δ.

Lemma 60. *Within time $O(\log\Delta\cdot\log n)$ MIS S^1 is computed with probability $1-\frac{1}{n^5}$.*

Proof. The success probability can be increased from $1-\frac{1}{n}$ to $1-\frac{1}{n^5}$ by using a factor 5 more time slots. Throughout the proof in [95] bounds of the form $e^{c\cdot\log n}=\frac{1}{n^c}$ are used. Therefore a multiplying the occurring constants by a factor of 5 yields the desired result.

With an estimate of Δ the time complexity can be improved to $O(\log\Delta\cdot\log n)$ by replacing n in p_v by Δ in Algorithm 1 in [95], the bound in Line 3 of $4\mu\delta\log^2 n$ by $4\mu\delta\log\Delta\log n$, $\log n$ by $\log\Delta$ in Line 15 and finally $\delta\log^2 n$ by $\log\Delta\log n$ in Line 16. [9] □

The next lemma shows that the set S^6 is computed correctly and efficiently using algorithm [106].

Lemma 61. *Assuming that no broadcast in M^{Broad} fails, algorithm [106] computes a correct S^6 within time $O(\log\Delta\cdot\log n)$ with probability $1-\frac{1}{n^4}$.*

Proof. Due to Lemma 59 the input for algorithm [106] is correct, i.e. the set $IS^6(v)$ for a node v contains all other nodes in S^1 that might join S^6 within distance 6 and is also symmetric, i.e. if $u\in IS^6(v)$ then also $v\in IS^6(u)$.

By definition set S^6 corresponds to a MIS in the graph $G'=(V'=\{u|u\in S\},E'=\{\{u,v\}|u,v\in V',v\in IS^6(u)\})$ on which we run the deterministic algorithm MIS [106]. (The graph is undirected since for $v\in IS^6(u)$ also $u\in IS^6(v)$.) The proof of the correctness of algorithm [106] in the message

[9] We refrain from restating the full proof from [95], since the improvement of the original bound of $O(\log^2 n)$ to $O(\log\Delta\cdot\log n)$ is not of crucial importance.

passing model, where no collisions occur, is given in [106]. Thus we must show that algorithm [106] receives all required information despite of lost messages. To compute a MIS on G', node $v \in S^1$ has to exchange messages with all neighbors $u \in IS^6(v)$ in G (by the definition of graph G'). In case node v misses some information of some neighbor u in G', it halts the execution of algorithm [106] and asks u (via a broadcast) to retransmit the data. Thus algorithm [106] eventually receives all necessary information for the computation, progresses and finishes outputting a correct result.

The time complexity can be derived as follows. The graph G' has constant degree, i.e. the number of neighbors of a node $u \in S^6$ in G' is bounded by the size of a MIS in G within 7 hops, which is $f(7)$ (by definition of f). The graph G' is thus also of bounded-independence with $f'(h) \leq f(7 \cdot h)$. Opposed to the message passing model, a node v cannot exchange messages with its neighbors v in one round but needs to use Algorithm Broadcast for it. Algorithm [106] progresses one step in its computation once every node $v \in S^1$ performed a successful broadcast. The number of required steps is bounded by $\mathrm{O}(f'(f'(2)+1) \cdot \log^* n) = O(\log^* n)$ due to the analysis in [106]. This implies that for a node v only a node $u \in N^{O(\log^* n)}(v) \cap S^1$ can influence v's computation, i.e. if such a node u fails to broadcast a message, node v might be delayed. The number of these neighbors is bounded by $|N^{O(\log^* n)}(v) \cap S^1| = f'(O(\log^* n)) \leq c_2 \cdot (\log^* n)^{c_1} = (\log^* n)^{c_3}$ with constants c_1, c_2, c_3.

Due to Theorem 56 the chance that Algorithm Broadcast with $\epsilon = \frac{1}{\sqrt{2}}$ transmits a message originated at v to all nodes $u \in IS^6(v)$ within time $\mathrm{O}(\log \Delta)$is at least $\frac{1}{\sqrt{2}}$. If a broadcast message was not delivered to some node, this node asks for a retransmission. The chance that both the request and the retransmission succeed is $\frac{1}{2}$. Let f_i be the number of requests and retransmissions for v until all $(\log^* n)^{c_3}$ nodes that influence the computation of v have performed one successful broadcast, i.e. algorithm [106] can execute one of the $O(\log^* n) = c_0 \cdot \log^* n$ (for some constant c_0) steps. The chance that for one node out of $(\log^* n)^{c_3}$ the request and retransmission broadcast fail f_i times in a row is $1 - (1 - \frac{1}{2^{f_i}})^{(\log^* n)^{c_1}}$, in particular if f_i is larger than $g := \frac{2 \cdot \log n}{\log^* n}$ the chance becomes $1 - (1 - \frac{1}{2^{f_i}})^{(\log^* n)^{c_3}} \leq \frac{1}{2^{\frac{f_i}{2}}}$ for $g > (\log^* n)^{c_3}$ (i.e. $n > \mathrm{e}^{(\log^* n)^{c_3 - 1}}$). We used $(1 - \frac{1}{2^{\frac{2 \log n}{\log^* n}}})^{(\log^* n)^{c_3}} = (1 - \frac{1}{n^{\frac{2}{\log^* n}}})^{(\log^* n)^{c_3}} \leq \frac{1}{n^{\frac{1}{\log^* n}}}$. The probability that more than $c_4 \cdot \log n$ (with constant c_4) broadcasts are needed (i.e.$\sum_{i=0}^{c_0 \cdot \log^* n} f_i > c_4 \cdot \log n$) can be bounded as follows: The total time until algorithm [106] computed MIS S_i

is estimated from above as follows:

$$\sum_{i=0}^{c_0\cdot\log^* n} f_i \leq \sum_{i=0, f_i\leq g}^{c_0\cdot\log^* n} f_i + \sum_{i=0, f_i>g}^{c_0\cdot\log^* n} f_i$$

$$\leq \sum_{i=0}^{c_0\cdot\log^* n} g + \sum_{i=0, f_i>g}^{c_0\cdot\log^* n} f_i \leq O(\log n) + \sum_{i=0, f_i>g}^{c_0\cdot\log^* n} f_i$$

Assume we fix a set e_k^{fix} of values $t_i > g$ with $0 \leq i \leq c_0\cdot\log^* n$ arbitrarily, such that the sum of all t_i equals k ($\sum_{i=0,t_i>g}^{c_0\cdot\log^* n} t_i = k$). The chance that an event e_k^{fix} occurs, i.e. the total delay is k and the number of needed request-s/retransmissions f_i are as given by the set e_k^{fix} (i.e. $f_i = t_i$) is $prob$(event e_k^{fix} occurs) $= \prod_{i=0}^{c_0\cdot\log^* n} prob(t_i = f_i) \leq \prod_{i=0}^{c_0\cdot\log^* n} \frac{1}{2^{\frac{f_i}{2}}} = \frac{1}{2^{\sum_{i=0}^{c_0\cdot\log^* n}\frac{f_i}{2}}} = \frac{1}{2^{\frac{k}{2}}}$. The number of events e_k^{fix}, which add up to a fixed k, can be bounded by

$$(c_0\cdot\log^* n)!\cdot\sum_{i_0=0}^{k}\sum_{i_1=0}^{k-i_0}\cdots\sum_{i_j=0}^{k-\sum_{s=0}^{j-1} i_s}\cdots\sum_{i_{c_0\cdot\log^* n}=0}^{k-\sum_{s=0}^{c_0\cdot\log^* n-1} i_s} 1$$

$$= k^{c_0\cdot\log^* n}$$

Therefore the chance that all fail becomes $(1-\frac{1}{2^{\frac{k}{2}}})^{k^{c_0\cdot\log^* n}}$. For $k \geq 32\cdot\log n$ we get $(1-\frac{1}{2^{\frac{k}{2}}})^{k^{c_0\cdot\log^* n}} \geq (1-\frac{1}{2^{\frac{k}{4}}})$. The chance none of all possible e_k^{fix} for any $k \geq 32\cdot\log n$ occurs is:

$$\prod_{k=32\cdot\log n}^{\infty}(1-\frac{1}{2^{\frac{k}{4}}}) = 2^{\sum_{k=32\cdot\log n}^{\infty}\log(1-\frac{1}{2^{\frac{k}{4}}})} \geq 2^{\sum_{k=32\cdot\log n}^{\infty}-\frac{1}{2^{\frac{k}{4}}}}$$

$$= 2^{-\frac{2\cdot 2^{-\frac{32\cdot\log n}{4}}}{2-2^{\frac{3}{4}}}} \geq 2^{-\frac{1}{n^7}} \geq 1-\frac{1}{n^7}$$

where we used $\log(1-x) \geq -x$ for $0 \leq x \leq 0.1$.

Thus the chance to have more than $O(\log n)$ failures is less than $\frac{1}{n^7}$. The time to perform $O(\log n)$ broadcasts with failure probability $\frac{1}{2}$ is $O(\log\Delta\cdot\log n)$ for $n > e^{(\log^* n)^{c_3-1}}$. □

The next three lemmas deal mainly with algorithm GetColor. Lemma 62 guarantees a synchronous start of GetColor for a node $v \in S^6$ and its uncolored neighbors $u \in N(v)$. Lemma 63 proves the correctness of the coloring and Lemma 64 gives a bound on the time complexity of algorithm GetColor.

Lemma 62. *Given that no broadcast in M^{Broad} fails, a node $v \in S^6$ and all uncolored nodes $u \in N(v)$ start and execute algorithm GetColor concurrently and no neighbor $u \in N^3(v) \setminus N(v)$ transmits.*

Proof. Due to Lemma 59 and 61 no nodes within distance 6 will assign colors concurrently. Since a node granting colors broadcasts $DoNotTransmit$ up to 4 hops right before entering GetColor, for a node $v \in S^6$ executing GetColor will hold that all nodes $N^3(v)$ have received message $DoNotTransmit(v)$ and will not transmit after forwarding the message (except for uncolored nodes $u \in N(v)$, which must transmit to get colored). In particular no neighbor $u \in N^2(v)$ will transmit, when v transmits $GrantingColors$ and therefore all uncolored nodes $u \in N(v)$ call algorithm GetColor concurrently. □

Lemma 63. *Given that no broadcast in M^{Broad} fails, all uncolored nodes $u \in N(v) \cup v$ with $v \in S^6$ will obtain a (correct) color on termination of algorithm GetColor.*

Proof. Due to Lemma 62 all uncolored nodes $u \in N(v) \cup v$ execute algorithm GetColor concurrently and no node $w \in N^3(v) \setminus (N(v) \cup v)$ transmits while v executes GetColor. A color is assigned using three time slots and due to the synchronous start no collision occurs in slots 2 and 3 of the schedule of algorithm GetColor.

No node picks a color already chosen by a neighbor. A node $v \in S^6$ chooses color 0. Since no node $u \notin S^6$ chooses color 0 and node v colors all its uncolored neighbors, no neighbor of node v will attempt to join S^6 after it has executed algorithm GetColor. Thus no node $u \in N(v)$ will get color 0. Consider a node $u \in V \setminus S^6$. Whenever a neighbor $w \in N(u)$ chooses some color c, there must be a node $v \in S^6 \cap N^2(u)$ in its 2 hop neighborhood. Since node v has transmitted $DoNotTransmit(v)$ up to at least 3 hops before starting to assign colors and only one node $w \in N(v)$ transmits in the third slot of algorithm GetColor, node $u \in N^2(v)$ will receive any message $Taken(c)$ by a neighbor $w \in N(u)$ and store it in its $TakenColors$ set. Therefore node u won't choose an already chosen color.

Since a node u has a neighbor $v \in S^6$ (which has color 0), once it can choose a color, we have that $|N(u) \setminus v| \leq \Delta - 1$. Thus the number of required colors for its neighbors is at most Δ and one color remains for u. □

Lemma 64. *Given that no broadcast in M^{Broad} fails, a node $v \in S^6$ and all uncolored nodes $u \in T \subseteq N(v)$ terminate algorithm GetColor within $O(\Delta + \log \Delta \cdot \log n)$ time slots with probability $1 - \frac{1}{n^4}$.*

Proof. If a node $v \in S^6$ receives a request by a node $u \in N(v)$, it can transmit $Grant(u)$ without collision and node u can transmit a message $Taken$ without

collision due to Lemma 63. Thus some neighbor $u \in N(v)$ gets colored, if it transmits a request without collision.

For a sequence of $t_s := 384 \cdot e^2 \cdot \log n$ time slots, either at least $8 \cdot \log n$ nodes got colored (i.e. *Grant* and Taken messages were transmitted) or the transmission probability $\frac{1}{\tilde{n}(v)}$ is doubled by each uncolored neighbor $u \in T$. Assume that (directly) after a doubling of $\tilde{n}(v)$ we have that $n(v) \leq \tilde{n}(v) \leq 2 \cdot n(v)$. The chance that the transmission probability of an uncolored node gets doubled if $\frac{n(v)}{2} \leq \tilde{n}(v)$ can be computed as follows. The probability that a node $u \in T$ transmits a request without collision is $\frac{1}{\tilde{n}(v)} \cdot n(v) \cdot (1 - \frac{1}{\tilde{n}(v)})^{n(v)-1} \geq \frac{1}{2} \cdot \frac{1}{e^2} = \frac{1}{2 \cdot e^2}$. Out of the t_s time slots, we use one third (i.e. $128 \cdot e^2 \log n$) for transmitting requests and thus expect at least $64 \cdot \log n$ transmissions of *Request* messages without collision. Using a Chernoff bound, the chance that there are less than $32 \cdot \log n$ (successful) transmissions is: $p(\text{less than } 32 \cdot \log n \textit{ Requests} \text{ transmitted}) \leq e^{\frac{1}{8} \cdot 64 \cdot \log n} = \frac{1}{n^8}$. Therefore, either a constant fraction of the last t_s time slots were used for successful transmissions or half of the remaining nodes got colored between two doublings of the transmission probability. Since $\tilde{n}(v) \leq \Delta$ after the probability is doubled at most $\log \Delta + 1$ times, all nodes have been colored with probability $1 - \frac{1}{n^4}$. If no sequence fails (i.e. $\tilde{n}(v)$ is halved despite $\tilde{n}(v) \leq n(v)$), then for every sequence of t_s slots holds: Either the transmission probability is multiplied by two or $32 \cdot \log n$ nodes get colored. The number of successful sequences to color all nodes is given by $O(\Delta + \log \Delta \cdot \log n) \in O(n)$. The chance that all sequences succeed is at least $(1 - \frac{1}{n^8})^{c \cdot n} \geq 1 - \frac{1}{n^4}$ for some constant c.

The chance that the leader $v \in S^6$ exits the algorithm before having assigned colors to all neighbors, can be computed as follows. Due to the algorithm, the leader only exists if it has not received a request for $384 \cdot e^2 \cdot \log n \cdot (\log \Delta + 1) = t_s \cdot (\log \Delta + 1)$ time slots. Within the last t_s time slots either there was at least one transmission without collision of a message *Request* or the transmission probability got doubled with probability at least $1 - \frac{1}{n^8}$ (see previous paragraph). The number of sequences as well as the overall success probability become $1 - \frac{1}{n^4}$. Therefore the time complexity becomes $O(\Delta + \log \Delta \cdot \log n)$ with probability $(1 - \frac{1}{n^4})^2$. □

Theorem 65. *Within time $O(\Delta + \log \Delta \cdot \log n)$ every node is colored with probability $1 - \frac{1}{n^3}$ using $\Delta + 1$ colors.*

Proof. Given that no broadcast in M^{Broad} fails, due to Lemma 60 MIS S^1 is computed within time $O(\log \Delta \cdot \log n)$. Thanks to Lemma 61 and 59, algorithm FastColoring correctly computes a set S^6 within time $O(\log \Delta \cdot \log n)$ with probability $1 - \frac{1}{n^4}$. A node $v \in S^6$ correctly colors its neighbors within time $O(\Delta + \log \Delta \cdot \log n)$ with probability $1 - \frac{1}{n^4}$ due to Lemma 64.

An uncolored node $v \in V$ must have a neighbor within 15 hops that is coloring all its uncolored neighbors (i.e. executing algorithm GetColor) or will do so within time $O(\log \Delta \cdot \log n)$. If node v is not executing algorithm TryMIS, then in case it is waiting due to a node $v \in S^6$ having transmitted message $DoNotTryMIS$ the statement holds. In case it is waiting due to message $InMIS$, i.e. due to a node $u \in S \cap N^7(v)$ then some neighbor $w \in N^7(u) \subseteq N^{15}(v)$ will join S^6 within $O(\Delta + \log \Delta \cdot \log n)$.

If a node assigns colors to its neighbors, all uncolored neighbors get colored (see Lemma 63). Therefore two nodes $u, v \in V$ that assign colors to all their neighbors are independent. Within distance 16 any node has at most $f(16)$ independent nodes (by definition of function f). The time until a node in $N^{16}(v)$ calls algorithm GetColor and finishes is $O(\Delta + \log \Delta \cdot \log n)$, thus after time $f(16) \cdot O(\Delta + \log \Delta \cdot \log n)$ any node must have a neighbor that assigns colors.

Next we bound the number of necessary broadcasts, i.e. $|M^{broad}|$. A node can only issue a broadcast, once it executes TryMIS. Per execution of algorithm TryMIS it issues at most 5 broadcasts. If a node executes algorithm TryMIS then using the same reasoning as above a node will get colored after at most $f(16)+1$ calls to TryMIS. Therefore overall at most $5 \cdot (f(16)+1) \cdot n$ successful broadcasts are needed. The chance that no broadcast and no execution of algorithm MIS [95] fails is $(1 - \frac{1}{n^5})^{6 \cdot (f(16)+1) \cdot n}$. Due to Theorem 56 the time for one broadcast is $O(\log \Delta \cdot \log n)$.

The chance that all executions of algorithm GetColor work in time $O(\Delta + \log \Delta \cdot \log n)$ (see Lemma 64) is given by $(1 - \frac{1}{n^5})^n$. If a node calls algorithm [106] to compute S^6 then after the calculation some neighbor (or itself) within distance 7 will color all its neighbors. Thus in total there are at most $f(7) \cdot n$ calls of algorithm [106]. The chance that all succeed in time $O(\log \Delta \cdot \log n)$ (see Lemma 61) is given by $(1 - \frac{1}{n^5})^{f(7) \cdot n}$.

The overall probability that all algorithms succeed in the given time and no broadcasts fails is given by $(1 - \frac{1}{n^4})^{c_4 \cdot n} \geq 1 - \frac{1}{n^3}$ with constant c_4. The overall time is bounded by $O(\Delta + \log \Delta \cdot \log n)$. □

Distance-d coloring

Most of the lemmas and proofs in Section 9.4.4 hold with minor modifications. The fact that $O(\Delta)$ colors are sufficient, can be seen as follows: All nodes within constant distance d of a node v can be covered by an independent set S_d^1 of constant size $f(d)$ (by Definition of f). Thus any node $u \in N^d(v)$ has at least one neighbor in S_d^1, thus the number of nodes $|N^d(v)|$ can be bounded as follows: $|N^d(v)| \leq \sum_{u \in S_d^1} d(u) \leq f(d) \cdot \Delta \in O(\Delta)$.

9.5 Usefulness of Collision Detection

9.5.1 MIS Algorithm

We present an algorithm containing the most essential ideas assuming simultaneous wake-up of all nodes in Section 9.5.1. In Section 9.5.1 we show how to extend the algorithm to allow for arbitrary wake-up times.

MIS, synchronous Wake-Up

In our deterministic algorithm a node performs a sequence of competitions against neighbors. After a competition a node might immediately compete again or it might drop out and wait for a while or it might join the MIS. During a competition a node transmits a value in a bit by bit manner, i.e. one bit per round only.

Since messages cannot be exchanged in parallel among interfering nodes, it looks like one communication round of a competition in the local model requires potentially $\Delta + 1$ rounds in the collision detection model. However, concurrent communication despite interference is possible, if a node v transmits its value r_v^j (with $r_v^0 := ID_v$) bit by bit (line 8 to 14), to get value r_v^{j+1} which is used to update its state and for the next competition. In case bit k of r_v^j is 1, node v transmits otherwise it listens. It starts from the highest order bit of r_v^j and proceeds bit by bit down to bit 0. As soon as it detected a transmission for the first time, say for bit l, node v sets its value r_v^{j+1} to l (line 11) and does not transmit for the remaining bits. If it has never detected a transmission while communicating r_v^j, its result is $\log^{(j)} n$. For example, consider the first competition of three nodes u, v, w, which form a triangle. Let ID_u be 1100, ID_v be 1001 and ID_w be 1101. Initially, all assume to have have highest ID, i.e. result $r_u^1 = r_v^1 = r_w^1 = 4$. In the first round all nodes transmit. In the second u and w transmit. Node v detects a transmission and sets its result to 1 and waits. In the third round no node transmits and in the fourth round w transmits and node u sets its result to 3, while w keeps its (assumed) result 4.

After each competition the states are updated in parallel (see algorithm Update State). A node starts out as *undecided* and competes against all *undecided* neighbors. For the first competition, which is based on distinct IDs, we can be sure that when node v transmitted its whole ID, i.e. has result $r_v^1 = \log n$, no other node $u \in N(v)$ has the same result $r_v^1 = r_u^1 = \log n$, since IDs differ. Thus, node v joins the MIS and informs its neighbors. All nodes in the MIS and their neighbors remain quiet and do not take part in any further competitions. For any competition $j > 1$ several nodes might be able to transmit their whole result bit by bit without detecting a transmission, e.g. $r_u^{j+1} = r_v^{j+1} = \log^{(j)} n$ for two adjacent nodes u, v. In this case, node v

changes its state to marked M. A marked node is on its way into the MIS but it will not necessarily join. A neighbor of a marked node remains quiet for a while. More precisely, the algorithm can be categorized into stages (lines 3 to 17), consisting of $f(2)+1$ phases (lines 4 to 13), being composed of a sequence of $\log^* n+2$ competitions. A node changes its state from undecided to some other state within a phase. An M node changes back to undecided after a phase (line 16). A neighbor of a marked node, i.e. an N_M node, changes back to undecided after a stage (line 18) and competes again in the next stage.

In order to update the state of neighbors of nodes having joined the MIS or having become M, two rounds are reserved. One round is used by M nodes to signal their new presence (line 4 in Algorithm Update State) and the other by *MIS* nodes (line 7). All other nodes listen during these rounds and update their states if required (lines 10 and 11).

Major differences in the proof compared to [106] can be found in Lemmas 66,72 and 74 as well as Theorem 77.

Definition 6. *A node $u \in V$ can be reached by a path p of M nodes from v, if*

$$\exists u \in \{w | (w \in V) \wedge \big(\exists p = (v = t_0, t_1, ..., u = t_j) | \forall (0 \leq i < j)(s_{t_i} = M)\big)\}$$

Let the set W_v be defined as:

$$W_v := \{u \in (N^2(v) \setminus N(v)) | u \text{ was competitor in the first competition}\}$$

Lemma 66. *No nodes in the MIS can be adjacent.*

Proof. Consider two cases: When a node v joins the MIS, no neighbor $u \in N(v)$ joins the MIS at the same time. Since all IDs are different, two IDs ID_v and ID_u must have some bit, which is 0 for ID_v and one for ID_u. Therefore, at most one of the two nodes could send its whole ID and obtain result $r = \log n$ and become a MIS node (line 10 in algorithm Update State).

Since a node v having joined the MIS signals its presence directly after the first competition, all neighbors $u \in N(v)$ adapt their state to $s_u = N_{MIS}$ and remain in that state. □

Lemma 67. *After a phase, i.e. after $\log^* n + 2$ competitions, no node is* undecided.

Proof. Observe that within a phase no node turns into an *undecided* node, i.e. once the state is different from *undecided* it remains different throughout the phase. The first competition is based on a number with $\log n$ bits, the second on a number of $\lceil \log \log n \rceil$ bits and so on. Thus after $\log^* n + 1$ the result of the competition is a single bit, i.e. 0 or 1. Then the following holds

for node v: if $\big((r_v = 1) \vee \big(\forall(u \in N(v))(r_u = 0)\big)$ then $s_v = M$ else $s_v = N_M$. Thus every node changes its state from *undecided* to another. □

Lemma 68. *After a phase every node that has once become N_{MIS} or N_M in one of the nested competitions $j \in [1, \log^* n + 2]$, is adjacent to a MIS or marked node.*

Proof. This follows since a node only becomes N_{MIS} or N_M if it is adjacent to a node in the MIS or a marked node and marked nodes do not update their states during a phase and a node in the MIS never leaves the MIS. □

Lemma 69. *Whenever a node v becomes marked together with some neighbors $U \subset N(v)$ in competition j and phase i, then no neighbor $w \in (N(v) \setminus U)$ can become marked or MIS in a later competition for the rest of the stage.*

Proof. When a node v becomes marked in phase i, all neighbors $u \in (N(v) \setminus U)$ must become N_{MIS} or N_M. During a stage a N_M node can only change to N_{MIS} or stay N_M (see Algorithm Update State). A N_{MIS} node does not change its state any more. If any node $u \in (N(v) \setminus U)$ had become a marked or MIS node itself in an earlier competition h with $h < j$, then v would be N_{MIS} or stay N_M and not compete any more. □

Lemma 70. *If a node v becomes a marked node in competition j of phase i and node u in a later competition in the same stage (but not necessarily in the same phase) then they cannot reach each other by a path of marked nodes.*

Proof. Assume u was reachable by a path $p = (v = t_0, t_1, ..., u = t_j)$ of marked nodes from v. Since v and u became marked in different competitions there must exist two neighboring marked nodes t_k and t_{k+1}, i.e. $t_{k+1} \in N(t_k)$, such that t_k became marked in competition j of phase i and t_{k+1} in competition t with $t > j$ of phase i or during phase s with $s > i$. But due to Lemma 69 nodes t_k and t_{k+1} cannot be neighbors. □

Lemma 71. *If a marked node v can reach another marked node u by a path of marked nodes then $r_v = r_u$ and node v,u have the same prefix, i.e. $y_v^i = y_u^i$ for $0 \leq i \leq r_v$.*

Proof. Due to lemma 70 we know that v and u must have become marked in the same competition of the same phase. Assume u was reachable by the path $p = (v = t_0, t_1, ..., u = t_j)$ of marked nodes from v and $r_v \neq r_u$. Then there must exist two marked neighbors t_k and t_{k+1} with $r_{t_k} \neq r_{t_{k+1}}$. Since either $r_{t_k} > r_{t_{k+1}}$ or the other way round, they could not both have become marked in the same competition of the same phase. Assume their prefixes differed, i.e. $y_v^k \neq y_u^k$ for $0 \leq k < r_v$. Then r_v could not be equal to r_u. □

Lemma 72. *Consider a node v, which has become marked in the j^{th} competition. If $|W_v| > 0$ then $\exists w \in W_v$, which cannot be reached by a path of marked nodes from v.*

Proof. Let us investigate the first competition, which is based on IDs.

Consider the value of r_v. If $r_v = \log n$, then v is a MIS node. If $r_v = 0$, then by definition $y_v^0 = 0$ and node v must have had a neighbor u with $y_u^0 = 1$. This neighbor u must have $r_u \geq 1$. Thus u had to stop transmitting its value r_u due to a neighbor $w \in W_v$ with $r_w > r_u$. Due to Lemma 71 all marked nodes s reachable by a path of marked nodes from v must have $r_s = r_v$ and the same prefix as v, i.e.$(y_v^0, y_v^1, ..., y_v^{r_v - 1})$. Since $0 = r_v < r_w$ marked node v cannot reach w by a path of marked nodes. So assume $r_v \in [1, \log n - 1]$.

By definition of r_v there must exist a node $u \in N(v)$, s.t. $0 = y_v^{r_v} < y_u^{r_v} = 1$. Since nodes u and v differ in position r_v, we have $r_u \neq r_v$. Since r_v is marked, either $r_v > r_u$ or in case $r_v < r_u$ node u had to stop transmitting r_u due to a node w with $r_w > r_u$. Due to Lemma 71 all marked nodes s reachable by a path of marked nodes from v must have $r_s = r_v$, thus in case $r_v < r_u$ such a node w is not reachable from v. So assume $r_v > r_u$ and therefore for a node $u \in N(v)$ holds: $0 = y_v^{r_v} < y_u^{r_v} = 1$ $y_v^j = y_u^j$ for $0 \leq j < r_v$. Thus $ID_v < ID_u$. Because $ID_v < ID_u$ and $r_v > r_u$, this neighbor u must itself have a neighbor w with $ID_w > ID_u$ and $0 = y_u^{r_u} < y_w^{r_u} = 1$. Since v and u have the same prefix up to bit r_v, the node w cannot be a neighbor of any marked node s with value $r_s = r_v$, since otherwise $r_v \leq r_u$ because $0 = y_v^{r_u} = y_u^{r_u} < y_w^{r_u} = 1$, i.e. the prefix of w is larger than that of v.

Let us look at the k^{th} competition. Assume node v was competing in all previous competitions and was in particular not a marked node after the $(k-1)^{st}$ one. The arguments are similar to those of the first competition.

Assume $0 \leq r_v^k < \log^{(k)} n$ then the same reasoning applies as for the first competition – only the value for r_v has to be substituted by r_v^k and ID_v by r_v^{k-1}.

Assume $r_v^k = \log^{(k)} n$. Since v was not a marked or MIS node in competition $k-1$, there exists a neighbor $u \in N(v)$ with $r_u^{k-1} > r_v^{k-1}$. Neighbor u cannot participate in competition k, since $r_v^k < \log^{(k)} n$. Since v is a marked node, u must have become N_M or N_{MIS} in competition $k-1$ by a neighbor $w \in N(u)$. If w became a MIS node in round $k-1$, all neighbors $s \in N(w)$ became N_{MIS} in round $k-1$ as well. Thus w cannot be reached by a path of marked nodes from v. If w turned into a marked node in round $k-1$ and v in round k, then due to Lemma 70 w cannot be reached by a path of marked nodes from v. □

Lemma 73. *If a node v has become a marked node in the $f(2)^{th}$ phase, then it is in exactly one clique of M nodes after that phase.*

Proof. Denote the set of W_v for node v in phase i by W_v^i. Let a node w, as defined in Lemma 72, for phase i be denoted by $w_i \in W_v^i$. Lemma 72 implies that no neighbor $t \in N(w_i)$ can be a marked node reachable by a path of marked nodes from v. Thus due to Lemma 69 no neighbor t competes with v for the rest of the stage. This implies that $W_v^{i+1} \subseteq (W_v^i \setminus (N(w_i) \cup w_i))$. As a consequence nodes $w_i \in W_v^i$ and $w_j \in W_v^j$ with $i \neq j$ are independent. The size of a maximum independent set in $N^2(v)$ is upper bounded by $f(2)$. In every phase i, at least one node $w_i \in W_v^i$ at distance 2 from v is removed. Thus after at most $f(2)$ iterations, node v cannot reach any marked node at hop distance at least 2 by a path of marked nodes, i.e. a marked node forms a clique with all neighboring marked nodes. □

Lemma 74. *If a node v is still competing in the $(f(2)+1)^{st}$ phase then either v or a neighbor of v becomes a MIS node in that phase.*

Proof. Using Lemma 73 we have that each competitor v is in exactly one clique in the beginning of the $(f(2)+1)^{st}$ phase. The node v with largest ID_v of all *undecided* neighbors $u \in U \subseteq N(v)$ in the clique transmits its whole ID and have result $r_v = \log n$ and join the MIS. □

Lemma 75. *After stage s, a node v is either a MIS node or there exists a node that has become a MIS node in stage s within hop distance $f(2)+2$.*

Proof. We show that the distance between a marked node in phase i and node v is at most i. The proof is done by induction: Due to Lemma 68 after the first phase (line 12), every node v is a marked node M itself or adjacent to a marked node.

Assume the distance was at most $j-1$ after the $(j-1)^{st}$ phase (line 12). In the j^{th} phase only marked nodes become competitors again and participate in the competitions. Again, due to Lemma 68 after the j^{th} phase, every competitor becomes a marked node or a MIS node or have at least one of the two in its neighborhood. Thus the distance between a marked node and a N_M node grows at most by 1 per phase. Due to Lemma 74 every competitor or one of its neighbors must become a MIS node in the $(f(2)+1)^{st}$ phase. □

Lemma 76. *After at most $f(f(2)+2)$ stages, i.e. executions of the repeat loop (line 1 to 13), the computation of the MIS has finished.*

Proof. Consider a node v. Due to Lemma 75 a MIS node is chosen for v within distance $f(2)+2$ in each stage. Since MIS nodes are independent (Lemma 66), the number of MIS nodes within distance $f(2)+2$ is upper bounded by the size of a maximum independent set in $N^{f(2)+2}(v)$, which is $f(f(2)+2)$. This yields that at most $f(f(2)+2)$ stages are needed. □

Theorem 77. *The total time to compute a MIS is in $O(f(f(2)+2)\log n)$ $= O(\log n)$ and messages of 1 bit are sufficient.*

Proof. During the whole algorithm only mechanisms based on collision detection are used - nowhere it is assumed that the actual content of a message gets delivered. Therefore messages of size 1 bit are sufficient.

A phase requires $O(\log n)$ rounds of communication. There are $\log^* n+2$ competitions in one phase. The first one needs $O(\log n)$ rounds for transmitting the $\log n$ bits of the ID and $O(\log\log n)$ bits to transmit the result and update the state. The second requires only $O(\log\log n)$ bits a.s.o.

$$\sum_{i=1}^{\log^* n+2} \log^{(i)} n \leq \log n+\log\log n+...+2^{2^{2^2}}+2^{2^2}+2^2+2+1 \leq \log n\cdot(\sum_{i=0}^{\infty}\frac{1}{2^i} \leq 2\cdot\log n$$

Since only a constant number of phases,i.e. $f(2)+2$ and stages, i.e. $f(f(2)+2)$ are required (see Lemma 76) the overall number of communication is bound by $O(\log n)$.

□

MIS, asynchronous Wake-Up

Unfortunately, asynchronism introduces some difficulties. For instance, if a node wakes up and transmits without having any information about the state of its neighbors then it might disturb and corrupt an ongoing computation of a MIS. Therefore, all nodes inform their neighbors concurrently about their state and current activity. We guarantee a synchronous execution of Algorithm MIS without disturbance of woken-up nodes by using a schedule repeating after six rounds (see Algorithm Asynchronous MIS).

The idea is that nodes involved in a computation (or in a MIS) transmit periodically and thereby, force woken-up neighbors to wait. More precisely, upon wake-up a node listens until no neighbor has transmitted for 7 rounds. If a node has detected transmission for two consecutive rounds it knows that there is a neighbor in the MIS. A node executes Algorithm MIS by iterating the six round schedule as soon as it has not detected transmission for 7 rounds. A node transmits in the first round, if it executes or is about to execute Algorithm MIS (during round 3 of the schedule). This ensures that for a node v either a neighbor starts executing Algorithm MIS concurrently with v or it waits until v has completed the algorithm. In the second and fourth round no transmissions occur. A node transmits in the fifth and sixth, if it is in the MIS. The schedule is iterated endlessly in order that nodes in the MIS continuously inform woken-up neighbors about their presence. This prevents them from attempting to join the MIS.

Let t_{MIS} denote the time Algorithm (synchronous) MIS takes for computing a MIS when all nodes start synchronously.

Theorem 78. *Algorithm Asynchronous MIS computes a MIS in time $O(t_{MIS})$.*

Proof. If a set of nodes $U \subseteq V$ start Algorithm MIS synchronously and are not disturbed by any node $w \notin U$ interfering the computation then a correct MIS is computed (see Analysis Algorithm MIS). A node v computing a MIS transmits a message at least every six rounds, since a neighbor $u \in N(v)$ must not start Algorithm MIS if it detected transmission within seven rounds, it cannot start a computation if it woke-up while v is active. Consider an arbitrary pair u, v of neighboring nodes, e.g. $u \in N(v)$ that are awake but not executing Algorithm MIS. If node v has not detected a transmission for seven rounds, it starts transmitting a message periodically every six rounds and executes Algorithm MIS. Any neighbor of v, i.e. u, that does not start at the same time, detects a transmission from v and waits.

If a node v detects two consecutive transmissions a neighbor must be in the MIS and does not take part in any new computation of a MIS. In case, it detects transmissions (but non-consecutive) ones, some neighbor $u \in N(v)$ is executing Algorithm MIS. Thus within time $O(t_{MIS})$ a node $w \in (N(u) \cup u) \subseteq N^2(v)$ within distance 2 from v joins the MIS. Since the size of a maximum independent set within distance 2 is bounded by $f(2)$ (see Model Section) within time $O(f(2)t_{MIS}) = O(t_{MIS})$ node v is in the MIS or it has a neighbor in the MIS. □

Broadcast Algorithm

Our deterministic algorithm iterates the same procedure, i.e. the same schedule, using a fixed number of rounds. First, the current set of candidates (rounds 1 and 2) for forwarding the message is determined. A *candidate* is a node having the message and also having a neighbor lacking it. Second, some candidates are selected using a leader election algorithm, i.e. by computing a MIS. Finally, the chosen nodes transmit the message to all their neighbors without collision. If all nodes in the MIS transmitted the message concurrently, then no node might receive the message. This is because any node can be adjacent to more than one node in the MIS and suffer from a collision if all of them transmit concurrently. For that reason we must select subsets of the nodes in the MIS and let the nodes in each subset transmit in an assigned round. Explicitly constructing such sets is difficult in a distributed manner because a node in the MIS is unaware of the identities of the other nodes in the MIS. However, we can use the combinatorial tool of so called (n, k)-strongly selective families[35] of sets $\mathcal{F} = \{F_0, F_1, ..., F_{m-1}\}$

Schedule	State $s_v = Comp$	$s_v = HaveMsg$	$s_v = LackMsg$
Round 1	Transmit		Listen
2	Listen if not dT then exit		if dT then Trans.
3	Transmit	if not dT then $s_v := Comp$	Sleep
FOR i=1..t_{MIS}			
$3 + i$	Compute step i of Algorithm MIS	Sleep	
ENDFOR			
still round $t_{MIS} + 3$	If not in MIS then $s_v := HaveMsg$		
FOR i=1..$\|\mathcal{F}\|$			
$3 + t_M + i$	if $v \in F_i$ then Transmit msg	Sleep	if recv msg then $s_v := Comp$
ENDFOR			
still round $3 + t_M + \|\mathcal{F}\|$	$s_v = HaveMsg$		

Table 9.3: Algorithm DetBroadcast, where *dT* stands for (*has*) *detected transmission* and returns true, if a node has listened and detected a transmission.

with $F_i \subseteq V$, which yield a direct transmission schedule of length $|\mathcal{F}| = m$ for each node in the MIS, such that every node out of the given set of k nodes can transmit to all its neighbors without collision. A node v transmits in round i if $v \in F_i$.

An essential point for making fast progress is that we distinguish between nodes that have (just) received the message and never participated in a leader election and nodes that have the message and already did so. The former ones, i.e. new candidates, are preferred for forwarding the message, since, generally, they have more neighbors lacking the message.

More precisely, in our deterministic Algorithm DetBroadcast (see Table 9.3) a node lacking the message (state *LackMsg*) that receives the message immediately joins the computation of leaders, i.e. of a MIS, in the next execution of the schedule by switching to state *Comp*. After it has participated in the leader election once, it switches to state *HaveMsg* if it has not become a leader. If it has become a leader, i.e. is in the MIS, it transmits the message and exits. A node can only reattempt to become a leader, i.e. switch back to state *Comp*, in case no neighbor of it has just received the message, i.e. changed from state *LackMsg* to *Comp*.

Next, we show that all neighbors of a candidate get the message within logarithmic time.

Theorem 79. *Any candidate v ends the algorithm in time $O(\log n)$.*

Proof. For any node v at most $f(2)$ nodes $u \in N^2(v)$ are in state *Comp* in round $t_M + 3$, i.e. after the execution of the MIS algorithm. This follows from the correctness of the MIS algorithm and the definition of GBG. For the existence of a strongly related $(n, f(2))$ family of size $\mathrm{O}(f(2)^2 \log n)$, we refer to [35]. The time to compute a MIS is $\mathrm{O}(f(f(2)+2)\log n)$ as shown in Theorem 77. Thus one execution of the schedule takes time $O((f(f(2)+2)+f(2)^2)\log n)$.

Either a candidate v is computing a MIS itself or at least one neighbor $u \in N(v)$ does so. Assume neighbor u participates in computing a MIS S_0, joins the MIS and transmits. At least a subset of the neighbors $U \subseteq N(u)$ receives the message for the first time and any candidate $w \in U$ participates in the next computation of a MIS S_1. Assume at least one candidate $w \in U$ exists, i.e. $|U| > 0$, and a neighbor $x \in N(w)$ joins the MIS. Note that $x \notin N(u)$, since u transmitted the message to all its neighbors $w \in N(u)$. Therefore, all nodes $w \in N(u)$ have changed to state *HaveMsg* before the computation of S_1. Therefore, some node $x \in N(w) \setminus N(u)$ receives the message for the first time and participates in the next computation of a MIS S_2. Assume a node $y \in N(x) \cap x$ joins the MIS. This node y is not adjacent to u, i.e. $y \notin N(u)$, because no nodes $N(u) \cap N(x)$ participate together with y since all neighbors $N(x)$ changed to state *HaveMsg* before the computation of S_2. Thus node y is independent of all nodes in the MIS S_0 that transmitted and also any prior nodes that transmitted. Therefore node v gets a transmitting (independent) node with three computations of a MIS within distance 4. The maximum size of any independent set is bounded by $f(4)$ within distance 4. Therefore within time $\mathrm{O}((f(f(2)+2)+f(2)^2)f(4)\log n) = O(\log n)$ all neighbors of node v must have received the message and therefore node v cannot be a candidate any more. □

Theorem 80. *Algorithm DetBroadcast finishes in time $O(D \log n)$ for a GBG.*

Proof. Due to Theorem 79 any neighbor of a node having the message also receives it within time $O(\log n)$. Therefore, within time $O(D \log n)$ any node receives the message. □

9.5.2 Lower Bounds for MIS, Coloring and Broadcasting with Collision Detection

To begin with, we present two lower bounds. One showing that indeed $\Omega(\Delta)$ colors and time is needed to color a GBG and one that shows that for any Δ, i.e. also $\Delta \in O(1)$, time $\Omega(\log n)$ is needed even to make a successful transmission with high probability, i.e. $1 - 1/n$. The second lower bound implies a bound on the MIS and the same techniques imply an $\Omega(\log n)$ lower bound for broadcasting.

Theorem 81. *Any (possibly randomized) algorithm requires time $\Omega(\Delta)$ (in expectation) to compute a $\Delta+1$ coloring with high probability in a GBG (with or without collision detection).*

In the proof we use an argument based on information theory. Essentially, any node must figure out the identities of the nodes in its neighborhood. We show that the amount of possibly shared information about the neighborhood with $\Omega(\Delta)$ communication rounds is not sufficient to narrow down the options of distinct neighborhoods sufficiently.

Proof. Let the disconnected graph G consist of a clique C of Δ nodes and some other arbitrary subgraph such that no node $v \in C$ is adjacent to a node $u \notin C$. To color the clique C, any algorithm requires $\Delta + 1$ colors. We restrict the possible choices of $\binom{n}{\Delta+1}$ cliques as follows. We pick $(\Delta + 1)/2$ sets $S_0, S_1 ..., S_{(\Delta+1)/2-1}$, each consisting of 4 nodes, i.e. $|S_i| = 4$.(We assume that $4(\Delta + 1) \leq n$.) The algorithm gets told all the sets S_i and that out of every set S_i consisting of four nodes, two nodes are in the clique C. However, it is unknown to the algorithm which two nodes out of the four are actually chosen. The algorithm must reserve two of the $\Delta + 1$ colors for each set S_i, i.e. the (unknown) nodes in the set. Assume (without loss of generality) that the algorithm assigns colors $2i$ and $2i + 1$ to the chosen nodes of set S_i.

Assume an algorithm could compute a correct coloring in time Δ/c_0 for some constant $c_0 \geq 6$. Within Δ/c_0 rounds at most Δ/c_0 out of the $\Delta + 1$ nodes in the clique can transmit without collision. Assume that even if there is a collision due to some transmitters, say u, v, w, in a round i, all nodes in the clique receive one message of the same node, say all nodes receive v's message.(Note, that more information about the neighborhood can only benefit a node.) Additionally, any node can detect whether there was 0,1 or more than 1 transmitter in its neighborhood. For an upper bound assume a node $v \in C$ gets to know Δ/c_0 of its neighbors, i.e. receives one message of each of these Δ/c_0 nodes, and additionally, it receives two bits of information in each round, i.e. one of the values $\{0, 1, > 1\}$ can be encoded by two bits of information, e.g. bits 11 correspond to > 1 transmitters, bits 10 correspond to 1 transmitter and bits 00 correspond to none. Thus, in total a node v

gets to know at most $2\Delta/c_0$ bits. Observe that every node gets the same information, i.e. bits. The transmitted information is used to figure out, which two nodes of each set S_i are actually in the clique C in order to get a correct coloring. Since the algorithm is supposed to know already Δ/c_0 nodes, at least for $(1-1/c_0)\Delta/2$ leftover sets S_i no node of the set transmitted its identity. Therefore the algorithm can use the $2\Delta/c_0$ bits to figure out the identities of the $(1-1/c_0)\Delta$ nodes of the $(1-1/c_0)\Delta/2$ leftover sets S_i. Thus, any algorithm must decide on how many bits it spends on determining the two nodes out of the four possible in each set that are actually in its neighborhood. On average, it can use $\frac{(2\Delta/c_0)}{(1-1/c_0)\Delta/2} = 4/(c_0-1)$ bits per set. For $c_0 = 9$ for at least half all sets S_i the algorithm can use at most 1 bit. Assume set S_0 consists of nodes $\{a, b, c, d\}$. Any node in the set S_0 must make a decision, which of the two colors $\{0, 1\}$, it chooses based on its ID and a single bit. Assume node a decides in favor of color 1 given it received bit 0, i.e. $col(a|0) = 1$. Then all other nodes in the set must decide to pick color 0 if they receive bit 0, i.e. $col(b|0) = 0$, $col(c|0) = 0$ and $col(d|0) = 0$. If not, consider a node $x \in S_0$ that also decides in favor of color 1. In this case, if a and x are chosen to be in the clique C, both are adjacent and choose the same color. Thus the coloring is incorrect. Assume all nodes receive bit 1 and assume $col(a|1) = 0$ then $col(b|1) = 1$, $col(c|1) = 1$ and $col(d|1) = 1$. Thus, if out of the set S_0, nodes b, c are chosen then both decide on the same color, whatever the given bit is, i.e. they both pick color 0 if the given bit is 0 and color 1 if the bit is 1. If $col(a|1) = 1$ then $col(b|1) = 0$, $col(c|1) = 0$ and $col(d|1) = 0$ and nodes b, c decide on color 0 whatever the given bit is. Thus the coloring cannot be correct. Randomization can not increase the amount of exchanged information. Thus, in the end any algorithm must also decide on whether to choose color $\{0, 1\}$ based on a single bit. Through a case enumeration one can see that it is not possible to correctly guess the right colors with probability more than $1/4$. Assume $col(a|0) = 1$ with probability $p_{\geq \frac{1}{2}} \geq 1/2$. Then all other nodes in the set must decide to pick color 0 with probability $p_{\geq \frac{1}{2}}$ to have a chance higher than $1/4$ of a correct coloring. Using the same reasoning as for the deterministic case a maximum probability of $1/4$ for a correct coloring of a single set using only one bit follows. Since for at least half of all $(1-1/c_0)\Delta/2$ sets S_i with unknown nodes we can use at most one bit, we expect at least $3/4(1-1/c_0)\Delta/4$ to be colored incorrectly. □

Theorem 82. *There exists a graph such that for any $\Delta > 1$, any (possibly randomized) algorithm using collision detection requires time $\Omega(\log n)$ to compute a MIS (in expectation).*

Proof. Consider a (disconnected) graph where every node has degree 1, i.e. a single neighbor. Assume every node $v \in V$ knows that its degree in the

network is one but it is unaware of the identity of its neighbor u. Consider a sequence of $\frac{\log n}{8}$ rounds. For every node $v \in V$ we can calculate the probability that node v transmits in round $0 \leq i < \frac{\log n}{8}$ given that it has not yet received a message but transmitted itself or listened without detecting any transmission by its neighbor for rounds $0 \leq j < i$. There must exist a set U of $n^{7/8}$ nodes such that for every round i with $i \in [0, \frac{\log n}{8} - 1]$ all nodes in U transmit either with probability $p_{\geq \frac{1}{2}}$ at least $\frac{1}{2}$ or with probability $p_{< \frac{1}{2}}$ less than $\frac{1}{2}$, since a node has only two choices in each round (transmit or not). Thus, out of n nodes on average at least $n/2^{\frac{\log n}{8}} \geq n^{7/8}$ must decide to transmit (and listen) in the same rounds for all $\frac{\log n}{8}$ rounds. Consider an arbitrary pair $u, v \in U$ and assume they are adjacent. The chance of a transmission from u to v or the other way around is at most $(1-p_{< \frac{1}{2}}) \cdot p_{< \frac{1}{2}} + p_{\geq \frac{1}{2}} \cdot (1-p_{\geq \frac{1}{2}}) \leq \frac{1}{2}$ for one round, since any term $p \cdot (1-p)$ can be at most $1/4$. After all $\frac{\log n}{8}$ rounds the probability is at most $1 - \frac{1}{4^{\frac{\log n}{8}}} \leq 1 - \frac{1}{n^{1/4}}$. Assume, we randomly create $n^{7/8}/2$ pairs of nodes from the set U, such that the two nodes from each pair are adjacent. We expect for $n^{7/8}/2/n^{1/4} = n^{5/8}/2$ pairs that no message is exchanged. Thus the nodes from these pairs must make the decision, whether to join or not to join the MIS based on the same information. Let $U_1 \subseteq U$ be all nodes that decide to join the MIS with probability at least $1/2$ if they have never received a message, i.e. transmitted in the same rounds as their neighbor(s), and let all other nodes be in set $U_2 = U \setminus U_1$. Either U_1 or U_2 is of size at least $|U|/2$. Assume it holds for U_1. If we pick pair after pair then the probability that both nodes are taken from U_1 is at least $1/16$ for a pair independent of all previously picked pairs as long as at most $|U_1|/4 \geq |U|/8$ pairs have been chosen, i.e. for the remaining nodes in U_1 holds $|U_1| \geq |U|/4$. Thus, we expect $|U|/8/16 = |U|/128$ pairs to have both nodes in the same set U_1 or U_2. The chance that both join for a pair in U_1 is $p_{\geq \frac{1}{2}} p_{\geq \frac{1}{2}} \geq 1/4$ or none does for a pair in U_2 is $(1-p_{< \frac{1}{2}})(1-p_{< \frac{1}{2}}) \geq 1/4$. Thus, $1/4$ of all the pairs having transmitted in the same round and being from the same set U_1 or U_2 either both join the MIS or not in expectation. The probability that at least half of the expected $n^{5/8}/2/128/4 = n^{5/8}/1024$ pairs transmit in the same rounds, i.e. do not exchange a message, and both nodes from the pair join the MIS or both do not join is larger than $1 - 1/n^c$ for some arbitrary large constant c using a Chernoff bound. Thus the probability that the algorithm finishes in time less than $\log n/8$ is at most $1/n^c$.

The argument for the deterministic case is analogous, i.e. an equally large set (as in the randomized case) of nodes must transmit in the same rounds and it is not possible that all pairs correctly decide to join the MIS or not for all possible choices of neighborhoods. More precisely, consider three nodes u, v, w that all transmit in the same round (given their only neighbor is also

one of u, v, w). If in a graph G_1 u, v are adjacent then either u or v must join the MIS without having received a message from its neighbor, i.e. v is unaware whether its neighbor is u, v or w. Assume u joins the MIS then v cannot join. If in a graph G_2 v, w are adjacent then both transmit the same sequence and since v does not join w has to. Now if in a third graph u, w are adjacent then both join the MIS violating the independence condition of a MIS. □

Observe that the above theorem even holds for synchronous wake-up. Since one can compute a MIS from a coloring in constant time, the lower bound also holds for the MIS. More precisely, for the graph in the above proof with $\Delta = 1$ one can compute a MIS from a $\Delta + 1$ coloring by putting all nodes with color 0 in the MIS.

Corollary 83. *Any (possibly randomized) algorithm using collision detection requires time $\Omega(\log n)$ to compute a coloring in GBG (in expectation).*

For broadcasting with conditional wake-up a lower bound of D is trivial. A lower bound of $\Omega(\log n)$ for networks of diameter two can be proven using the same idea as for the proof of Theorem 82.

Theorem 84. *There exists a graph such that for any $\Delta > 1$, any (possibly randomized) algorithm using collision detection requires time $\Omega(\log n)$ to make a transmission among all nodes with probability $1 - \frac{1}{n}$.*

Proof. Assume the following network of diameter two. The source is adjacent to two nodes and these two nodes in turn are adjacent to all other nodes. Consider a sequence of $\frac{\log n}{2}$ rounds. For every node $v \in V$ and every round i we can calculate the probability that node v transmits in round $0 \leq i < \frac{\log n}{2}$ given that it has received the message (and possibly other information). There must exist two nodes u, v such that for all $\frac{\log n}{2}$ rounds either they both send with probability at least $\frac{1}{2}$ or less than $\frac{1}{2}$ given they received the same information, since any node has only these two options and we have that $n > 2^{\frac{\log n}{2}}$. Thus the chance of a successful transmission of either u or v to its neighbor is at most $\frac{1}{2}$ for one round and at most $1 - \frac{1}{2^{\frac{\log n}{2}}} = 1 - \frac{1}{\sqrt{n}}$ after $\frac{\log n}{2} \in \Omega(\log n)$ rounds. □

9.5.3 Lower Bound for Broadcasting without Collision Detection in GBG

The lower bounds for randomized [82] as well as for deterministic [70] algorithms for general graphs can be adapted to GBG. Both rely on constructing a graph with layers $L_0, L_1, ..., L_{\Omega(D)}$, where nodes L_i in layer i are independent and they are at distance i from the source, i.e. broadcast initiator. In

[70] the graph consists of two alternating layers consisting of a single node in layer i that is connected to all nodes L_{i+1} in layer $i+1$. Only a subset $W_{i+1} \subseteq L_{i+1}$ of the nodes in layer $i+1$ is connected to the single node in layer $i+2$. In the lower bound graph in [82] the nodes in layer i are connected to some nodes $W_{i+1} \subseteq L_{i+1}$ in layer L_{i-1} and L_{i+1}. There are no fixed layers of single nodes. The difficulty for the algorithm is figuring out the number of nodes in L_i. If it knows the number $|L_i|$ it can transmit with probability $1/|L_i|$, yielding an O(1) algorithm to get to the next layer. However, in [82] one could also use the same topology as in [70], i.e. every second layer consists only of a single node. Though this allowed the algorithm to pass every second layer in one round by transmitting with probability 1 for the other half of the layers the algorithm is unaware of the number of nodes in a layer.

For GBGs it is not possible for a node v in layer i to have an arbitrary number of independent nodes in layer $i+1$. Thus, we assume that all nodes in layer i form a clique. Therefore, a node in layer i knows all the successful transmissions that occurred in layer i. However, by elongating any protocol P by a factor of 4, the nodes in layer i in a general graph also know all successful transmissions in layer i. The idea is to let all layers with single nodes repeat their received message to the other layers. Since a single node is adjacent to two layers, it might face a collision if a node from each of its adjacent layers transmits concurrently and thus is not able to tell any layer if the transmission was successful. Thus, we repeat 8 rounds in a round robin fashion. Any node in layer i executes one step of the algorithm in round t if $2i \mod 8 = t \mod 8$.[10] Any node in layer i transmits in round t if $2i-1 \mod 8 = t \mod 8$ and if it received a message in the previous round. Thus a node in layer i having transmitted the message in round t knows that some node in layer i transmitted without collision, if and only if it detects transmission in round $t+1$.

Thus, we have arrived at the following proposition:

Proposition 85. *Any deterministic broadcasting algorithm takes time* $\Omega(n \log_{n/D} n)$ *and any randomized broadcasting algorithm takes time* $\Omega(D \log \frac{n}{D})$ *for GBG.*

The lower bound of $\Omega(\log^2 n)$ [2] cannot be extended in the same manner. The lower bound graph consists of two layers L_1 and L_2, where nodes within a layer are independent. In particular, layer L_2 consists of $\Omega(\log n)$ nodes. Each node l in L_2 is connected to some set H_l of nodes in L_1. There are $\Omega(\log n)$ sets H and in some round exactly one node from each set H_l must transmit in order that all nodes in L_2 receive the message. In GBG this is not the case, since in L_2 some nodes are adjacent, i.e. by definition of

[10] Note, that all nodes having the message are synchronized, i.e have global clocks, if the message includes the current time t.

a GBG at most $f(2)$ nodes in L_2 can be independent. Thus, a node in L_2 having received the message might forward it to other nodes in L_2 and it might be sufficient that for only $f(2)$ out of all $\Omega(\log n)$ sets H from L_1 a node transmit to its neighbor in layer L_2.

Algorithm FastColoring

Upon wake-up:

1: $TakenColors(v) := WaitFor(v) := \{\}$; $color_v := -1$

TryMIS():

2: $s_v^{MIS} := false$
3: Compute MIS[95] S^1 {Out: state s_v^{MIS}}
4: **if** $s_v^{MIS} = true$ **then**
5: **wait until** Broadcast(v,6, $InMIS(v)$,$\frac{1}{n^5}$) completed plus the duration of a Broadcast(.,6, .,$\frac{1}{n^5}$)
6: $IS^6(v) := \{u|$recv msg $InMIS(u)\}$
7: **wait until** Broadcast(v,7, $MIS6(v, IS^6(v))$,$\frac{1}{n^5}$) completed plus the duration of a Broadcast(.,7, .,$\frac{1}{n^5}$)
8: $IS^6(v) := IS^6(v) \cup \{u|$recv msg with $MIS6(u, IS^6(u)) \wedge (v \in IS^6(u))\}$ {Ensure symmetry of IS^6}
9: Calculate MIS [106] S^6 on $G' = (V' = \{u|u \in S\}, E' = \{\{u, v\}|u, v \in V', v \in IS^6(u)\})$ {Out: state s_v^{MIS6}}
10: **if** $s_v^{MIS6} = true$ **then**
11: Broadcast(v,8,$DoNotTryMIS(v)$,$\frac{1}{n^5}$)
12: Broadcast(v,3,$DoNotTransmit(v)$,$\frac{1}{n^5}$)
13: Transmit $GrantingColors$
14: GetColor() {see Table 9.2; Out: $color_v$}
15: Broadcast(v,9, $TransmitAndTryMIS(v)$,$\frac{1}{n^5}$)
16: **end if**
17: **end if**

At any time:

18: **if** recv msg $GrantingColors$ **then** GetColor(); Exit; **end if** {see Table 9.2; Out: $color_v$}
19: **if** $(s_v^{MIS6} = false) \wedge$ (recv msg $\in \{DoNotTransmit(u), DoNotTryMIS(u)\}$) **then** Stop executing TryMIS(); $s_v^{MIS} := false$;$WaitFor(v) := WaitFor(v) \cup msg$ **endif**
20: **if** recv msg $TransmitAndTryMIS(u)$ **then** $WaitFor(v) := WaitFor(v) \setminus \{DoNotTryMIS(u), DoNotTransmit(u)\}$ **end if**
21: **if** (recv msg $InMIS) \wedge (s_v^{MIS} = false)$ **then** Stop executing TryMIS() **end if**
22: **if** $(color_v = -1) \wedge (WaitFor(v) = \{\}) \wedge$ (no msg recv for the duration $constant \cdot \log \Delta \cdot \log n$ slots) **then** TryMIS() **end if** {duration bounds time from start of broadcast $InMIS$ until $DoNotTryMIS$ finished}

Algorithm MIS

For each node $v \in V$

State $s_v := \textit{undecided}$
for l:=1 to $f(f(2)+2)$ by 1 **do**
  for i:=0 to $f(2)$ by 1 **do**
    $r_v^0 := ID_v$
    for j:=1 to $\log^* n + 2$ by 1 **do**
      $r_v^j := \log^{(j)} n$
      for k:=0 to $\log^{(j)} n$ by 1 **do**
        if $s_v = \textit{undecided}$ **then**
          if (Bit k of $r_v^{j-1} = 1$) $\wedge (r_v^j = \log^{(j)} n)$ **then** transmit
          else if (Detected transmission) $\wedge (r_v^j = \log^{(j)} n)$ **then** $r_v^j := k$
          end if
        end if
      end for
      Update state s_v
    end for
    if $s_v = M$ **then** $s_v := \textit{undecided}$ **end if**
  end for
  if $s_v = N_M$ **then** $s_v := \textit{undecided}$ **end if**
end for

Algorithm Update State

For each node $v \in V$

if $(s_v = \textit{undecided}) \wedge (r_v^j = \log^{(j)} n)$ **then**
  if $j = 1$ **then**
    $s_v := MIS$
    Wait 1 round and transmit
  else
    $s_v := M$
    Transmit and wait 1 round
  end if
else
  if (Detected transmission) $\wedge (s_v = \textit{undecided})$ **then** $s_v := N_M$ **end if**
  if Detected transmission **then** $s_v := N_{MIS}$ **end if**
end if

Asynchronous MIS

Upon wake-up:

1: **Listen until** no transmission detected for 7 consecutive rounds
2: **if** ever detected transmission for 2 consecutive rounds **then** $s_v := N_{MIS}$ **else** $s_v := executing$; SixRoundSchedule() **end if**

SixRoundSchedule():

3: **loop forever**
4: **if** $s_v = executing$ **then** Transmit **else** Sleep **end if**
5: Sleep
6: **if** $s_v = executing$ **then** Execute 1 step in Algorithm MIS **else** Sleep **end if**
7: Sleep
8: **if** $s_v = MIS$ **then** Transmit twice **else** Sleep two rounds **end if**
9: **end loop**

Chapter 10

Practical Work

We give a low power data aggregation MAC protocol in Section 10.1 and a localization scheme for wireless networks in Section 10.2.

10.1 Message Position Modulation

10.1.1 Introduction

Low duty cycles, e.g. infrequent transmissions, is a must for Medium Access Control(MAC) for low power sensor networks. For many applications information, e.g. measured data, reaches the base station only after tens of seconds. Thus, a small variation of the data transmission time of a (sensing) node is generally of no concern. The base station itself usually has much larger storage, processing and energy means than a sensing node. Message Passing Modulation (MPM) exploits this asymmetry by keeping the base station awake and letting the neighbors of the base transmit some information essentially without energy consumption, i.e. through passed time between the earliest possible transmission and the actual transmission. MPM also results in less channel usage, since message-position modulated messages are shorter than ordinary messages.

Pulse-position modulation is a well-known signal modulation technique where m bits are encoded using 2^m time-shifts, yielding a transmission ratio per time shift of $m/2^m$. In (only) one of the m time-shifts (or time slots) a pulse, carrying itself little or no information, is transmitted, say in time-slot $x \in [0, 2^m - 1]$, which is decoded by the receiver as value x. We suggest using pulse-position modulation to code only parts of the data to transmit using time-shifts. The technique is sensitive to multipath effects for high data rates, i.e. short time-shifts, and clock drift since a signal running on a long path and

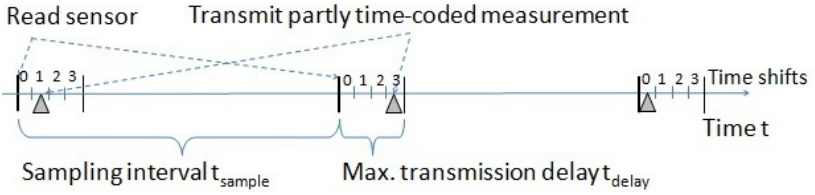


Figure 10.1: Basics of MPM

arriving at a later time-shift than intended can cause the decoding of wrong data. We avoid the danger of faulty data decoding due to multipath effects by using time-shifts that are long enough such that a signal must cover long distances. Therefore, it attenuates below the threshold for reception when reaching the receiver in the wrong shift. Differential position modulation, i.e. measuring the time between transmissions rather than absolute message arrival times at the base station, helps to deal with clock drift.

10.1.2 Message Passing Modulation Protocol

We describe all important parameters for MPM for a low power sensor network such as a network measuring temperature, humidity, light etc. using exemplary values (We discuss settings in general in Section 4). A node should read its sensor value every $t_{sample} = 100$ seconds. The transmission of a sample can be postponed by $t_{delay} = 4$ seconds.

To account for multipath effects we assume that the longest path a signal might travel and still be received takes $t_{multi} = 0.1$ms longer to follow than the shortest distance between base station and the sensing node. This corresponds to an additional length of at least 20 km. Typical sensor nodes can communicate up to a few kilometers given very good conditions (e.g. line of sight) and thus any signal that traveled an extra 30 km is too weak to be decoded.

We assume a clock drift d_{ppm} of about 45 ppm between the base and a sensing node. Given that data is transmitted reliably (i.e. the base station acknowledges each message and the sensing node retransmits if it misses an ACK), the longest time between two messages received by the base via MPM from a sensing node is in $[t_{sample}, t_{sample} + t_{delay}]$. If the system works in a best effort manner, i.e. lost messages mean data loss, the time between two received messages might be (much) larger, which increases the (maximum possible) clock drift. To deal with this problem and allow for regular resynchronization, a sensing node might demand an ACK from time

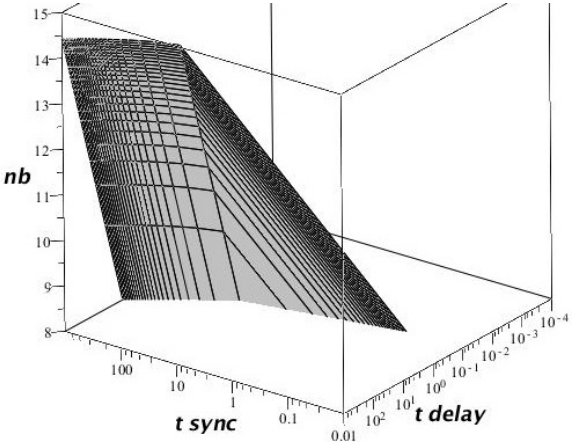


Figure 10.2: Number of saved bits n_b per transmission (of at least one byte) depending on the allowed (maximum) delay of a transmission (t_{delay} in [s]) and the maximum time between two message receptions t_{sync} (in [s]) by the base, i.e. the synchronization interval.

to time to make sure that the time between received messages is not too large. Assuming a maximal time span of $t_{sync} = t_{sample} + t_{delay}$ between two received transmissions from a sensing node implies that the clock of the base station and the sensing node drift up to $t_{drift} = (t_{sample} + t_{delay})/10^6 \cdot d_{ppm} = 4.7$ms for this time interval.[1] If the sensing node's clock shows time t then the clock of the base station might have any value from $[t - t_{drift}, t + t_{drift}]$. Additionally to the clock drift, the time between execution of the "send" command and the actual transmission might vary. For example, the delay might depend on the amount of data to copy to the radio buffer after the "send" command is called. We assumed a random transmission (plus reception) jitter, i.e. delay, of up to $t_{trans} = 2$ ms.

Therefore, the length of a time shift is given by $t_{shift} := t_{drift} + t_{multi} + t_{trans} = 6.8$ms. Using MPM we can code $n_b := \lfloor \log(t_{delay}/t_{shift}) \rfloor = \lfloor \log(4000/6.8) \rfloor = 9$ bits through time-coding per transmission. From the view of the base station the transmission by the sensing node should occur in the middle of a time-shift, i.e. with an offset of $t_{shift}/2$ to account for a positive or negative clock drift. The sensing node can compute the time

[1] The number of clock ticks for $t_{sync} = t_{sample} + t_{delay}$ is $t_{sync} \cdot f$, where f is the frequency of the oscillator. The number of drifted clock ticks is $t_{sync} \cdot f/10^6 \cdot d_{ppm}$, dividing by f yields the drift $t_{drift} := t_{sync}/10^6 \cdot d_{ppm}$.

shift t_{shift} to transmit as follows $t_{shift} := (n_b + \frac{1}{2}) \cdot t_{shift}$. Thus after the sample has been obtained and the time shift t_{shift} is computed the sensing node sleeps for time span t_{shift}, wakes up and transmits.

10.1.3 Evaluation

Our test network consisted of a base station, two sensing nodes and a jammer node, all positioned within one room. The most interesting test case is a system where data is transmitted in a best effort way (without ACK), since lost messages prolong the time between two receptions of a message. We intentionally used a jammer node close to the base station to create interference by transmitting randomly every 8 ms on average. Therefore, a longer time shift t_{shift} as in Section 10.1.2 of 14ms was used to correctly decode a message and still save 1 byte per transmission. The other parameters, i.e. $t_{sample} = 100$s and $t_{delay} = 4$s, are as in Section 10.1.2. For implementation the TinyNode 584 sensor platform running TinyOS 2.1.1 was taken. We only relied on standard TinyOS commands for transmission. TinyOS does not guarantee that the point of transmission is the same as the time when the send procedure was called due to other (long-running) tasks. This is usually of minor concern, since low-power networks generally do not perform long and heavy computations requiring lots of energy. We did not use any synchronization mechanism (e.g. [85]) or temperature clock drift compensation[105] but employed the milli seconds timer provided by TinyOS. Our implementation uses only little extra system resources, e.g. less than 20 bytes of RAM. The system was ran until 500 messages were received by the base station. In theory, due to multiple missed messages the synchronization might deteriorate and data might be decoded wrongly. Still, all received time coded data was decoded correctly, the maximum clock drift between two receptions of a message by the base was 5ms.

10.1.4 Discussion

The usefulness of MPM depends on the (worst case) drift between both clocks. A constant offset or slowly varying drift can be compensated to a large extent, e.g. by using temperature measuring capabilities of the node[105] or special algorithms, e.g.[85]. Either way, to maintain sufficient synchronization messages must reach the base station (or the sensing node) from time to time. For all applications, where the message loss is low or only a certain (or no) data loss is acceptable, this requirement is automatically fulfilled.

Figure 2 shows the time-coded bits per transmission $n_b := \lfloor \log(t_{delay}/t_{shift}) \rfloor$, where $t_{shift} := t_{drift} + t_{multi} + t_{trans} = t_{sync}/10^6 \cdot d_{ppm} + t_{multi} + t_{trans}$. The clock drift d_{ppm} is 45 ppm. The (unknown) random trans-

mission (plus reception) delay t_{trans}, depending on the hardware is 2ms. The time used for multipath compensation t_{multi} is 0.1ms.

Roughly speaking, the number of "saved" bits grows by 1 when doubling t_{delay} (up to at most t_{sync}) and decreases by 1 when doubling t_{sync} or the clock drift rate d_{ppm}. Thus, the number of "saved" bits is relatively insensitive to changes in the setup. In many scenarios it is straight forward to save one byte and with better clocks (or more advanced synchronization) it is not hard to save two bytes, but saving three bytes requires special hardware, e.g. a clock drift of less than one microsecond for time interval t_{sync}.

10.2 Three Plane Localization

10.2.1 Introduction

Localization in wireless networks is a well studied topic. Some solutions rely on special hardware, such as the geo positioning systems(GPS), other techniques simply work with the available capabilities of a standard wireless node, such as RSSI measurements or connectivity information[54, 111]. It is also common to have a non-homogenous network, i.e. anchor nodes with known positions and a set of simple nodes of unknown positions. Typically, measurements are noisy (in particular RSSI) and suffer from multipath effects and other issues. Thus, frequently, many measurements using distinct nodes are combined to compute an estimate, e.g. [54] checks for (each set of) three nodes whether a node is within the triangle. Our approach can be used to extend and improve free range and RSSI/LQI based measurement schemes by providing additional information, i.e. measurements.

Though the approaches regarding leveraging RSSI measurements and range free localization differ in many aspects, they all share the fact that only one node transmits at a time (such that collisions are avoided). Though this sounds natural, our idea is to intentionally let multiple nodes transmit concurrently to introduce collisions in the case of range free localization schemes or to create additional RSSI/LQI measurements due to overlapping signals. In the following we focus on free range localization.

10.2.2 Three Plane Localization Technique

For illustration consider a simple network consisting of two fixed nodes and a mobile node(such as in Scenario 1 in Figure 1). Assume that all nodes can communicate with each other and we want to (roughly) localize the mobile node M within the network with two nodes A and B of known positions. We do not measure RSSI/LQI but only rely on link quality. With ordinary range free localization techniques, i.e. only using connectivity information to

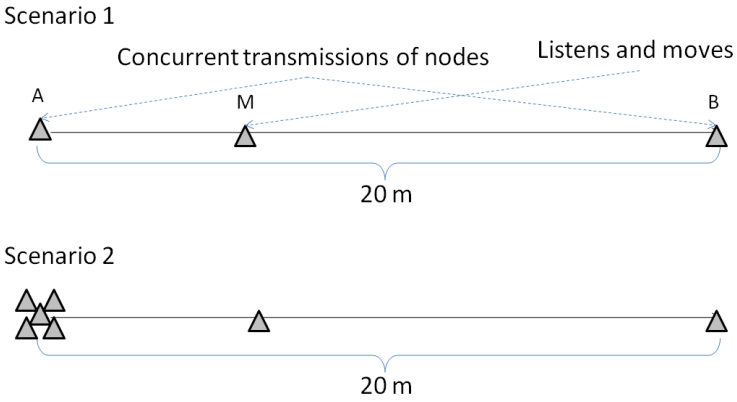


Figure 10.3: Evaluation scenarios

the two depicted nodes in Figure 1, it is only possible to say that node M is within communication range of A or B. Generally, it cannot be deduced whether node M is closer to A or B. To improve on that we explicitly schedule multiple concurrent transmissions. In case both nodes A and B transmit at the same time node M might receive nothing, the message sent by A or the message sent by B. Intuitively, node M receives the message from A, if it is close to A and nothing if it is around the middle of nodes A and B. Thus, when nodes A and B transmit concurrently (a few times), node M can simply count the number of received messages of each node and make an inference about its location based on the collected data. Signal strength decreases quickly with distance, i.e. in the SINR model it is assumed to decrease by the power of 2 to 6 of the distance to the sender. Thus, often (depending also on the hardware) the area where nothing is received is relatively small compared to the area where either a message from A or B is received. Therefore, node M can determine whether it is closer to A or closer to B or close to the middle, i.e. the entire plane is split into three parts. In general, for a large network we can let all pairs transmit. Each pair yields an area of which we know that node M is in it. The intersection of all planes gives a more accurate estimate of the position of node M than conventional free range localization. The idea of computing an intersection (of circles or triangles) has also been used (and studied extensively, e.g.[54]) to improve the quality of localization. Thus, we do not describe it here.

One can also let more than two nodes transmit concurrently in the hope to create additional planes beyond those when letting all pairs of nodes transmit. In principle, if several nodes are perfectly synchronized and transmit

exactly at the same time, i.e. interference is constructive, this is the case. Typical transmission frequencies are in the order of 1 Ghz, meaning that the synchronization must be in the nanoseconds range. However, with standard hardware this is a challenge. Let alone the difficulty of synchronization, due to the characteristics of signal strength, i.e. its fast fading with distance, the number of concurrently transmitting nodes must increase significantly to get planes that differ clearly from all planes created by node pairs. Let us go through an example. Assume, that node M is closest to nodes A and B, i.e. say it is at distance $0.4d$ from A and distance $0.6d$ from B (similar to Scenario 1 Figure 1). Assume that either A or B is received, i.e. if M is at distance less than $d(A,B)/2$ node A's message is received, otherwise B's message. Now, we want to have another plane at distance $0.4d$ from A to localize M more precisely by letting more nodes transmit concurrently. We compute an estimate how many (perfectly) synchronized nodes s additionally to B at distance at least $0.6d$ must transmit, such that M does not receive A any more. Assuming a constant power level P for all nodes and the standard SINR model with no noise, i.e. $\frac{P/(0.4d)^\alpha}{(s+1)P/(0.6d)^\alpha} \geq \beta$. Node M receives a message, if the condition is fulfilled, i.e. the ratio is larger than $\beta \geq 1$(We use $\beta = 1$). The parameter α is typically between 2 and 6, e.g. $\alpha = 2$ in free space (in vacuum) and six for larger distances (for nodes not extremely high above the ground). We use four, which accounts for the two ray propagation model. Thus, we get $\frac{P/(0.4d)^\alpha}{(s+1)P/(0.6d)^\alpha} = \frac{(0.6d)^4}{(s+1)(0.4d)^4} = \frac{(0.6)^4}{(s+1)(0.4)^4} \geq 1$. This yields that we need at least $s+1 = 5$ concurrently transmitting nodes to add a plane at distance $0.4d$. To add a plane at distance $0.25d$ would require 81 nodes! Furthermore, letting all kinds of subsets of nodes transmit concurrently is not feasible in practice.

10.2.3 Evaluation

It is essential to verify the theoretical study of three plane localization. Due to the environment, radio and antenna characteristics node M might receive only one of the two nodes A or B much beyond the middle. Additionally, due to multi path effects, messages might be received at unexpected locations leading to wrong position estimates. To check the validity of these potential drawbacks we considered two test scenarios depicted in Figure 1. We used TinyOS 2.1.1.

In the first scenario two TinyNodes 584 A, B are situated at distance about 20m from each other. A node M is moved between the two nodes. Node M always stays in line of sight of both A and B. The indoor hall environment contained some obstacles, e.g. tables, chairs, pillars. Still, without interference there was (very) good connectivity between A and B at least up to 25m distance. We walked from A to B and let both transmit concurrently

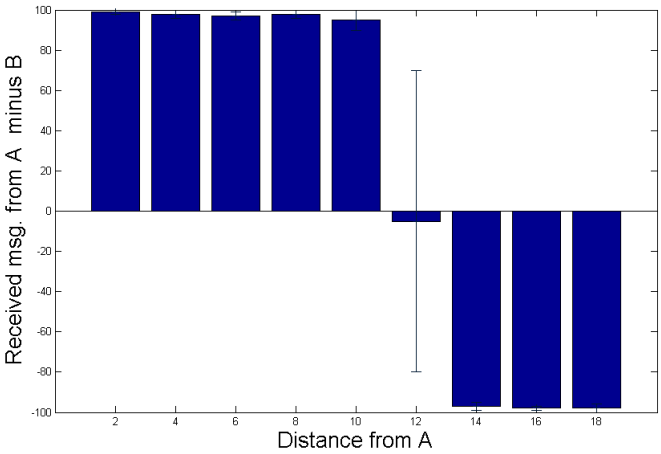


Figure 10.4: Node M moves from node A to node B. The plot shows the received messages by M from A subtracted by the received messages by M from B. The vertical lines correspond to the standard deviation.

for 100 times at several positions. We recorded the number of received messages from each node. This was repeated three times. There was a relatively sharp drop-off at about 12m (See Figure 2). The drop-off might be at 12m rather than in the middle, since node B was somewhat higher above the ground or due to the antenna (made of a simple wire) or due to the obstacles that hindered node A more than B. The gap of about 3 m between receiving "almost" all messages from A to almost all from B was not too far from expected, but the behavior within this gap was very unpredictable. Link quality was very sensitive to time (there are other testbeds in the building) as well as location and fluctuated sometimes from one extreme to the other. We also found that the TinyNode had problems in case of vibrations. Therefore, we held the node still when performing measurements. In the second scenario we tested the effect of multiple concurrent transmitting nodes, i.e. we put 4 more nodes next to A and let all five nodes transmitted concurrently. Our findings were similar to Figure 2, i.e. the gap was only a little shifted towards B and did not become much larger. The shift is likely to be attributed to the hardware and position of the individual nodes rather than to their interaction (i.e. constructive interference). For the TelosB platform we focused on scenario 1. We relied on the onboard antenna and used smaller

distances for A and B, i.e. 6m. The environment was harsher, i.e. an office room full of obstacles and frequent interference from WLAN. The results were qualitatively the same, but the observed variation in the number of received messages was higher and sensitive to location and time, e.g. even close to A (0.5-1.5 m away) sometimes node M received many messages from B. We attribute this mainly to multi-path effects and the different radio characteristics.

Chapter 11

Conclusions

It is an unfortunate truth that a vast amount of theory in wireless networks has little or no impact for the basic understanding of real wireless systems – not to mention an actual impact on current system design. To some extent this seems natural since research looks into the future, e.g., looks at networks not (yet) existing: large/ dense networks with 1000 of hops and nodes. However, if the number of wireless transmitters, e.g., wireless access points, grows at the current pace then we will not have to wait for long until such networks become reality. But, unfortunately, also inaccurate models heavily contribute to the fact that theory is often looked down upon by practioneers. For example, using general graphs for modeling wireless networks is very hard to be justified (except for certain special scenarios). Geometric graphs like the UDGs seem more appropriate. We showed that some lower bounds for general graphs also hold for UDGs. But the upper bounds for general graphs can be dramatically improved for UDG graphs. Additionally, assuming that two transmissions cause interference for sure is not correct. As we demonstrated in the practical part, if two devices A, B transmit concurrently, then a node C will generally suffer from interference only at a relatively small area out of the total area where either A or B can be received. This observation is not new and is captured (more or less) in the SINR model. The SINR model allows to describe behavior that are inconsistent with the standard UDG model. For example, assuming four nodes A,B,C and D on a line in the given order. Assume these nodes can transmit with varying power levels. If A and B transmit concurrently with the right power levels and all nodes are positioned at certain distances, it might be possible that node C receives the message from B while D receives the message from A. Given arbitrary large power levels the capacity of a network can be much larger than using a single power level. However, these days many devices feature only a fixed

transmission power level or power levels that vary to a relatively small extent. Since transmissions with high energy are not only energy inefficient but also require special hardware (if the energy level becomes too high) this might well remain this way. To increase the transmission range of wireless devices, techniques like beamforming (using an array of antennas creating constructive interference at a certain location) or directional antennas are more prominent than using a single omnidirectional antenna. The SINR model allows to account for the fact that several concurrently transmitting nodes might cause interference at a node far away from them. Still, it has been shown in [30] how to transform any graph based algorithm in the message passing model into the SINR model with little overhead, yielding basically the same time complexities as for graph based models for wireless networks. In fact, our coloring algorithm for the unstructured radio network model could be transformed the same way. Since the transmission power goes down rapidly with distance d, i.e. roughly d^2 for short distances and d^5 for larger distances with a few obstructions, the number of nodes that transmit far away from v but still cause an impact on v must be large. Thus, employing two radii – one radius giving the interference range of a node and a smaller radius giving transmission range of a node – should yield good approximations of real world behavior. This model is called the so called *protocol model* [52]. In fact, our line of thought has been confirmed [115].[1] It is not hard to extend our work to the protocol model. Thus, after all we believe that the UDG model (or its extensions like Quasi-UDG graphs or the protocol model[52]) are still very beneficial (and in some cases might be preferable to to the SINR model). However, the theory community has yet to derive models and algorithms that have an impact on practical system design.

[1] There is more work dealing with this issue, e.g. [84].

Part III

Shared Memory Systems

Chapter 12

Introduction

The era of computers with a single central processing unit (CPU) seems to end. Practically any new notebook or desktop computer is equipped at least with a dual-core processor. Quad-core chips are widely available and oct-core chips are already being tested. The hardware development is likely to continue at this pace. In contrast, a large fraction of the available software today is not designed to make full use out of these mighty processors. To be able to fully utilize them, software development must focus more on issues related to concurrent programming.[1]

Programming with multiple threads is demanding. It is considered difficult and time-consuming to write efficient and correct concurrent code. Major defects such as deadlocks or race conditions arise from locking shared resources (such as data objects) wrongly. As we discuss in Chapter 13 transactional memory is one approach to simplify concurrent programming. In particular, we deal with scheduling problems arising in transactional memory systems, i.e. in Chapters 14 and 15. Apart from general mechanisms to help writing concurrent code, for frequently reoccuring problems it makes sense to derive special algorithms using task specific knowledge that outperform automatically generated code which (typically) does not employ problem specific

[1] There are subtle differences between the terms involving "concurrent" and "parallel". Two threads run in parallel, if their executions overlap in time, i.e. are executed (mainly) at the same time. Thus, to have parallel execution, a computer must have at least two cores/CPUs. Two threads run "concurrently", if the order of (individual) instructions is not predetermined. For, example, if thread one consists of operations $A1$ and $B1$ and thread two of operations $A2$ and $B2$, then a concurrent execution might be in fact a sequential execution of all operations like $A1, B1, A2, B2$ or $A1, B2, A2, B1$ or $A2, A1, B1, B2$ etc. But concurrent execution also incoporates any parallel execution of (certain) instructions and is thus more general. However, in practice, exact parallel execution is not possible on modern computers. Using synchronizers (e.g. a barrier in the message passing interface), one might restrict the number of different interleavings of instructions of different threads.

information for optimization. A very common example of everyday software engineering are tasks related to manipulate data (efficiently), e.g. operations such as inserting, deleting, removing values from data structures (based on some condition). Another (though less prominent) example are graph based algorithms, i.e. one wants to solve a graph related problem one a multi-core system. We discuss techniques related for both examples. We discuss these two examples in Chapter 16.

Chapter 13

Basics of Transactional Memory

Transactional memory promises to make concurrent programming easier. Instead of specifying exactly when to lock and unlock each shared resource, a programmer only has to define a section of code as transaction. The transactional memory system should guarantee correctness and efficiency. Despite intensive research the success of transactional memory has yet to come. One of the main issues is performance. A key influence factor on the throughput of a transactional memory system is the behavior when conflicts arise. A conflict occurs when a transaction demands a resource that is in use by another transaction. The system can resolve the conflict by aborting the transaction holding (or demanding) the resource or by delaying the transaction requesting the resource. Wrong decisions have a dramatic influence on the overall performance, i.e. the system might abort a long running transaction just a split second before it commits, thereby wasting its entire work. The system might also delay a short transaction for a long time which itself uses a lot of resources and thereby blocks other transactions. Almost all transactional memory systems assign the task of resolving conflicts to a specific module called contention manager. Contention managers operate in a distributed fashion, that is to say, a separate instance of a contention manager is available for every thread, operating independently. In the past a number of policies have been proposed and evaluated, however, none of them has performed really well for all tested applications. The ones that do perform somewhat well, often lack progress guarantees and thus might suffer from starvation or livelock. Furthermore, often non-trivial tuning of the system and benchmark specific parameters are required to achieve good performance.

In Chapter 14 we derive several theoretical results for contention management.

In Chapter 15 we consider the general idea to improve performance by adapting the load of the system. Keeping some (computational) resources idle can can improve overall system performance in case these resources interfere in a negative manner, e.g. congest communication buses or abort each other in case of transactions.

13.1 Model and Definitions

A set of *transactions* $S_T := \{T_1, ..., T_n\}$ sharing up to s *resources* (such as memory cells) are executed on m *processors* $P_1, ..., P_m$.[1] For simplicity of the analysis we assume that a single processor runs one *thread* only, i.e., in total at most m threads are running concurrently. A thread running on processor P_i consists of a sequence of transactions $T_0^{P_i}, T_1^{P_i}, T_2^{P_i}, ...$ The sequence is executed sequentially on the same processor P_i, i.e., transaction $T_j^{P_i}$ is executed as soon as $T_{j-1}^{P_i}$ has completed, i.e. committed.

The *duration* of transaction T is denoted by t_T and refers to the time T executes until commit without contention (or equivalently, without interruption). The length of the longest transaction of a set S of transactions is denoted by $t_S^{max} := \max_{K \in S} t_K$. If an adversary can modify the duration of a transaction arbitrarily during the execution of the algorithm, the competitive ratio of any online algorithm is unbounded: Assume two transactions T_0 and T_1 face a conflict and an algorithm decides to let T_0 wait (or abort). The adversary could make the opposite decision and let T_0 proceed such that it commits at time t_0. Then it sets the execution time T_0 to infinity, i.e., $t_{T_0} = \infty$ after t_0. Since in the schedule produced by the online algorithm, transaction T_0 commits after t_0 its execution time is unbounded. Therefore, in the analysis we assume that t_T is fixed for all transactions T.[2] We consider an *oblivious adversary* that knows the (contention management) algorithm, but does not get to know the randomized choices of the algorithm before they take effect.

Each transaction consists of a sequence of operations. An operation can be a read or write access of a shared resource R or some arbitrary computation. A value written by a transaction T takes effect for other transactions only after T commits. A transaction either successfully finishes with a commit after executing all operations and acquiring all modified (written) resources or unsuccessfully with an abort anytime. A resource can be acquired either once it is used for the first time or at latest at commit time. A resource can be read in parallel by arbitrarily many transactions. A read of

[1] Transactions are sometimes called jobs, and machines are sometimes called cores.

[2] In case the running time depends on the state/value of the resources and therefore the duration varied by a factor of c, the guarantees for our algorithms (see Section 14.3.3) would worsen only by the same factor c.

transaction A of resource R is *visible*, if another transaction B accessing R after A is able to detect that A has already read R. We assume that all reads are visible. In fact, we prove in Section 14.2.4 that systems with invisible readers can be very slow. To perform a write, a resource must be acquired exclusively. Only one transaction at a time can hold a resource exclusively. This leads to the following types of *conflicts*: (i) Read-Write: A transaction B tries to write to a resource that is read by another transaction A. (ii) Write-Write: A transaction tries to write to a resource that is already held exclusively (written) by another transaction, (iii) Write-Read: A transaction tries to read a resource that is already held exclusively (write) by another transaction. A contention manager comes into play if a conflict occurs. It decides how to resolve the conflict by making a transaction wait (arbitrarily long), or abort, or assist the other transaction. We do not explicitly consider the third option. Helping requires that a transaction can be parallelized effectively itself, such that multiple processors can execute the same transaction in parallel with low coordination costs. In general, it is difficult to split a transaction into subtasks that can be executed in parallel. Consequently, state of the art systems do not employ helping. If a transaction gets aborted due to a conflict, it restores the values of all modified resources, frees its resources and restarts from scratch with its first operation. A transaction can request different resources in different executions or change the requested resource while waiting for another transaction.

Usually *conflicts* are handled in a *lazy* or *eager* way. We assume that conflicts are handled eagerly, i.e. once a transaction tries to access a resource that is held by another transaction. For lazy conflict handling a conflict is dealt with once a conflicting transaction tries to commit. Depending on the scenario, experimental evaluation showed that one or the other way leads to a shorter makespan. Even for "typical" cases neither consistently outperforms the other. A transaction keeps a resource locked until commit, i.e. *no early release*. By introducing additional writes in our examples, any transaction indeed cannot release its resources before commit.

A *schedule* shows for each processor P at any point in time whether it executes some transaction $T \in S_T$ or whether it is idle. The *makespan* of a schedule for a set of transactions S_T is defined as the duration from the start of the schedule until all transactions S_T have committed. We say a schedule for transactions S_T is *optimal*, if its makespan is minimum possible. We measure the quality of a contention manager in terms of the makespan. A contention manager is *optimal*, if it produces an optimal schedule for every set of transactions S_T.

13.2 Related Work

The idea to use transactions in the same manner as for database systems, i.e. to synchronize access to shared data, was developed more than 40 years ago [89]. Actual implementations came up about 20 years ago[114, 56]. In [58] Dynamic STM (DSTM) targeted towards dynamic data structures was described, which suggests the use of a contention manager as an independent module. Since then, many systems have been proposed and large advances have been made, e.g. DSTM2 [57] or NOrec [28].

Most proposed contention managers have been assessed by specific benchmarks only[104, 101, 49], and not analytically. The experiments yield best performance for randomized algorithms, which all leave a (small) chance for arbitrary large completion time. Apart from that, the choice of the best contention manager varies with the considered benchmark. Still, an algorithm called Polka [104] exhibits good overall performance for a variety of benchmarks and has been used successfully in various systems, e.g. [29, 101]. However, it did not do that well in our evaluation and also in [5]. In [101] an algorithm called SizeMatters is introduced, which gives higher priority to the transaction that has modified more (shared) memory. We show that from a worst-case perspective Polka and SizeMatters may perform exponentially worse than *RandomizedRounds*. Timestamping managers (e.g.[49, 104]) seem more robust and yield good results in a number of scenarios. The idea of mixing contention managers has been implemented in [50]. However, for low contention the introduced overhead might slow down the system considerably. The cost of monitoring the load distribution and costs of switching (and simply just having a mechanism to switch) among contention managers puts additional load on the system. It is computationally not feasible, i.e. it is NP-hard (as we show), to pick a contention manager that approximates the makespan just very coarsely. Nevertheless, switching contention managers seems a good idea, as supported by the evaluation for red-black trees which were good for high contention. Recently in [112] our algorithm and analysis *RandomizedRounds* was generalized to the case where several transactions are executed per thread/core.

In [118] the idea of load balancing based on contention has been investigated. A thread approximates the current contention based on the number of previous commits and aborts of transactions it has executed. Recent aborts and commits have a larger influence. When a transaction starts, it checks whether its contention approximation is beyond a threshold and resorts to a central scheduler that maintains a queue of transactions. The first element in the queue can execute until commit and is then removed. The evaluation was done using an HTM and an STM system on similar benchmarks as in this work, e.g. RBTree, LinkedList, LFUCache. With the load adaption scheme

in [118] as well as with our system the throughput is kept high, though (still) with some decrease with increasing threads.[3] Though the system of [118] was designed to be simple, our system is simpler, entirely decentral and does not rely heavily on setting parameters correctly. In the spirit of [118], in [33] a system was proposed that also serializes transactions. Initially, a central dispatcher assigns a transaction to a core. Each core contains a queue of transactions. Suppose a transaction A running on core 1 is aborted due to a transaction running on core 2. In this case transaction A is appended to the queue of 1 and the next transaction in the queue of core 1 is executed. Unfortunately, the results cannot be directly related to our system and [118], since the evaluation was done using a different benchmark. However, not surprisingly their implementation also outperforms the compared non-load adapting system. Still, a central instance sooner or later becomes a bottleneck.

The first analysis of a contention manager named Greedy was given in [51], using time stamps to decide in favor of older transactions. [51] guarantees that a transaction commits within bounded time and that the competitive ratio (i.e. the ratio of the makespan of the schedule defined by an online scheduler and by an optimal offline scheduler, knowing all transactions in advance) is $O(s^2)$, where s is the number of (shared) resources of all transactions *together*. The analysis was improved to $O(s)$ in [7]. In contrast to our contribution, access to a global clock or logical counter is needed for every transaction which clearly limits the possible parallelism with a growing number of cores. In [102] a scalable replacement for a global clock was presented using synchronized clocks. Unfortunately, these days most systems come without multiple clocks. Additionally, there are problems due to the drift of physical clocks.

Also in [7] a matching lower bound of $\Omega(s)$ for the competitive ratio of any (also randomized) algorithm is proven, where the adversary can alter resource requests of waiting transactions. We show that, more generally, if an adversary can reduce the possible parallelism (i.e., the number of concurrently running transactions) by a factor k, the competitive ratio is $\Omega(k)$ for deterministic algorithms and for randomized algorithms the expected ratio is $\Omega(\min\{k, \sqrt{m}\})$, where m is the number of parallel threads. In the analysis of [7] an adversary can change the required resources such that instead of $\Omega(s)$ transactions only $O(1)$ can run in parallel, i.e. all of a sudden $\Omega(s)$ transactions write to the same resource. Though, indeed the needed resources of transactions do vary over time, we believe that the reduction in parallelism is rarely that high. Dynamic data structure such as (balanced) trees and lists usually do not vary from one extreme to the other.

Furthermore, the complexity measure is not really satisfying, since the

[3] In [118], the benchmarks were only performed on a 2-way (dual core) SMP machine, whereas we tested on a 16 core machine.

number of (shared) resources in total is not correlated well to the actual conflicting transactions an individual transaction potentially encounters. As a concrete example, consider the classical dining philosophers problem, where there are n unit length transactions sharing n resources, such that transaction T_i demands resource R_i as well as $R_{(i+1) \bmod n}$ exclusively. An optimal schedule finishes in constant time $O(1)$ by first executing all even transactions and afterwards all odd transactions. The best achievable bound by any scheduling algorithm using the number of shared resources as complexity measure is only $O(n)$. Furthermore, with our more local complexity measure, we prove that for a wide variety of scheduling tasks, the guarantee for algorithm Greedy is linearly worse, whereas our randomized algorithm *RandomizedRounds* is only a factor $\log n$ off the optimal, with high probability.

We relate the problem of contention management to coloring, where a large amount of distributed algorithms are available in different models of communication and for different graphs [109]. Our algorithm *RandomizedRounds* essentially computes a $O(\max\{\Delta, \log n\})$ coloring for a graph with maximum degree Δ.

Contention management is related to online scheduling. In contrast to contention management, most scheduling algorithms are centralized and assume known conflicts. For illustration, in [38] the competitive ratios of scheduling algorithms are given for conflicting jobs. Their algorithms are non-distributed and on arrival of a new job J all conflicting jobs of J are known all at once, taking effect immediately, without change. Furthermore, the completion of a job cannot create new conflicts. In our model a conflict between two transactions happens when both access the same resource, which is not necessarily directly at their start. Additionally, dynamic data structures change their structure when modified and thus a transaction might access different resources due to the commit of another transaction, which might introduce new conflicts. Therfore, it is difficult to reliably predict conflicts, since they might change any time.

In [34] the effects of selfishness among programmers on the makespan is investigated for various contention managers from a game theoretic perspective.

Chapter 14

Contention Management Policies

14.1 Introduction

For a general introduction to transactional memory and contention management we refer to Chapter 13.

We show that even coarsely approximating the makespan of a schedule is a difficult task. (Informally, the makespan is the total time it takes to complete a set of transactions.) This holds even in the absence of an adversary. However, in case an adversary is able to modify resource requirements such that the number of conflicting transactions increases by a factor of k, the length of the schedule increases by a factor proportional to k. Second, we propose a complexity measure allowing more precise statements about the complexity of a contention management algorithm. Existing bounds on the makespan, for example, do not guarantee to be better than a sequential execution. However, we argue that since the complexity measure only depends on the number of (shared) resources overall, it does not capture the (local) nature of the problem well enough. In practice, the total number of (shared) resources may be large, though each single transaction might conflict with only a few other transactions. In other words, a lot of transactions can run in parallel, whereas the current measure only guarantees that one transaction runs at a time until commit. Third, we analyze widely used contention managers. For instance, some algorithms schedule certain sets of transactions badly, while others require all transactions – also those facing no conflicts – to modify a global counter or access a global clock. Thus, the amount of parallelism declines more and more with a growing number of cores. Fourth, we state and analyze two algorithms. Both refrain from using globally shared data. From a worst-case perspective, the randomized algorithm *RandomizedRounds* improves on existing contention managers drastically (exponen-

tially) if for each transaction the number of conflicting transactions is small. We also show that achieving a short makespan (from a worst case perspective) requires to detect and handle all conflicts early, i.e. for every conflict a contention manager must have the possibility to abort any of the conflicting transactions.

We start by giving lower bounds in Section 14.2 on the problem complexity. Then we discuss the sensitivity of contention management algorithms regarding to changes in the (number of) conflicts. Afterwards we prove that systems using invisible reads can become linearly slower than systems using visible reads. Finally, we analyze the competitive ratio of several well-known contention management algorithms.

In Section 14.3 we present a deterministic and a randomized contention management algorithm.

14.2 Lower Bounds

Before elaborating on the problem complexity of contention management, we introduce some notation related to graph theory and scheduling. We show that even coarse approximations are NP-hard to compute. We give a lower bound of $\Omega(n)$ for the competitive ratio of algorithms Polka, SizeMatters and Greedy, which holds even if resource requirements remain the same over time. We consider both eager and lazy conflict handling.

14.2.1 Notation

We use the notion of a *conflict graph* $G = (S, E)$ for a subset $S \subseteq S_T$ of transactions executing concurrently, and an edge between two conflicting transactions. The neighbors of transaction T in the conflict graph are denoted by N_T and represent all transactions that have a conflict with transaction T in G. The degree d_T of a transaction T in the graph corresponds to the number of neighbors in the graph, i.e., $d_T = |N_T|$. We have $d_T \leq |S| \leq \min\{m, n\}$, since at most m transactions can run in parallel, and since there are at most n transactions, i.e., $|S_T| = n$. The maximum degree Δ denotes the largest degree of a transaction, i.e., $\Delta := \max_{T \in S} d_T$. The term t_{N_T} denotes the total time it takes to execute all neighboring transactions of transaction T sequentially without contention, i.e., $t_{N_T} := \sum_{K \in N_T} t_K$. The time $t^+_{N_T}$ includes the execution of T, i.e., $t^+_{N_T} = t_{N_T} + t_T$. Note that the graph G is highly dynamic. It changes due to new or committed transactions or even after an abort of a transaction. Therefore, by d_T we refer to the maximum size of a neighborhood of transaction T that might arise in a conflict graph due to any sequence of aborts and commits. If the number of processors equals the number of transactions ($m = n$), all transactions can start concurrently. If,

additionally, the resource requirements of transactions stay the same, then the maximum degree d_T can only decrease due to commits. However, if the resource demands of transactions are altered by an adversary, new conflicts might be introduced and d_T might increase up to $|S_T|$.

14.2.2 Problem Complexity

If an adversary is allowed to change resources after an abort, such that all restarted transactions require the same resource R, then for all aborted transactions T we can have $d_T = \min\{m, n\}$. This means that no algorithm can do better than a sequential execution (see lower bound in [7]).

We show that even if the adversary can only choose the initial conflict graph and does not influence it afterwards, it is computationally hard to get a reasonable approximation of an optimal schedule. Even, if the whole conflict graph is known and fixed, the best approximation of the schedule obtainable in polynomial time can be exponentially worse than the optimal for certain graphs. The claim follows from a straight forward reduction to coloring.

Theorem 86. *If the optimal schedule requires time k, it is NP-hard to compute a schedule of makespan less than* $\max(k \cdot n^{1-1/\log^\epsilon n}, n)$ *(for any constant $\epsilon > 0$), even if the conflict graph is known and transactions do not change their resource requirements.*

Proof. Assume all accesses to resources are writes. There are n transactions of unit length, running on n processors, each transaction requires its resources on start up. Consider a coloring of the conflict graph $G = (S, E)$. Every set $C_i \subseteq S$ of transactions of color i forms an independent set (i.e., no nodes in C_i are neighbors) and thus all transactions in C_i can execute in parallel without facing any conflicts. The makespan of an optimal schedule is equal to the chromatic number $\chi(G)$, i.e., the minimum number of colors that is needed to color graph G. If this was not the case then the independent sets IS_i of the allegedly faster schedule of length l with $l < \chi(G)$ colors, formed a coloring with $C_i = IS_i$ and l colors. In [68] it was shown that computing an optimal coloring given complete knowledge of the graph is NP-hard. Even worse, computing an approximation within a factor of $n^{1-1/\log^\epsilon n}$ (for any constant $\epsilon > 0$ and $k \leq n^{1/\log^\epsilon n}$[1]) is NP-hard as well. □

As an approximation it seems reasonable to schedule transactions M, such that M is a maximum independent set (MaxIS, i.e an independent set of maximum cardinality) in $G = (S, E)$. Once all transactions in M have committed, the next MaxIS is scheduled. Iteratively scheduling a MaxIS

[1] In case $k \cdot n^{1-1/\log^\epsilon n} \geq n$ all transactions execute sequentially, thus the makespan is always at most n

yields a 4-approximation for the average response time or equivalently for the minimum sum of the transactions completion times [11]. Unfortunately, approximating the MaxIS problem within a factor of n^c for $c > 0$ is NP-hard [68]. Instead of a MaxIS one could try to schedule a maximal independent set (MIS, i.e., an independent set not extendable by adding a transaction). This yields a $O(\Delta \cdot t_S^{max})$ approximation. The factor t_S^{max} is a bound on how long it takes at most until the next MIS can be scheduled. So, how to obtain a MIS without any knowledge about the conflict graph? The well-known distributed algorithm by Luby [90] computes a MIS with high probability (i.e., $1 - \frac{1}{n}$) in time $O(t_S^{max} \cdot \log n)$. Unfortunately, it requires the degree of each transaction. Our Algorithm *RandomizedRounds* works for dynamic conflict graphs, does not need any information about them and can also be bounded by $O(t_S^{max} \cdot \log n)$. Thus, the total approximation ratio is $O(\Delta \cdot t_S^{max} \cdot \log n)$. In fact, for conflict graphs where no new edges (conflicts) are added, it can be improved to $O(\max\{\Delta, \log n\} \cdot t_S^{max})$.

14.2.3 Power of the Adversary

We show that if the conflict graph can be modified, the competitive ratio is proportional to the possible change of a transaction's degree. Initially, a contention manager is not aware of any conflicts. Thus, it is likely to schedule (many) conflicting transactions. All transactions that faced a conflict (and aborted) change their resources on the next restart and require the same resource. Thus, they must run sequentially. The contention manager might schedule transactions arbitrarily – in particular it might delay any transaction for an arbitrary amount of time (even before it executed the first time). The adversary has control of the initial transactions and can state how they are supposed to behave after an abort (i.e. if they should change their resource requirements). During the execution, it cannot alter its choices. Furthermore, we limit the power of the adversary as follows: Once the degree of a transaction T has increased by a factor of k, no new conflicts will be added for T, i.e. all initial proposals by the adversary for resource modifications augmenting the degree of T are ignored from then on.

Theorem 87. *If the conflict graph can be modified by an oblivious adversary such that the degree of any transaction is increased by a factor of k, any deterministic contention manager has competitive ratio $\Omega(k)$ and any randomized has $\Omega(\min\{k, \sqrt{m}\})$.*

Proof. We run m transactions on m parallel threads. In the initial conflict graph each transaction faces only one conflict and all transactions have the same duration t. Thus, we have $m/2$ pairs $\{U, T\} \subseteq S_T$ of conflicting transactions. For each pair $\{U, T\}$ both transactions read the same resource R_{UT}

on start-up and write it before their commits. Therefore, if two conflicting transactions start within time $t - \epsilon$ for some constant $\epsilon > 0$, both must have read R_{UT} and only one of them can commit while the other must abort. For every pair $\{U, T\}$ we can choose one transaction and let it change its resource demands after an abort, i.e. any (chosen) aborted transaction will write to resource R on startup until k transactions write to R. Thus, if k aborts take place, any schedule will be of length at least kt.

The scheduled transactions are known for a deterministic algorithm. Therefore, we can fix the transactions' resource requirements (before the start of the algorithm) such that (enough) aborts happen. Assume the algorithm schedules $x > 2$ transactions at at time. Since, the algorithm has no information about the conflicts, at least $x/3$ can be made to abort (in case three transactions are scheduled concurrently, two transactions can commit and one has to abort). We can set the aborted transactions, such that at least $\min\{x/3, k\}$ transactions write to the same resource R on startup. Thus, either a deterministic strategy schedules at most two transactions at a time or at least 1/3 of the transactions are aborted and therefore we can choose $\min\{m/3, k\}$ of them and let them write to the same resource R on startup. Therefore, the total time for a deterministic manager is $\min\{m/3, k\} \cdot t$. The optimal contention manager being aware of all conflicts finishes within time $2 \cdot t$.

Assume a randomized algorithm schedules a set $X > 4 \cdot \sqrt{m}$ of transactions at a time. Clearly, if the algorithm chooses transactions in a non-uniform manner, i.e. the chance that a pair $\{U, T\} \subseteq S_T$ is scheduled together is larger than a pair $\{V, T\} \subseteq S_T$ the adversary can make use of this knowledge. Thus, the algorithm is best off by treating all transactions equally. The chance that a transaction $T \in X$ does not face a conflict is given by $(1-1/m)^{|X|-1}$. The chance that none of the transactions in X is involved in a conflict is given by $(1-1/m)^{|X|-1} \cdot (1-1/m)^{|X|-2} \cdot (1-1/m)^{|X|-3} \cdot ... \cdot 1 = (1-1/m)^{\sum_{i=1}^{|X|-1} i} = (1-1/m)^{|X| \cdot (|X|-1)/2} \leq (1-1/m)^{8m} \leq 1/e^8$. Assume two transactions $\{U, T\}$ conflict. The algorithm must decide on one of the transactions to abort. Assume it aborts U with probability $p \geq 1/2$. Then the adversary lets U be the transaction that chooses resource R on startup. The overall chance that out of X transactions with $|X| > 4 \cdot \sqrt{m}$ one transaction aborts and chooses resource R on startup is $1/2 \cdot (1 - 1/e^8)$. If $k \cdot 4 \cdot \sqrt{m}$ transactions run in parallel then we expect at least a constant fraction of them to abort. Thus, either the algorithm schedules less than $4 \cdot \sqrt{m}$ at a time or we expect in total up to $\Omega(\min\{k, \sqrt{m}\})$ transactions to choose the same resource on startup.

□

14.2.4 Visible vs. Invisible Reads

We show: If an optimal contention manager is employed for a set of transactions, which do not alter their resource requirements over time, a system using visible reads can be linearly faster than a system with invisible reads. This is due to the fact that for invisible reads all aborts might take place without the influence of a contention manager, since read-write conflicts might not be handled by a contention manager, but can simply force a transaction to abort. It underlines the importance of detecting all conflicts and resolving them by a contention manager.

Theorem 88. *The competitive ratio of a system employing invisible reads is a factor $\Omega(n)$ worse than a system using visible reads, if both make use of an optimal contention manager.*

Proof. Suppose we have n processors and schedule $2 \cdot n$ transactions, i.e., transactions $T_0^{P_i}$ and $T_1^{P_i}$ on processor P_i. All transactions $T_0^{P_i}$ with $0 \leq i \leq n-1$ start at the same time, read resource R_i on startup and have duration $n + 2 \cdot i \cdot \epsilon$. Transactions $T_1^{P_i}$ with $0 \leq i \leq n-1$ write to all resources R_j $i < j \leq n-1$ on startup and have duration ϵ.

For invisible reads transaction $T_0^{P_0}$ commits after time n and $T_0^{P_1}$ after time $n+\epsilon$. A transaction $T_0^{P_i}$ with $1 \leq i \leq n-1$ will abort at time $n+2 \cdot i \cdot \epsilon$. After time $2 \cdot (n + 2 \cdot \epsilon)$ transaction $T_0^{P_2}$ commits and ϵ time units later $T_1^{P_2}$. Again all transactions $T_0^{P_i}$ with $2 \leq i \leq n-1$ abort. Thus, all transactions $T_0^{P_i}$ with $0 \leq i \leq n-1$ execute sequentially. The time it takes until all transactions have committed is lower bounded by $\Omega(n^2)$.

For visible reads, the contention manager decides to give all transactions $T_0^{P_i}$ with $0 \leq i \leq n-1$ higher priority. Afterwards all transactions $T_1^{P_i}$ with $0 \leq i \leq n-1$ execute sequentially. Therefore, the makespan equals $n + 3 \cdot n \cdot \epsilon = O(n)$. □

14.2.5 Competitive Ratio of Algorithm Greedy

The next theorem states that for certain problem instances algorithm Greedy [51] executes a large fraction of transactions entirely sequentially, even if a large amount of them could be run in parallel. In contrast to the lower bound in [7], our lower bound holds even if transactions do not modify their resource requirements after an abort (i.e. the adversary must not alter the demanded resources of a transaction). In algorithm Greedy each transaction gets a unique time stamp on start up and keeps it until commit. In case of a conflict, the older transaction proceeds. The younger aborts, if it has already acquired the resource needed by the older transaction, otherwise it waits. A waiting transaction is always aborted.

Theorem 89. *Algorithm Greedy [51] has competitive ratio of $\Omega(n)$ even if transactions do not alter their resource requests over time.*

Proof. Consider the dining philosophers problem (see Section 13.2) and assume eager conflict handling. Suppose all transactions have unit length and transaction i requires its first resource R_i at time 0 and its second $R_{(i+1) \bmod n}$ at time $1-i\cdot\epsilon$. Since the algorithm is deterministic, we know the time stamp of each transaction. Let transaction i have the i^{th} oldest time stamp. At time $1 - i \cdot \epsilon$ transaction $i + 1$ with $i \geq 1$ will get aborted by transaction i and only transaction 1 will commit at time 1. After every abort transaction i restarts ϵ time units before transaction $i-1$. Since transaction $i-1$ acquires its second resource $(i - 1) \cdot \epsilon$ time units before its termination, transaction $i-1$ will abort transaction i at least $i-1$ times. Thus, after $i-1$ aborts transaction i can commit. The total time until the algorithm is done is bounded by the time transaction n stays in the system, i.e., $\sum_{i=1}^{n}(1 - i \cdot \epsilon) = \Omega(n)$. An optimal schedule requires only $O(1)$ time.

For lazy conflict handling, we let transaction i have duration 1 and require its second resource at time $\frac{1}{2}$. Let all transactions start with their first operation at the same time. Just before commit every transaction i acquires resource i at the same time and also each transaction i with $i < n$ aborts transaction $i + 1$ concurrently. The transaction with the oldest time stamp commits. All other transactions start again at the same time and the process repeats. □

14.2.6 Competitive Ratio of Algorithm SizeMatters

SizeMatters[101] decides a conflict in favor of the transaction, that has accessed (read or written) more unique bytes, i.e. an access to a memory cell is only counted once during an execution. Thus, if the same byte is accessed multiple times, the overall increase of the priority is only 1. The priority is reset to 0 on restart. After a threshold c of restarts, it reverts to time-stamp. Unfortunately, the authors in do not explain how the time-stamps are chosen. Thus, we assume that a transaction running on processor P_i gets the i^{th} smallest time-stamp.

Theorem 90. *Algorithm SizeMatters [101] has competitive ratio of $\Omega(n)$ even if transactions do not alter their resource requests over time.*

Proof. We use the same transactions as in the proof of Theorem 89 and say that an access of resource R_i equals an access to $n - i$ bytes. The rest is analogous to the proof of Theorem 89. Transaction i will always have larger priority than $i + 1$, independent of whether the priority is calculated using time-stamps or the number of accessed bytes. □

14.2.7 Competitive Ratio of Algorithm Polka

Algorithm Polka works as follows: A transaction increases its priority by one for every acquired object until commit (it keeps its priority on abort). A transaction with higher priority can abort a lower priority one. If a transaction with lower priority wants a resource held by a transaction with higher priority, Polka waits for a number of intervals given by the difference in priority between the two conflicting transactions. The length of interval i has mean 2^i according to a fixed distribution chosen by the algorithm designer. For instance, assume transaction A wants a resource held by B and the difference in priorities is 2. After having tried to access the resource the first time, transaction A waits for a (random) time interval with mean 2^1. Then it tries to access the resource again. If it fails, it waits for a time interval with mean 2^2. If it was not able to access the resource again, transaction B is aborted, frees the resource and A can access it.

Theorem 91. *Algorithm Polka has at least competitive ratio* $\Omega(n)$.

Proof. Consider eager conflict handling and the probability that the back off time X_B is more than n time units. First, assume $p(X_B \geq n) \geq \frac{1}{n}$. Assume n transactions of unit length run on n processors. Each transaction i faces only one conflict on startup, i.e., transaction 1 with transaction 2, transaction 2 with transaction 3 etc. Therefore, directly after startup half the transactions will acquire a resource and they have priority 1, whereas the rest will wait for an interval of random length with mean 2. The probability that no transaction waits for n time units is $(1 - \frac{1}{n})^{\frac{n}{2}} \leq \frac{1}{\sqrt{e}}$. Therefore the expected schedule is at least $n \cdot (1 - \frac{1}{\sqrt{e}})$. An optimal schedule is of length 2. Now assume $p(X_B \geq n) < \frac{1}{n}$ and consider two transactions T_1, T_2 of length $3 \cdot n$. Let them start simultaneously and conflict after running for time n due to resource R. Assume transaction T_1 acquires resource R and thus it has priority 1. Transaction T_2 will wait in expectation for 2 time units before aborting T_1 and increasing its priority to 1. Once T_1 aborted it will conflict again after time n with T_2. Both will have priority 1 and T_1 aborts T_2 and sets its priority to 2. The process repeats: Again T_2 will execute for n time units and then wait in expectation for 2 time units. The chance that T_2 waits until T_1 completed, i.e., at least time n, is less than $\frac{1}{n}$. Therefore in expectation n trials of duration n are needed until transaction T_2 waits long enough. In total expected time $O(n^2)$ is needed. The optimal requires time $O(n)$.

For lazy conflict handling assume that there are two transactions of equal length starting at the same time. Transaction T_1 writes to resources R_1 and R_2. So does transaction T_2 but in the opposite order. Just before trying to

commit transaction T_1 acquires R_1 and at the same time transaction T_2 acquires R_2. Then T_1 aborts T_2 and concurrently T_2 aborts T_1, since both have the same priority and thus do not back off before aborting another transaction. Again, both start at the same time and the scenario repeats. Therefore the system will livelock and the competitive ratio becomes unbounded. □

14.3 Upper Bounds

Our first algorithm *CommitRounds* (Section 14.3.1) gives assertions for the response time of individual transactions, i.e., how long a transaction needs to commit. Although we refrain from using global data and we can still give guarantees on the makespan, the result is not satisfying from a performance point of view, since the worst-case bound on the makespan is not better than a sequential execution. Therefore we derive a randomized algorithm *RandomizedRounds* (Section 14.3.2) with better performance.

Algorithm Commit Rounds (*CommitRounds*)

On conflict of transaction T^{P_i} with transaction T^{P_j}:
$c_{P_i}^{max} := \max\{c_{P_i}^{max}, c_{P_j}^{max}\}$
$c_{P_j}^{max} := c_{P_i}^{max}$
if $c_{P_i} < c_{P_j} \vee (c_{P_i} = c_{P_j} \wedge P_i < P_j)$
then Abort transaction T^{P_j}
else Abort transaction T^{P_i}
end if

After commit of transaction T^P:
$c_P^{max} := c_P^{max} + 1$
$c_P := c_P^{max}$

14.3.1 Deterministic Algorithm *CommitRounds*

The idea of Algorithm *CommitRounds* is to assign priorities to processors, i.e. a transaction T^P running on a processor P inherits P's priority, which stays the same until the transaction committed. When T commits, P's priority is altered, such that any transaction K having had a conflict with transaction T will have higher priority than all following transactions running on P. Furthermore, transaction T will inform every transaction (more precisely, processor) with which T conflicts, that it should set its priority (after a commit) such that transaction K can abort it. To do so every processor P maintains two variables: (i) Variable c_P represents the priority, such that

the smaller c_P the higher transaction T's priority, and (ii) Variable c_P^{max} holds the next priority for a transaction running on processor P. In case a conflict occurs between transactions T^{P_i} and T^{P_j}, the transaction running on the processor P with smaller c_P proceeds. In case both processors have the same value ($c_{P_i} = c_{P_j}$), the transaction running on the processor with smaller identifier obtains the resource. The variable $c_P^{max} + 1$ is the next value for c_P, i.e., on commit we increment c_P^{max} and set $c_P := c_P^{max}$. The value of $c_{P_i}^{max}$ should be such that after a commit of a transaction running on P_i, the next transaction running on P_i should have lower priority than any transaction running on some processor P_j, that got previously aborted by the committed transaction executed on P_i, i.e. $c_{P_i} > c_{P_j}$. Thus, on every conflict we set $c_{P_i}^{max} := c_{P_j}^{max} := \max\{c_{P_i}^{max}, c_{P_j}^{max}\}$. Additionally, once the transaction running on P_i commits, we increment c_P^{max} and set $c_P := c_P^{max}$. For the first execution of the first transaction on processor P_i, the variable $c_{P_i}^{max}$ and c_{P_i} are initialized with 0.

Algorithm Randomized Rounds (*RandomizedRounds*)

procedure Abort(transaction T, K)
  Abort transaction K
  K waits for T to commit or abort before restarting
end procedure

On (re)start of transaction T:
  $x_T :=$ random integer in $[1, m]$

On conflict of transaction T with transaction K:
  if $x_T < x_K$ **then** Abort(T, K)
    else Abort(K, T)
  end if

14.3.2 Randomized Algorithm *RandomizedRounds*

For our randomized Algorithm *RandomizedRounds* a transaction chooses a discrete number uniformly at random in the interval $[1, m]$ on start up and after every abort. In case of a conflict the transaction with the smaller random number proceeds and the other aborts. The routine Abort(transaction T, K) aborts transaction K. Moreover, K must hold off on restarting until T committed or aborted.

To incorporate priorities set by a user, a transaction simply has to modify the interval from which its random number is chosen. For example, if one transaction chooses from $[1, \lfloor \frac{m}{2} \rfloor]$ instead of $[1, m]$, it doubles the chance of

succeeding in a round.

14.3.3 Proof of upper bounds

We study two classic efficiency measures of contention management algorithms, the makespan (the total time to complete a set of transactions) and the response time of the system (how long it takes for an individual transaction to commit).

Deterministic Algorithm *CommitRounds*

Theorem 92. *Any transaction will commit after being in the system for a duration of at most* $2 \cdot m \cdot t_{S_T}^{max}$.

Proof. When transaction T^{P_i} runs and faces a conflict with a transaction T^{P_j} having lower priority than T^{P_i} i.e., $c_{P_i} < c_{P_j}$ or $c_{P_i} = c_{P_j}$ and also $P_i < P_j$, then T^{P_j} will lose against T^{P_i}. If not, transaction T^{P_j} will have $c_{P_j}^{max} \geq c_{P_i}^{max} \geq c_{P_i}$ after winning the conflict. Thus, at latest after time $t_{S_T}^{max}$ one of the following two scenarios will have happened: The first is that T^{P_j} has committed and all transactions running on processor P_j later on will have $c_{P_j} > c_{P_j}^{max} \geq c_{P_i}^{max} \geq c_{P_i}$. The second is that T^{P_j} has had a conflict with another transaction T^{P_k} for which will also hold that $c_{P_k}^{max} \geq c_{P_i}^{max}$ after the conflict. After time $t_{S_T}^{max}$ either a processor has got to know $c_{P_i}^{max}$ (or a larger value) or committed knowing $c_{P_i}^{max}$ (or a larger value). In the worst-case one processor after the other gets to know $c_{P_i}^{max}$ within time $t_{S_T}^{max}$, taking time at most $m \cdot t_{S_T}^{max}$ and then all transactions commit one after the other, yielding the bound of $2 \cdot m \cdot t_{S_T}^{max}$. □

Randomized Algorithm *RandomizedRounds*

To analyze the response time, we use a complexity measure depending on local parameters, i.e., the neighborhood in the conflict graph (for definitions see Section 14.2.1).

Theorem 93. *The time span a transaction* T *needs from its first start until commit is* $O(d_T \cdot t_{N_T^+}^{max} \cdot \log n)$ *with probability* $1 - \frac{1}{n^4}$.

Proof. Consider an arbitrary conflict graph. The chance that for a transaction T no transaction $K \in N_T$ has the same random number given m discrete numbers are chosen from an interval $[1, m]$ is: $p(\nexists K \in N_T | x_K = x_T) = (1 - \frac{1}{m})^{d_T} \geq (1 - \frac{1}{m})^m \geq \frac{1}{e}$. We have $d_T \leq \min\{m, n\}$ (Section 14.2.1). The chances that x_T is at least as small as x_K of any transaction $K \in N_T$ is $\frac{1}{d_T+1}$. The chance that x_T is smallest among all its neighbors

is at least $\frac{1}{e\cdot(d_T+1)}$. If we conduct $y = 64 \cdot e \cdot (d_T + 1) \cdot \log n$ trials, each having success probability $\frac{1}{e\cdot(d_T+1)}$, then the probability that the number of successes X is less than $16 \cdot \log n$ becomes (using a Chernoff bound): $p(X < 16 \cdot \log n) < e^{-4\cdot\log n} = \frac{1}{n^4}$

The duration of a trial, i.e., the time until T can pick a new random number, is at most the time until the first conflict occurs, i.e., the duration t_T plus the time T has to wait after losing a conflict, which is at most $t^{max}_{N^+_T}$. The duration of a trial is bounded by $2 \cdot t^{max}_{N^+_T}$. □

Theorem 94. *If n transactions $S = \{T^{P_0}, ..., T^{P_n}\}$ run on n processors, then the makespan of the schedule by algorithm* RandomizedRounds *is $O(\max_{T\in S_T}(d_T \cdot t^{max}_{N^+_T}) \cdot \log n) + t_{last}$ with probability $1 - \frac{1}{n}$, where t_{last} is the time, when the latest transaction started to execute.*

Proof. Once all transactions are executing, we can use Theorem 93 to show that $p(\exists K \in S$ finishing after $O(\max_{T\in S}(d_T \cdot t^{max}_{N^+_T}) \cdot \log n) < \frac{1}{n}$. In the proof of Theorem 93, we showed that for the probability $p(E_T)$ of the event E_T for any transaction T holds: $p(E_T) = p(T$ finishes before $O(d_T \cdot t^{max}_{N^+_T} \cdot \log n) > 1 - \frac{1}{n^4}$. Since $O(d_T \cdot t^{max}_{N^+_T} \cdot \log n) \leq O(\max_{T\in S}(d_T \cdot t^{max}_{N^+_T}) \cdot \log n)$ we have $p(E_T) \geq p(T$ finishes before $O(\max_{T\in S}(d_T \cdot t^{max}_{N^+_T}) \cdot \log n) > 1 - \frac{1}{n^4}$. The probability $p(\text{not } E_{T_1}|E_{T_2})$ for two transactions T_1 and T_2 that given E_{T_2} has occurred, E_{T_1} does not happen, is at most $\frac{2}{n^4}$, since in the worst case E_{T_1} and E_{T_2} are maximal negatively correlated. That is to say, all outcomes for E_{T_1} that are excluded due to the occurrence E_{T_2} would cause E_{T_1} to occur. Since E_{T_2} only excludes outcomes of probability $1/n^4$ its impact on the probability of E_{T_1} is also only $1/n^4$. In the same manner, we have for $p(\text{not } E_{T_1}|E_{T_2} \wedge E_{T_3}) \leq 3/n^4$, since due to E_{T_2} a fraction $1/n^4$ of all possibilities for E_{T_1} to occur are excluded and due to E_{T_3} the same amount. In general for $p(\text{not } E_{T_1}|E_{T_2}\wedge...\wedge E_{T_n}) \leq 1/n^4+(n-1)/n^4 \leq 1/n^3$. Thus, the probability that no transaction out of all n transactions exceeds the bound of $O(\max_{T\in S}(d_T \cdot t^{max}_{N^+_T}) \cdot \log n)$ is $(1 - \frac{1}{n^3})^n \geq 1 - \frac{1}{n}$. □

The theorem shows that if an adversary can increase the maximum degree d_T by a factor of k the running time also increases by the same factor. The bound still holds if an adversary can keep the degree constantly at d_T despite committing transactions. In practice, the degree might also be kept at the same level due to new transactions entering the system. In case, we do not allow any conflicts to be added to the initial conflict graph, the bound of Theorem 94 (and also the one of Theorem 93) can be improved to $O(\max_{T\in S_T}(\max\{d_T, \log n\} \cdot t^{max}_{N^+_T}))$, with an analogous derivation as in

[108]. The idea of the proof is as follows: After time $d_T \cdot t^{max}_{N^+_T}$ we can show that the new maximum degree d'_T is at most $c \cdot d_T$ for a constant $c < 1$, i.e. it is reduced by a constant factor. This is because every transaction has constant probability to commit if it runs d_T times. To again reduce d'_T by a constant factor requires time $d'_T \cdot t^{max}_{N^+_T}$) time, i.e. a factor c less as before. Thus, the total time until the degree is less than one is given by $\sum_{i=0}^{O(\log n)} c^i d_T \cdot t^{max}_{N^+_T}) = O(\log n \cdot t^{max}_{N^+_T}))$.

Let us consider an example to get a better understanding of the bounds. Assume we have n transactions starting on n processors having equal length t. All transactions only need a constant amount of resources exclusively and each resource is only required by a constant number of transactions, i.e., d_T is a constant for all transactions T – as is the case in the dining philosophers problem mentioned in Section 13.2. Then the competitive ratio is $O(\log n)$, whereas it is $O(n)$ for the Greedy and SizeMatters algorithms (see Sections 14.2.5 and 14.2.6). For the Polka contention management strategy, the examples used in the proof of Theorem 91 disclose an exponential gap between *RandomizedRounds* and Polka, since the makespan of Algorithm *RandomizedRounds* for both examples is within a factor of $O(\log n)$ of the optimal with high probability.

A frequently used heuristic for contention management algorithms is to base the priority of a transaction on some measure of the work it has already completed. Since algorithm *RandomizedRounds* does not use any information about the progress of a transaction such as the number of accessed resources, it looks like *RandomizedRounds* does not follow this heuristic at all. However, we show that the probability that a transaction T has high priority increases with every transaction aborted due to T. Assume a set W of transactions has aborted due to T. Then the probability that the randomly chosen priority x_T is less than $a \in [1, m]$ is:

$$\begin{aligned} p(x_T \leq a) &= 1 - p(x_T > a) \\ &= 1 - p(x_K > a, \forall K \in (W \cup T)) \\ &= 1 - (1 - \frac{a}{m})^{|W|+1} \end{aligned}$$

This indicates that in general the more conflicts a transaction has won the higher are its chances to succeed in the next one as well.

Chapter 15

Load Adaption Policies

15.1 Introduction

For a broader introduction to transactional memory and contention management we refer to Chapter 13.

Frequently throughput peaks at a specific load. Thus, it might be better to leave some of the resources, such as network bandwidth or processor cores unused, to reduce the coordination effort and prevent harmful interaction, i.e. packet collisions and conflicts of transactions, respectively. The approach of load control based on the contention level has been investigated for transactional memory systems in prior work. However, as we shall see in the related work section, the attempts so far are complex and non-distributed. Any system relying on a central scheduler will eventually face scalability issues. In particular, complexity becomes an issue, if an actual implementation is to be carried out in hardware.

A requirement for our approach is to maintain scalability. Thus, for instance, any kind of central scheduler or dispatcher is not acceptable. However, a thread may collect information about its (executed) transactions and furthermore, if two transactions of two different threads conflict, i.e. share data, then also the aggregated information of each thread might be shared (until one of the transactions commits). We investigate two different schemes from both a practical and theoretical perspective. The first scheme uses an exponential backoff based on the number of aborts, i.e. before a restart a transaction must wait a timespan exponential in the number of its aborts. In the second scheme, load adaption is performed through delaying a transaction until some other (conflicting) transaction(s) have committed.

Apart from our experimental evaluation we also compare different strategies through analysis. In principle any strategy chooses a subset of all avail-

able transactions to run, postponing the others. Clearly, a good strategy guesses/predicts a large set of non-conflicting transactions and ideally, only delays those that would face a conflict anyway. Thus, a strategy might assume conflicts among transactions it has not observed. In general the effectiveness of a strategy depends on the topology of the so called *conflict graph*, which has transactions as nodes and edges between nodes, where a conflict might occur. We look at conflict graphs of typical benchmarks such as linked lists and analyze the throughput of several strategies. If the load adapting strategy correctly guesses the conflicts (or equivalently the topology of the graph) then the expected speed-up can be a constant. However, if, for instance, a dense conflict graph is assumed, but the true conflict graph is sparse, the worst-case behavior can be exponentially slower compared to a scheme without load adaption.

The chapter is split into a theory section discussing different algorithms for different load levels of the system and a section performing a practical investigation through benchmarks for various load adaption and contention management schemes.

15.2 Theoretical Analysis

15.2.1 Load Adapting Approaches

The first approach uses an exponential backoff scheme named *AbortBackoff*. A transaction maintains an abort counter, which is 0 initially and is incremented after every abort. The counter is used as a transaction's priority. For any conflict one transaction is aborted (i.e. no waiting). In case two transactions with the same priority conflict, an arbitrary one proceeds. If a transaction is aborted, it increments its waiting exponent i and waits for a random time interval in $[0,2^i]$. The second approach deals with different variants of (deterministically) serializing conflicting transactions. If two transactions are serialized, then at no future point in time they will run in parallel. A transaction keeps track of a set of (possibly) conflicting transactions. The set contains all transactions with which it has ever faced a conflict or with which it assumes to have a conflict. After an abort, a transaction is only allowed to restart if none of its conflicting transactions is executing. If a transaction C, having blocked two transactions A with conflict set $\{B, C\}$ and B with set $\{A, C\}$, then either A or B might restart. We choose each with probability $\frac{1}{2}$. In general, if C has blocked x transactions, one of the waiting transactions restarts. Each has a chance of $\frac{1}{x}$ to be selected. In our first conservative policy *SerializeFacedConflicts* a transaction only assumes to conflict with a job it actually had a conflict with. In our second policy *SerializeAllHopConflicts*, a transaction A having faced a conflict with B

assumes that it also conflicts with all transactions in the conflict set of B. The set of conflicting transactions is always kept up to date, if transaction A conflicted with B and later B with C, then C will also be in A's set of conflicting transactions.

We consider the above strategies as well as the naive strategy where a transaction restarts without delay. For the analysis it is crucial what type of contention management strategy is used. We employ two contention management strategies covering a wide set of available managers. In the first strategy, each transaction has a random priority in $[1, n]$ such that no transactions get the same priority. A transaction keeps the same priority until commit. In the second strategy, a transaction's priority equals the net-executing time of a transaction. The first scheme covers all contention managers, where the priority calculation is done in a way that is not (or weakly) related to the actual work performed by a transaction. The second scheme is a representative of a contention manager estimating a transaction's work.

Unfortunately, the conflict graph is dynamic and depends on many factors such as what resources are needed by a transaction and from what time on the resources are needed etc. Due to these difficulties we focus on extreme cases of conflict graphs. In the first scenario all transactions want the same resource, i.e. the conflict graph is a clique. We model a shared counter, where we assume that a transaction is very short and all transactions attempt to access the resource concurrently. We also consider a linked list and look at the expected length of the schedule. In the second scenario, all transactions want distinct resources, i.e. the conflict graph is a tree and thus sparse.

15.2.2 Moderate Parallelism – Conflict on Start-up

Assume that all n transactions start at the same time and want to write to the same memory cell directly after their start. Thus the conflict graph is a clique. Such a situation occurs, for instance, if multiple transactions want to concurrently increment a shared counter. Our primary concern is the expected delay due to transactions with low priority holding resources also wanted by transactions of higher priority. This happens since all transactions access the resource concurrently and have the same chance to acquire it – independent of their priority. The transactions of higher priority are delayed since they must abort the resource holding transaction. We assume that it takes 1 time unit to abort a transaction and to acquire its resource. Furthermore, if multiple transactions try to get a resource concurrently, a random one gets it.

Proposition 95. *For immediate restart the expected time span until all transactions committed is* $n \cdot t_T + \Omega(n \cdot \log n)$.

Proof. Assume that the resource is available (i.e. not accessed) and x transactions try to access it, then a random transaction T gets it. Once T got the resource, all transactions face a conflict with T. If T does not have highest priority, it gets aborted by some (random) transaction U with higher priority. Again transaction U is aborted, if its priority is not highest. The expected delay until the resource with highest priority obtained the resource can be computed through a recursive formula. The expected delay for one transaction is 0. The expected delay for two transactions is $\frac{1}{2}$, since we assumed that an unsuccessful try to acquire a resource delays a transaction by 1 time unit and the probability that the transaction with smaller priority gets the resource is $\frac{1}{2}$. Given x transactions the expected delay until the transaction with largest priority has the resource can be computed through the following recursion:

$$E[x] = (1 - \frac{1}{x}) + \frac{1}{x-1} \sum_{i=1}^{x-1} E[i]$$

The first term $1 - \frac{1}{x}$ denotes the chance that a transaction not having highest priority gets the resource given x transactions try. The second term states that any of the $x-1$ remaining transactions (of higher priority) has the same chance to be chosen. Assume $E[x] \geq \ln x$.

$$E[x] = (1 - \frac{1}{x}) + \frac{1}{x-1} \sum_{i=1}^{x-1} \ln i = (1 - \frac{1}{x}) + \frac{1}{x-1} \ln((x-1)!)$$

Using Stirling's Formula we get (for large x):

$$E[x] = (1 - \frac{1}{x}) + \frac{1}{x-1} \ln((x-1)!)$$

$$= (1 - \frac{1}{x}) + \frac{1}{x-1} \ln((\frac{x-1}{e})^{x-1} \cdot c_0 \cdot \sqrt{x-1})$$

$$= 1 - \frac{1}{x} + \ln \frac{x-1}{e} + \frac{\ln(c_0 \cdot \sqrt{x-1})}{x-1} - \frac{1}{x}$$

$$= \ln(x-1) + \frac{\ln(x-1)}{4 \cdot (x-1)} \geq \ln x$$

The last step can be seen as follows. We have that $\frac{\ln(x-1)}{4 \cdot (x-1)} - \frac{1}{x} > \frac{1}{x}$ for large x and also since $\ln x$ is concave, it is bounded by a tangent at an arbitrary position. In particular, $\ln x \leq \ln(x-1) + \frac{d(\ln(x))}{dx} \cdot 1 = \ln(x-1) + \frac{1}{x}$. Assume $E[x] \geq 2 \cdot \ln x$. The derivation is analog to the previous case.

$$E[x] = (1 - \frac{1}{x}) + \frac{1}{x-1} \sum_{i=1}^{x-1} 2 \cdot \ln i$$

$$= 1 - \frac{1}{x} + 2 \cdot \ln \frac{x-1}{e} + 2 \cdot \frac{\ln(c_0 \cdot \sqrt{x-1})}{x-1} - \frac{1}{x}$$

$$= 2 \cdot \ln(x-1) - 1 + \frac{\ln(x-1)}{2 \cdot (x-1)} - \frac{1}{x} \leq 2 \cdot \ln x$$

The last step can be seen as follows. We have that $\frac{\ln(x-1)}{2\cdot(x-1)} - \frac{1}{x} < 1$. Thus we have $\ln x \leq E[x] \leq 2 \cdot \ln x$. Initially, n transactions try to access the resource. After the first commit there are $n-1$ left a.s.o. Therefore the total time until all transactions committed is bounded by $\sum_{i=1}^{n} \ln i$. Using Stirling's Formula (for large n) as before, we have $\sum_{i=1}^{n} \ln i = \ln n! = O(n \cdot \log n)$. □

Proposition 96. *For* AbortBackoff *the expected time span until all transactions committed is* $n \cdot t_T \cdot 2^{O(\sqrt{\log n})}$.

Proof. Assume we have n jobs left in the system and all have priority (at least) $\log(8n \cdot t_T)$, i.e. each job aborted at least $\log(8n \cdot t_T)$ times. This must be the case after time $8nt_T$, since a transaction with priority i waits at most time 2^i before it restarts and the priority i corresponds to the number of aborts, thus the total time for $\log(8n \cdot t_T)$ aborts is given by $\sum_{i=0}^{\log(8n \cdot t_T)-1} 2^i \leq 8nt_T$.

If a transaction does not face a conflict, it commits. A single transaction A takes time t_T and if another transaction B starts either while A is running or $t_T - \epsilon$ (for some $\epsilon > 0$) before A then A and B face a conflict. Thus, if each transaction runs once in an interval of duration $4nt_T$, $n-1$ transactions together occupy at most an interval of length $2(n-1) \cdot t_T$. In other words a transaction has a chance of less than $1/2$ to face a conflict, if it starts at an arbitrary point in time during this interval. If instead of all $n-1$ transactions only a fraction a of all n transactions is active, the probability for a transaction to face a conflict becomes $a2n \cdot t_T/(4n \cdot t_T) = a/2$. More generally, after $\log(8n \cdot t_T) + x$ aborts, i.e. for a waiting interval of length $4n \cdot t_T \cdot 2^x$ and a fraction a of active nodes, the chance to abort becomes $a/2/2^x < a/2^x$. Assume that for $x = 0$, i.e. up to $\log(8n \cdot t_T)$ aborts all nodes are in the system, i.e. $a(0) = 1$. Then, using the above relation $a/2^x$ for an interval $[8nt2^{x-1}, 8nt2^x]$ we have $a(1) = a(0)/2$, $a(2) = a(1)/2^2$, a.s.o. remain in the system in expectation. More generally, $a(x+1) \leq a(x)/2^{x+1}$. As long as $a(x) \geq c_0 \cdot \log n/n$ for arbitrary constant c_0, using a Chernoff bound the probability that $a(x+1) \leq a(x) \cdot (3/4)^{x+1}$ is at least $1 - 1/n^{c_1}$ for some constant $c_1(c_0)$. Thus, $a(x+1) \leq (3/4)^{\sum_{i=1}^{x} i} = 1/2^{O(x^2)}$. With $x = O(\sqrt{\log n})$ we have that $a(x) < c_0 \log n/n$ less than $c_0 \log n$ transactions are active, i.e. all others committed. For the remaining transactions the chance to abort within a time interval $[2^{x-1}8nt, 2^x 8nt]$ with $x \in 2^{O(\sqrt{\log n})}$ is $\frac{\log n \cdot t_T}{n \cdot t_T \cdot 2^{O(\sqrt{\log n})}} \leq 1/2^{O(\sqrt{\log n})}$. Thus when looking at $O(\sqrt{\log n})$ additional in-

tervals the chance becomes $(1/2^{O(\sqrt{\log n})})^{O(\sqrt{\log n})} < 1/n^{c_2}$ for some arbitrary constant c_2. □

Proposition 97. *For the* SerializeAllHopConflicts *policy the expected time span until all transactions committed is* $n \cdot t_T + 1$.

Proof. Initially, all n transactions try to access the available resource and a random transaction T gets it. All transactions conflict with T and also assume that they conflict with each other. Therefore from then onwards, no conflicts occur. □

Proposition 98. *For* SerializeFacedConflicts *the expected time span until all transactions committed is* $n \cdot t_T + \Theta(n)$.

Proof. Initially, all n transactions try to access the (available) resource and a random transaction T gets it. All transactions conflict with T and add T to their sets of conflicting transactions. If T runs again it does not face a conflict. Therefore after n aborts all jobs have all others in their conflict set and the delay is $O(n)$. Given n transactions try to access a resource we expect $\log n$ aborts to happen before the transaction having highest priority (and thus running until commit) obtains the resource (see proof of Proposition 95). Thus, after $\frac{n}{c}$ aborts for some constant c, we expect (still) only $\frac{n}{\log n \cdot c}$ commits and the expected delay is $\Omega(n)$. □

15.2.3 Moderate Parallelism – Conflict at Arbitrary Time

Consider a typical benchmark such as a linked list where transactions either insert, delete or find a value in a list or traverse the list to compute some (aggregate) value. Each transaction keeps the entire read set until it committed, i.e. a transaction A considers a (long) traversed object O as read and conflicts with any transaction B modifying O. In some cases non-written objects might be releasable before commit, but this depends on the semantics of the transaction and, generally, has to be specified by the programmer. Therefore, it would make life for complex for the software engineer and we do not consider it. We focus on operations like inserts and deletes, i.e. after an arbitrary number of read operations an object is modified. Thus, the dense conflict graph is dense, since all transactions potentially conflict with each other. Still, a potential conflict might not necessarily occur. For example, consider two transactions A and B, both performing some write operation towards the end of the list. If A has started way ahead of B, then at the time A is committing, B will still be traversing the list and will not have accessed the element modified by A. Thus, A and B will not conflict. Furthermore, opposed to the shared counter example, in such a scenario a transaction does

not necessarily face the conflict directly after start-up due to the first accessed resource (i.e. the head of the list) but more likely, at some later point in time. For simplicity, let us assume that all transactions conflict within time $[\frac{t_T}{c_0}, \frac{(c_0-1)\cdot t_T}{c_0}]$. This is not much of a restriction, since clearly the vast majority of transactions (more precisely a fraction $1-\frac{1}{c_0}$) is expected to face a conflict within time $[\frac{t_T}{c_0}, \frac{(c_0-1)\cdot t_T}{c_0}]$. We assume that all transactions start randomly within time $[0, t_T]$.

Proposition 99. *For immediate restart the expected time span until all transactions committed is* $\Theta(n \cdot t_T)$.

Proof. After time t_T all transactions have started and within time $2t_T$ at least one transaction – say A – has committed. The time until the transaction B with highest probability commits is at least $\frac{t_T}{c_0}$. The same holds for the transaction C of third highest overall probability. Thus, the time until all n transactions committed adds up to $\Theta(n \cdot t_T)$. □

For *AbortBackoff* the expected time span until all transactions committed is $O(n \cdot t_T \cdot 2^{O(\sqrt{\log n})})$ using an analogous analysis as in the proof of Proposition 96. For the policies *SerializeFacedConflicts* and *SerializeAllHopConflicts* the expected time span until all transactions committed is $\Theta(n \cdot t_T)$, since all transactions execute sequentially.

15.2.4 Substantial Parallelism

It is not surprising that the serialization policy *SerializeAllHopConflicts* works well, when we consider a clique, since the policy assumes the graph to be a clique. Given that an adversary, maximizing the running time of the policy, can choose transactions priorities and their starting time, it is not hard to construct an example, where immediately restarting is exponentially faster than *SerializeAllHopConflicts*. Therefore, from a worst case perspective, serialization might be very bad. Due to the randomization the backoff scheme is more robust in such cases.

Proposition 100. *For the* SerializeAllHopConflicts *policy the expected time span until all transactions committed is* $O(n \cdot t_T)$ *for d-ary tree conflict graphs of logarithmic height and* $O(\log n \cdot t_T)$ *for immediate restart and* SerializeFacedConflicts.

Proof. For immediate restart consider an arbitrary node v_0 and look at all paths S_P with $P = (v_0, v_1, ..., v_x) \in S_P$ of nodes of increasing priority in the conflict graphs. Look at an arbitrary longest path $P \in S_P$. For any such path $P = (v_0, v_1, ..., v_x) \in S$ holds that the transaction v_x has maximum priority among all its neighbors and thus commits within time t_T. Thus any longest

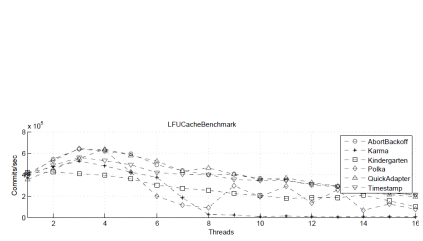
path reduces by 1 in length within time t_T. Since the graph is a tree of height $\log_d n$, i.e. for the number of nodes holds $d^{\log_d n} = n$, the length x of any path is bounded by $O(\log n)$ and the claim follows. For *SerializeFacedConflicts* we have that for a transaction v_0 itself or a neighbor is executing. Therefore, overall at least a fraction $1/d$ of transactions is running and within time t_T they either commit or face a conflict and abort. Thus, after time $d \cdot t_T$ any transaction is aware of all its conflicting neighbors and is not scheduled together with them again. However, any transaction always has at least one neighbor that is executing, i.e. a maximal independent set of transactions is scheduled and commits. Thus, the total time is $\mathrm{O}(d \cdot t_T)$, which is $\mathrm{O}(\log n \cdot t_T)$ since the tree is of logarithmic height $\log_d n$. For *SerializeAllHopConflicts* we assume that all leaves have lower priority than their parents. Furthermore, we make a node first conflict with its children. More precisely, assume all transactions start concurrently and a transaction conflicts with up to d other transactions. Assume that the root acquires all its resources within time interval $]t_T - d, t_T]$. All children of the root acquire their resources within time interval $]t_T - 2d, t_T - d]$. In general, a node at level i of the tree gets its resources within time $]t_T - id, t_T - (i-1)d]$. Thus when the root transaction R commits, all its neighboring children $N(R)$ must have faced a conflict and have got aborted by R. In general, before a transaction $T \in N(T_P)$ conflicts with its parent T_P, it aborts all its children $N(T) \setminus T_P$. Thus, the leafs are aborted first and the children of the root at last. Therefore, within time t_T all transactions are assumed to conflict with each other and are executed sequentially, resulting in a running time of $n \cdot t_T$. □

For the *AbortBackoff* policy the expected time span until all transactions committed is $\mathrm{O}(d \cdot t_T \cdot 2^{O(\sqrt{\log n})})$ for d-ary tree conflict graphs of logarithmic height. The proof is analogous to Proposition 96.

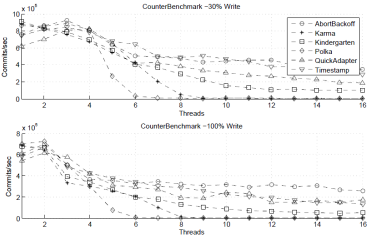
15.3 Experimental Results

15.3.1 Contention Management Policies

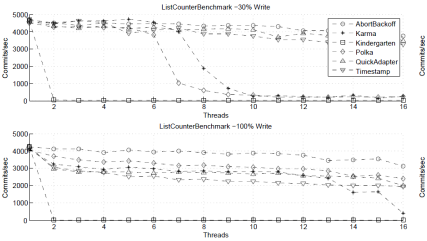
The policies *SerializeFacedConflicts* and *SerializeAllHopConflicts* ignored the overhead of keeping track of conflicts among transactions. In practice, it turned out that logging each conflict causes too much overhead in many scenarios. That is why, we derived a new serialization technique called *Quick-Adapter*. It does not come with a priority calculation scheme. For the implementation we used the timestamp manager. Using time as priority results in the same theoretical properties as *SerializeAllHopConflicts*. In particular, deadlock- and livelock-freedom, since the oldest transaction runs without interruption until commit. According to [107] from a theoretical (worst case)



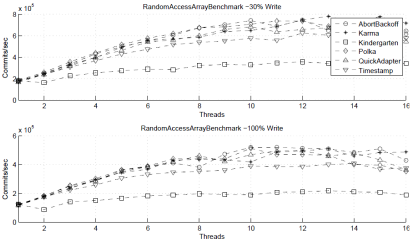
(a) LFUCache



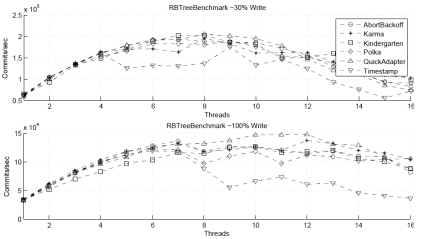
(b) Counter



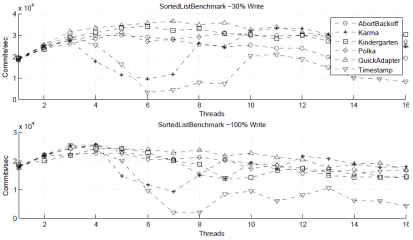
(c) ListCounter



(d) RandomAccessArray



(e) RBTree



(f) SortedList

Figure 15.1: Benchmarks: For 0% writes most policies behave similar since they introduce almost no overhead. For 60% writes the results lie (as expected) in between 30% and 100%.

perspective assigning random priorities yields better results. Still, in practice we found that the choice of the manager is of secondary importance. Every transaction has a flag which is set if it is not allowed to (re)start.[1] If a transaction gets aborted it sets its flag and does not restart. A committed transaction selects one of the flags and unsets it. For the implementation we chose an array (of flags), which equals the length of the maximum number of transactions. We investigated two variants. For the *QuickAdapter* each thread maintains a counter and whenever a thread commits it increments the counter and unsets the flag at the position in the array given by counter modulo array length. In case contention is very high and most transactions are aborted, any committing transaction has a high chance to restart a waiting transaction. But in such a situation it might be better to be more restrictive and rather not activate another transaction on commit. *SmartQuickAdapter* accounts for this and looks at the status of two (random) transactions. In case both are active, it selects a flag and unsets it (if it is set). Clearly, the higher contention, the more transactions are aborted and the smaller the chance for a committing transaction to reactivate an aborted transaction.

For the *AbortBackoff* manager the priority is determined by the number of aborts of a transaction. In case two transactions have the same priority, the one that runs on the thread with smaller identifier is aborted. For any conflict the transaction with smaller priority gets aborted. Before an aborted transaction can restart it has to wait a period which grows exponentially (by a factor of 4) with the number of times the transaction already got aborted. Each transaction starts with 0 aborts. We also evaluated a scheme *RememberingBackoff*, where a transaction carries over the number of aborts minus 1 of the previous transaction (executed by the thread).

15.3.2 Experimental Results

The benchmarks were executed on a system with four Quad-Core Opteron 8350 processors running at a speed of 2 GHz. The DSTM2.1 Java library was used, compiled with Sun's Java 1.6 HotSpot JVM. We present experimental results for the described contention managers on six different benchmarks. Five of them have been used already in prior work of Scherer at al. [104]. The sixth benchmark, RandomAccessArray, works on an array with 255 integers. The write operation chooses a random field $i \in [0, 254]$ and alternately increments or decrements the element and its eight neighbors $j \in [i + 1, i + 8]$. The read operation reads those elements. Every benchmark represents the average throughput of three runs with the same configuration for a duration

[1] If there are few commits, a single transaction having a set flag might wait for a long time until restart. Therefore one might consider adding a maximum waiting duration until a transaction restarts or check from time to time if there are any active transactions. Usually commits are frequent and, thus, this is not an issue.

of 10 seconds. Except LFUCache all the benchmarks were tested among different contention levels. Low contention was achieved through a write to read ratio of 0% and 30%, middle and high contention with ratios of 60% and 100%. The throughput was measured with 1 up to a maximum of 16 active threads.

Figure 15.1 shows the experimental results on the six benchmarks and several contention management policies (see [104]). In the plots we only show the *QuickAdapter* and not the *SmartQuickAdapter* scheme. Overall both perform similarly. *AbortBackoff* and *RememberingBackoff* (and other variants) all achieve similar throughput and therefore we only show *AbortBackoff* in the plots.

Generally, it can be seen that for most benchmarks the throughput declines with an increasing number of used threads, i.e. cores. This happens even in the case without writes. That is to say, even without conflicts, i.e. irrespective of the used contention management and load adaption policy, performance decreases. DSTM2 is used with visible readers. For a visible reader system the metadata of a read object is modified, which generally causes cache misses and slows down the system with an increasing number of threads.[2] Another reason for the decrease in performance when using more cores might be that for practically all benchmarks only little computation is done but (relatively) a lot of memory is accessed. In such a scenario the memory bus becomes a bottleneck and main memory accesses become slower.

The ListCounter benchmark provides the longest transactions among the six tested benchmarks. Therefore it is very prone to livelocks. Kindergarten's throughput drops to zero if more than one thread is active, same happens for Karma and Polka if more than 9 threads are running. Both *QuickAdapter* and *AbortBackoff* (and their variants) achieve consistently good results on all different benchmarks. In the majority of the cases they outperform the existing policies. If not, they do not lag behind much. The throughput of all other managers depends very much on the benchmark. Each drops off by more than 50% compared to the best manager on at least one of the benchmarks. This is not surprising, since any traditional (non-load adapting) contention manager adapts a specific heuristic for calculating the priority of a transaction, which is only efficient in some cases and fails for other cases. It is particularly interesting to compare *QuickAdapter* and *TimeStamp*, since both use time as priority. In all but one scenario *QuickAdapter* is faster: For the counter benchmark, which enforces sequential execution, for *QuickAdapter* a committing transaction A wakes up an old transaction that aborts the new transaction after A. But it would be better to assign a new timestamp whenever a transaction is woken up. This seems to be less of an issue

[2] At the time of writing, the discussion whether visible or invisible readers are preferable has not come to a definite end.

with the *TimeStamp* manager. It is not clear how to create a final ranking. Some benchmarks might be more important than others and some application scenarios might not be covered by the benchmarks. We ranked each benchmark based on the number of committed transactions for 12 threads and 60% writes. Managers are ranked equally if the throughput differs by less than 5%. The average rank of *QuickAdapter* is 1.7, that of *AbortBackoff* is 1.8 and only then follows *TimeStamp* and *Karma* with rank 3.5. *Polka* reaches an average of 4 and *Kindergarten* was last with 4.2.

Chapter 16

Parallel Algorithms Involving Graphs

16.1 Introduction

Parallel algorithms are a well established field since almost three decades. In this thesis we restricted ourselves to algorithms dealing with graphs only in two different ways. First (Section 16.2), we consider at how one can improve on the performance of tree like data structures and second (Section 16.3), we look at certain graph problems (like coloring) and derive algorithms to solve them very efficiently, i.e. without any synchronization at all.

16.2 Tree Decomposition

With the rise of multi-core processors efficient and simple parallel data structures gain more and more importance. Though there is a rich literature on concurrent data structures, outside of the database community surprisingly little work has been carried out to deal with general operations beyond inserts, deletes and finds. For instance, any common operation such as an update query on a subset of all elements in a list is often not handled well with conventional techniques, meaning that either the whole data structure is locked or each modified element is locked. In the first case, all operations are sequential, and in the second case the overhead due to acquiring and releasing locks frequently induces an unwanted time penalty. In databases this issue has been addressed from the 1970s onwards through various mechanisms such as hierarchical locking. Our proposal is less complicated and does not involve hierarchies. We group elements together, such that only one lock per group must be acquired.

16.2.1 Related Work

For database systems, predicate locks are an approach to logically lock elements [37]. A predicate, such as "all data records with field x larger some value", defines all tuples that are of interest for a transaction. In theory and practice, finding intersecting predicates is a difficult undertaking and has not been used in real systems [47, 60], whereas hierarchical locking is standard since the 1970s. In database systems only whole blocks with several data records can be locked due to physical constraints such as the addressable size of a block on a hard drive, whereas for concurrent data structures any object can be locked. Our approach intentionally introduces a minimum granularity for locking objects. We argue that the complexity of hierarchical locking is well justified for large centralized database systems where hundreds or thousands of operations are executed in parallel, but it is not necessary in a typical multi-core system, where the number of actual concurrent transactions is in the order of the number of cores.

Many implementations of concurrent data structures which support only insert, delete and find operations have been proposed. For example, in [113] a concurrent binary tree implementation is given such that locks are only acquired for modified objects (as for the lazy concurrent list in [55]). Our algorithm and the one in [55] need one lock for such an operation, but we have another book keeping overhead (i.e. remembering the leader of the currently traversed group) and furthermore, in general, we slightly restrict the potential parallelism. Thus, for the mere standard operations insert, delete and find, these algorithms are somewhat more efficient than our approach. However, the proposed strategies [55, 113] are not well suited for operations beyond inserts, deletes and finds. In particular, if one iterates through a list or tree modifying many objects, all these objects must be protected (e.g. locked). Thus there is "no free lunch" in the general case.

16.2.2 Data Structure Decomposition

We distinguish two types of nodes: group leaders and group members. All group members that are reachable from a leader by member objects form a group (together with the leader). See Figure 16.1.

A traversal of the data structure starts from the root, which is always a leader. When an object to be modified is encountered, any thread working on some object in the same group must also have passed by the group leader. Therefore, it suffices to lock leaders, i.e. before a group member is modified its group leader is locked. This holds for lists as well as for all kinds of tree traversals such as depth or breadth first. The choice of group leaders can be made on different basis. For example, when a new node is inserted it is randomly chosen to be either a group leader or a group member. On the

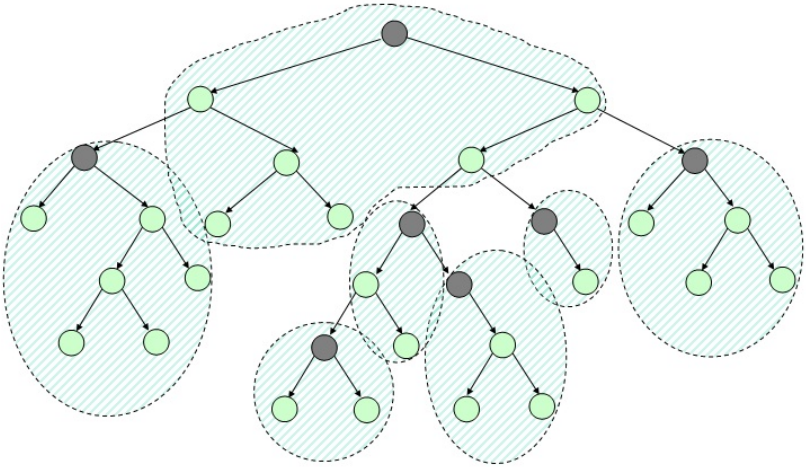


Figure 16.1: Decomposition of a directed rooted graph, where dark vertices indicate leaders and pale vertices show group members. A group is shown by a striped area.

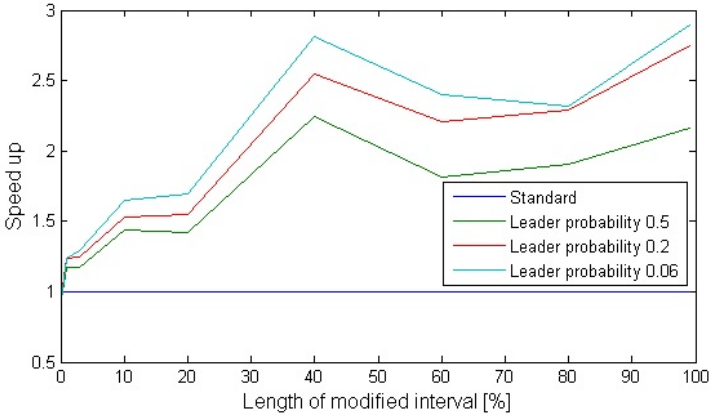


Figure 16.2: Speedup of a concurrent linked list when using groups of expected sizes 2, 5 and 17. The x-axis gives the length of the modified interval in per cent of the total list length.

positive side, if several elements are modified in the same group, only one lock needs to be acquired. On the negative side, if several threads operate on different elements in the same group, they face a conflict and cannot run concurrently. Thus, if the size of the group is large, e.g. the whole tree in the most extreme case, the restriction of possible parallelism might have a bigger impact than the savings due to using less locks. However, since our technique is simple and comes with little overhead, it results in a performance gain already for small groups. The overhead is low, since membership relations of the groups are implicit, i.e. group members do not need to know their own group (i.e. their leader) and a leader object does not need to know the nodes in its group. If this was not the case then any deletion of a leader caused an overhead proportional to the size of the group, since all members must be updated. In our case leader deletions also cause complications, i.e. if a leader L is deleted then the group led by L is implicitly merged with the prior group. Though, an operation can notice that L got deleted when it attempts to lock L (e.g. by having a deleted flag), it might be unaware of its new group leader, but it could retraverse the list/tree to find it. When a new leader L gets inserted, a group might be split. To find out, if the leader is still current, versions can be used, i.e. whenever a group is split by inserting a new leader, the version of the original group leader is incremented. Thus, by checking the version of a leader L, an operation can be sure that the group has not been changed. However, there are also other ways to deal with these issues, i.e. a simple way to deal with deletions is to use "dummy" nodes as leaders that cannot be deleted. In future work, we will evaluate several strategies. A series of deletions and inserts might degenerate the decomposition, i.e. create a few very large groups and many small groups. In future work, we intend to develop and test rebalancing strategies for groups.

For evaluation we used Java with 16 threads on a system with four quad-core Opteron 8350 processors. We implemented a sorted concurrent linked list running Algorithm *StdModifyRangeValues(int x,int y)*. The algorithm traverses the list and changes all objects, which have a key larger than x and smaller than y. If we have found the first object O with key larger than x (and smaller y), then we cannot just lock object O but must lock the last accessed leader.

We added all numbers in the range [0,10000]. Then Algorithm *StdModifyRangeQuery* was executed on an interval $[x, y]$ of varying width $y - x$, start element x and with varying group sizes (see Figure 16.2). If no elements are modified our algorithm is about 5% slower due to the overhead of remembering the leader. It holds that the larger the groups and the modified interval, the larger the speedup (up to some point). If an interval containing one percent of all elements is modified, the performance improvement is about 25% for groups of average size 5. If 10% of all elements are altered the gain

```
Algorithm StdModifyRangeValues(int x,int y)
 1: Node curr = list.getFirst(); Node lastLeader = null; {1st object is always
    a leader, e.g. dummy with key −∞}
 2: while curr.getValue()<y do
 3:    if curr.isLeader() then lastLeader = curr; end if
 4:    if (curr.getValue() > x) or (curr.getValue() < y) then
 5:       if lastLeader!=null then lastLeader.lock() endif
 6:       Change some field of curr
 7:       if curr.getNext().isLeader() then lastLeader.unlock(); lastLeader =
          null; endif
 8:    end if
 9:    curr = curr.getNext();
10: end while
11: if lastLeader!=null and lastLeader.isLocked() then lastLeader.unlock()
    endif
```

is about 65% for group sizes still below 20. For much larger group sizes (e.g. 200) the throughput does not increase compared to group sizes of around 20 – possibly, since the running time for groups beyond 20 is mainly dominated by the list traversal and locking operations account only for a small amount of the overall running time.

16.3 Graph Algorithms without Synchronization

Synchronization operations needed for shared data access cause a large performance penalty for parallel algorithms. When solving graph problems the task of parallelization boils down to split a graph into subgraphs keeping several goals in mind: First, the decomposition process itself should be efficient and simple. Second, if different threads work on different subgraphs, synchronization among these threads should be simple and cause only little overhead. Third, the solution quality, e.g. approximation ratio, should be as good as possible. For example, for coloring the number of employed colors should be minimized. But even coarsely approximating an optimal solution is NP-hard. Thus, in basically any application one is willing to settle for much more than the minimum number of colors and save on computing time. In particular, in fast changing environments, e.g. all kinds of highly dynamic graphs/networks, it is necessary to compute an approximate solution as quickly as possible, since it might be valid only for a short time span. In such situations one is well willing to trade solution quality for computation time. Even checking the correctness of a coloring requires looking at (the

colors of) all neighbors of a node, i.e. run time $\Omega(|E|)$. We match this lower bound without relying on synchronization.

16.3.1 Related Work

One way to parallelize operations on graphs is to decompose a graph into subgraphs and let each processor compute a solution on a subgraph. Unfortunately, these subgraphs are usually not independent and thus boundary constraints stay in place, which require slow synchronization primitives such as locks. For instance, [67] proposes to iteratively compute maximal matchings. Any pair u, v of matched nodes in a graph G is merged together to form a new node v' in G'. Then a matching is computed in G'. This process is repeated until only p nodes are left. Each node corresponds to a (collapsed) subgraph and is assigned to a processor/core. Thus, for a simple ring with n nodes $\log(n/p)$ matchings must be computed. Every matched node requires an access to a synchronization primitive, since several processors might try to access the same node. For other techniques such as spectral partitioning see [67] for related work.

Coloring has been studied extensively in a local setting, where each node in the graph corresponds to a processor see Chapter 2.2 in Part I. The fastest algorithm requires no communication to compute an $O(\Delta^2 \log^2 n)$ coloring if a node knows its neighbors. Computing an $O(\Delta + \log^{1+1/\log^* n} n)$ coloring needs $O(\log^* n)$ rounds,. i.e. synchronization steps of all processors. In comparison, our algorithm requires no synchronized rounds at all in its simplest version. In the worst-case it might use up to $O(\Delta \cdot p)$ colors, where p is the number of processors. From a theoretical point of view, using that many colors is not too bad given that coloring is a hard problem to approximate and the number of processors is often negligible (i.e. constant) compared to the size of the maximum degree. Still, in many cases heuristics allow to compute colorings that allow for less colors than Δ. Unfortunately, the previously mentioned (constant) time algorithm in general requires colors from the entire set of available colors, e.g. to color a tree might require $\Omega(\Delta^2 \log^2 n)$ colors, though two colors suffice. A more effective sequential heuristic is the following greedy strategy: Color node after node, such that any node gets the smallest color not taken by its neighbor. [21] uses this approach. The graph is partitioned into disjoint clusters (obtained by some algorithm, e.g. [67]). Each node is either a boundary node, i.e. it has a neighbor in another cluster, or a non-boundary node. The algorithm iterates two phases. In the first phase each processor speculatively colors the (uncolored) vertices assigned to it in parallel using the greedy sequential heuristic. The second phase consists of a color conflict-detection (for the boundary nodes).

For minimum dominating sets [79] gives a algorithm in a local setting for

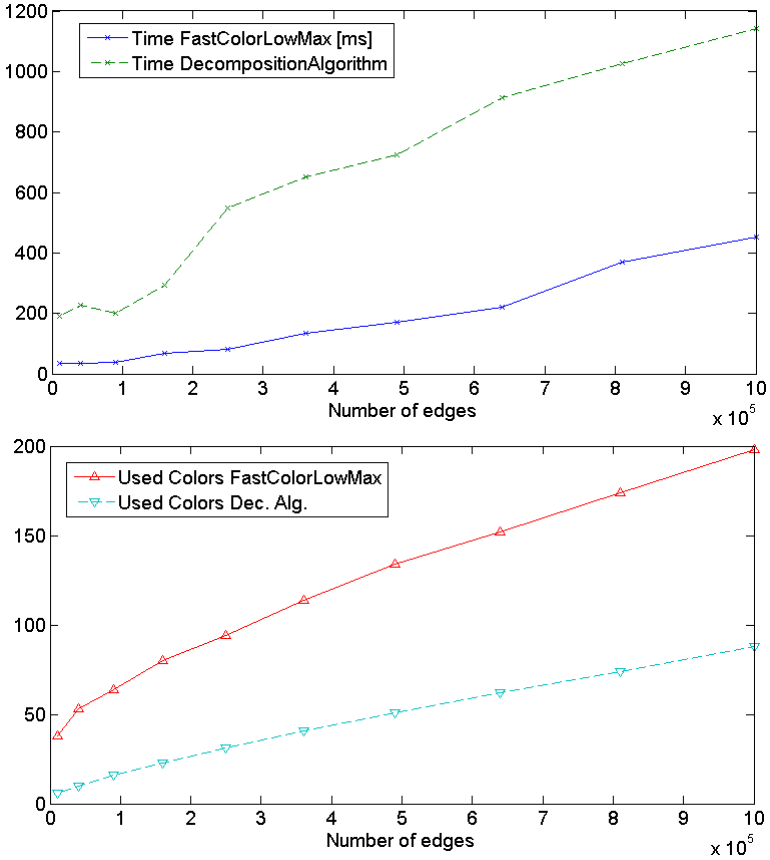


Figure 16.3: Used colors and time in milliseconds for Algorithm *FastColorLowMax* compared to [21], where random matchings [67] were used for graph decomposition. The number of nodes are fixed to 5000, and the number of edges are varied from 10^4 to 10^6.

an $O(k\Delta^{2/k} \log \Delta)$ approximation using $O(k^2)$ synchronization steps. Alternatively, one might employ [67] to compute a decomposition using synchronization and then greedily compute a $\log \Delta$ approximation. We achieve a $p \log \Delta$ approximation without synchronization.

16.3.2 Technique and Applications

We assign a distinct solution space to each processor, e.g. for the coloring problem, processor one can use colors $[0, \Delta]$ to color all nodes assigned to it, processor two can use colors $[\Delta + 1, 2\Delta + 1]$ and so forth. Nodes are assigned in an arbitrary manner (e.g. randomly) to processors. Therefore, we do not have any boundary constraints, since nodes assigned to different processors get different colors. Additionally, since any graph can be colored with $\Delta + 1$ colors, every processor can compute a solution, independently of all others. Unfortunately, this algorithm uses up to $(\Delta + 1) \cdot p$ colors and even if fewer colors are used they will be distributed in the range $[0, (\Delta + 1) \cdot p]$. We overcome this issue by first computing a coloring using up to $(\Delta + 1) \cdot p$ and then only keep the color ranges used by the processors, e.g. consider a system with two processors. If processor 1 has assigned colors $[0, c_{p1}]$, where c_{p1} is the largest assigned color, and processor 2 colors from $[\Delta + 1, \Delta + c_{p2}]$, then we set a node assigned by processor 2 having color c to color $c - (\Delta + 1) + c_{p1}$. In other words, we do not simply use the offset of $i \cdot (\Delta + 1)$ for the colored nodes of processor i but rather compute the offset based on the number of actually used colors of processors j with $j < i$. We refer to this approach by Algorithm *FastColorLowMax*. By slightly modifying our algorithm, we can give a tradeoff between synchronization effort and the number of used colors. Say in Algorithm *FastColorLowMax* a processor p can use only $(\Delta + 1)/r$ colors instead of $\Delta + 1$ for some parameter r. If processor i lacks sufficient colors to color all its assigned nodes, it passes its uncolored nodes to the next processor $i + 1 \bmod p$, which tries to color them. If the total number of colors $(\Delta + 1)/r \cdot p \geq \Delta + 1$, i.e. $p \geq r$, where r is the number of times colors are passed to the next processor, we can be sure that a correct coloring is computed using at most $p/r(\Delta + 1)$ colors.

Though, Algorithm *FastColorLowMax* using $\Delta + 1$ colors per processor requires little coordination among processors, as a drawback more colors are likely to be needed. Apart from the upper bound of $(\Delta + 1) \cdot p$, the number of colors will be in $\Omega(p)$ even for graphs of lower degree than p. This happens because usually we have many more nodes than processors and thus, if we assign nodes randomly to processors, almost surely, every processor gets assigned at least one node and must use one color for it. For illustration of the approximation quality, assume we have a disconnected graph consisting of cliques of size $\Delta + 1$. Thus, using a (non-parallel) sequential greedy strategy

for the whole graph will result in a coloring with $\Delta + 1$ colors. For the parallel algorithm *FastColorLowMax*, we have that the probability that a processor gets assigned an entire clique is $1/p^{\Delta+1}$. Assume we have $p \ll n$ and $p^{\Delta+1}$ cliques all of small degree, i.e. $n = (\Delta + 1) \cdot p^{\Delta+1}$ or roughly, $\Delta \approx \log n / \log p$. Then, the probability for a processor to be assigned a whole clique becomes $1 - (1 - 1/p^{\Delta+1})^{p^{\Delta+1}} \approx 1 - 1/e \approx 1/2$. Using a Markov bound, the probability that more than $3/4$ of all p processors are not associated with an entire clique is at most $2/3$, thus we expect indeed an approximation factor of at least $3/4 \cdot 2/3 \cdot p = \Omega(p)$. However, if the maximum degree is larger, say $\Delta = n^c - 1$ for some constant $c < 1$, then using a Chernoff bound the probability that a processor gets assigned more than $(1+\log n/n^{c/2}) \cdot n^c/p$ or less than $(1-\log n/n^{c/2}) \cdot n^c/p$ nodes is smaller than $1-1/n^{\log n}$, i.e. roughly $(1-1/n^{\log n})^p > 1-1/n^{\log n-1}$ for all processors since by assumption $p < n$. Thus, most likely the total number of used colors is only $(1 + \log n/n^{c/2}) \cdot n^c/p \cdot p = (1 + \log n/n^{c/2}) \cdot n^c$ compared to n^c by an optimal algorithm, yielding an approximation ratio converging to 1 as n increases, i.e. $1 + \log n/n^{c/2}$.

Similar thoughts apply for computing a dominating set. After assigning nodes randomly to processors, each processor iteratively picks an (assigned) node with the maximum number of non-dominated neighbors. Since a greedy approach yields an $\log \Delta$ approximation of a minimum dominating set, the overall approximation ratio is at most $p \log \Delta$.

For evaluation we used Java on a system with four quad-core Opteron 8350 processors. Figure 16.3 shows that for dense graphs our algorithm uses twice as many colors as [21], since for every node its neighbors are distributed relatively evenly among the processor. For sparser graphs it gets worse. But it is always 3 to 5 times faster.

Chapter 17

Conclusions

Though the optimal contention manager cannot be determined with reasonable computational effort, there are simple distributed strategies that give good guarantees regarding throughput (as well as progress and fairness guarantees in some cases). From a practical point of view there are a couple of "all-purpose" contention managers like timestamp that yield good results over a wide set of benchmarks (though having bad worst-case performance), whereas others (like Polka) dramatically fall behind under certain conditions as predicted by theoretical analysis. Given that an "all-purpose" contention manager is chosen, employing load adaption can yield significant improvement of overall system performance. For a number of graph problems such as coloring and dominating sets, our graph decomposition technique allows to trade solution quality for computation time by reducing the synchronization overhead. We also presented a method for partitioning data structures to allow for efficient traversal and modification of elements. We believe that iteration through sets is a common operation – in many cases equally important as the standard find, insert and delete operations. Thus, it might be worthwile to incorporate our technique in libraries containing list or tree data structures.

Bibliography

[1] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986.

[2] N. Alon, A. Bar-Noy, N. Linial, and D. Peleg. A lower bound for radio broadcast. *J. Comput. Syst. Sci.*, 43(2), 1991.

[3] K. Alzoubi, X. Li, Y. Wang, P. Wan, and O. Frieder. Geometric Spanners for Wireless Ad Hoc Networks. *IEEE Transactions on Parallel and Distributed Systems*, 14(5), 2003.

[4] A. Andersson, T. Hagerup, S. Nilsson, and R. Raman. Sorting in Linear Time. *Journal of Computer and System Sciences*, 57:74–93, 1998.

[5] M. Ansari, C. Kotselidis, M. Luján, C. Kirkham, and I. Watson. On the Performance of Contention Managers for Complex Transactional Memory Benchmarks. In *Symp. on Parallel and Distributed Computing*, 2009.

[6] S. Arora and E. Chlamtac. New approximation guarantee for chromatic number. In *Symp. on Theory of computing(STOC)*, 2006.

[7] H. Attiya, L. Epstein, H. Shachnai, and T. Tamir. Transactional contention management as a non-clairvoyant scheduling problem. In *Symposium on Principles of distributed computing*, 2006.

[8] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Fast distributed network decompositions and covers. *J. Parallel Distrib. Comput.*, 39(2):105–114, 1996.

[9] B. Awerbuch, A. V. Goldberg, M. Luby, and S. A. Plotkin. Network decomposition and locality in distributed computation. In *Symp. on Foundations of Computer Science (FOCS)*, 1989.

[10] B. Awerbuch and G. Varghese. Distributed program checking: a paradigm for building self-stabilizing distributed protocols. In *Symp. on Foundations of Computer Science (FOCS)*, 1991.

[11] A. Bar-Noy, M. Bellare, M. M. Halldorsson, H. Shachnai, and T. Tamir. On chromatic sums and distributed resource allocation. *Information and Computation*, 140(2):183–202, 1998.

[12] R. Bar-Yehuda, O. Goldreich, and A. Itai. On the Time-Complexity of Broadcast in Radio Networks: an Exponential Gap between Determinism and Randomization. In *Proceedings of the 6^{th} ACM Symp. on Principles of Distributed Computing (PODC)*, pages 98–108, 1987.

[13] L. Barenboim and M. Elkin. Sublogarithmic distributed mis algorithm for sparse graphs using nash-williams decomposition. In *Symposium on Principles of distributed computing(PODC)*, 2008.

[14] L. Barenboim and M. Elkin. Distributed $(\delta + 1)$-coloring in linear (in δ) time. In *Symposium on Theory of Computing(STOC)*, 2009.

[15] L. Barenboim and M. Elkin. Deterministic distributed vertex coloring in polylogarithmic time. In *Symp. on Principles of distributed computing(PODC)*, 2010.

[16] L. Barenboim and M. Elkin. Combinatorial algorithms for distributed graph coloring. In *Symp. on Distributed Computing (DISC)*, 2011.

[17] L. Barenhoim and M. Elkin. Sublogarithmic Distributed MIS Algorithm for Sparse Graphs using Nash-Williams Decomposition. In *Journal of Distributed Computing Special Issue of selected papers from PODC 2008*, 2010.

[18] A. Blum. New approximation algorithms for graph coloring. *Journal of the ACM*, 41:470–516, 1994.

[19] B. Bollobas. Chromatic nubmer, girth and maximal degree. *Discrete Math.*, 24:311–314, 1978.

[20] O. Bonorden, B. Degener, B. Kempkes, and P. Pietrzyk. Complexity and approximation of a geometric local robot assignment problem. In *ALGOSENSORS*, 2009.

[21] D. Bozdag, A. H. Gebremedhin, F. Manne, E. G. Boman, and Ü. V. Çatalyürek. A framework for scalable greedy coloring on distributed-memory parallel computers. *J. Parallel Distrib. Comput.*, 68(4), 2008.

[22] B. Chlebus, L. Gcasieniec, A. Gibbons, A. Pelc, and W. Rytter. Deterministic broadcasting in unknown radio networks. In *Symp. on Discrete Algorithms*, 2000.

[23] A. E. F. Clementi, A. Monti, and R. Silvestri. Distributed broadcast in radio networks of unknown topology. *Theor. Comput. Sci.*, 302(1-3), 2003.

[24] R. Cole and U. Vishkin. Deterministic Coin Tossing with Applications to Optimal Parallel List Ranking. *Inf. Control*, 70(1):32–54, 1986.

[25] A. Czumaj and W. Rytter. Broadcasting algorithms in radio networks with unknown topology . In *Symp. on Foundations of Computer Science (FOCS*, 2003.

[26] A. Czygrinow, M. Hanckowiak, and W. Wawrzyniak. Fast distributed approximations in planar graphs. In *DISC*, 2008.

[27] J. Czyzowicz, L. Gasieniec, D. R. Kowalski, and A. Pelc. Consensus and mutual exclusion in a multiple access channel. In *Symposium on Distributed Computing (DISC)*, 2009.

[28] L. Dalessandro, M. F. Spear, and M. L. Scott. Norec: streamlining stm by abolishing ownership records. In *PPOPP*, 2010.

[29] P. Damron, A. Fedorova, Y. Lev, V. Luchangco, M. Moir, and D. Nussbaum. Hybrid transactional memory. In *Proc. ASPLOS Conference*, 2006.

[30] B. Derbel and E.-G. Talbi. Distributed node coloring in the sinr model. In *ICDCS*, 2010.

[31] A. Dessmark and A. Pelc. Broadcasting in geometric radio networks . In *Journal of Discrete Algorithms*, volume 5, pages 187–201, 2007.

[32] Y. Dinitz, S. Moran, and S. Rajsbaum. Bit complexity of breaking and achieving symmetry in chains and rings. *J. ACM*, 2008.

[33] S. Dolev, D. Hendler, and A. Suissa. CAR-STM: scheduling-based collision avoidance and resolution for software transactional memory. In *PODC*, 2008.

[34] R. Eidenbenz and R. Wattenhofer. Good Programming in Transactional Memory: Game Theory Meets Multicore Architecture. In *Symposium on Algorithms and Computation (ISAAC)*, 2009.

[35] P. Erdoes, P. Frankl, and Z. Fiiredi. Families of finite sets in which no set is covered by the union of r others. In *Israel J. of Math*, volume 51, 1985.

[36] P. L. Erdös, P. Frankl, and Z. Füredi. Families of finite sets in which no set is covered by the union of two others. *J. Comb. Theory, Ser. A*, 33(2):158–166, 1982.

[37] K. P. Eswaran, J. Gray, R. A. Lorie, and I. L. Traiger. The notions of consistency and predicate locks in a database system. *Commun. ACM*, 19(11), 1976.

[38] G. Even, M. M. Halldorsson, L. Kaplan, and D. Ron. Scheduling with conflicts: online and offline algorithms. In *Journal of Scheduling*, 2008.

[39] P. Fraigniaud, C. Gavoille, D. Ilcinkas, and A. Pelc. Distributed computing with advice: information sensitivity of graph coloring. *Distributed Computing*, 2009.

[40] P. Fraigniaud and G. Giakkoupis. On the bit communication complexity of randomized rumor spreading. In *SPAA*, 2010.

[41] G. N. Frederickson and N. A. Lynch. Electing a leader in a synchronous ring. *J. ACM*, 34(1), 1987.

[42] C. Gavoille, R. Klasing, A. Kosowski, L. Kuszner, and A. Navarra. On the complexity of distributed graph coloring with local minimality constraints. *Networks*, 54(1), 2009.

[43] B. Gfeller and E. Vicari. A Randomized Distributed Algorithm for the Maximal Independent Set Problem in Growth-Bounded Graphs. In *Symp. on Principles of Distributed Computing*, 2007.

[44] A. Goldberg, S. Plotkin, and G. Shannon. Parallel Symmetry-Breaking in Sparse Graphs. *SIAM Journal on Discrete Mathematics (SIDMA)*, 1(4):434–446, 1988.

[45] D. A. Grable and A. Panconesi. Nearly optimal distributed edge colouring in o(log log n) rounds. In *Symp. on Discrete Algorithms (SODA)*, 1997.

[46] D. A. Grable and A. Panconesi. Fast distributed algorithms for Brooks-Vizing colorings. *J. Algorithms*, 37(1):85–120, 2000.

[47] G. Graefe. Hierarchical locking in b-tree indexes. In *BTW*, 2007.

[48] A. G. Greenberg and S. Winograd. A lower bound on the time needed in the worst case to resolve conflicts deterministically in multiple access channels. *J. ACM*, 32(3), 1985.

[49] R. Guerraoui, M. Herlihy, and B. P. Michal Kapalka and. Robust Contention Management in Software Transactional Memory. In *Proceedings of the Workshop on Synchronization and Concurrency in Object-Oriented Languages*, 2005.

[50] R. Guerraoui, M. Herlihy, and B. Pochon. Polymorphic contention management. *DISC*, 2005.

[51] R. Guerraoui, M. Herlihy, and B. Pochon. Toward a theory of transactional contention managers. In *Symp. on Principles of distributed computing*, 2005.

[52] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2), 2000.

[53] M. M. Halldórsson and J. Radhakrishnan. Greed is good: approximating independent sets in sparse and bounded-degree graphs. In *STOC*, 1994.

[54] T. He, C. Huang, B. M. Blum, J. A. Stankovic, and T. F. Abdelzaher. Range-free localization and its impact on large scale sensor networks. *ACM Trans. Embedded Comput. Syst.*, 4(4), 2005.

[55] S. Heller, M. Herlihy, V. Luchangco, M. Moir, W. N. S. III, and N. Shavit. A lazy concurrent list-based set algorithm. *Parallel Processing Letters*, 17(4), 2007.

[56] M. Herlihy, J. Eliot, and B. Moss. Transactional Memory: Architectural Support For Lock-free Data Structures . In *Symp. on Computer Architecture*, 1993.

[57] M. Herlihy, V. Luchangco, and M. Moir. A flexible framework for implementing software transactional memory. In *Proceedings of the OOPSLA Conference*, 2006.

[58] M. Herlihy, V. Luchangco, M. Moir, and W. Scherer. Software transactional memory for dynamic-sized data structures. In *Symp. on Principles of distributed computing*, 2003.

[59] T. Herman and S. Tixeuil. A Distributed TDMA Slot Assignment Algorithm for Wireless Sensor Networks. In *Lecture Notes in Computer Science*, volume 3121/2004, pages 45–58, 2004.

[60] H. B. H. III and D. J. Rosenkrantz. The complexity of testing predicate locks. In *SIGMOD Conference*, 1979.

[61] D. Ilcinkas, D. R. Kowalski, and A. Pelc. Fast radio broadcasting with advice. *Theor. Comput. Sci.*, 411(14-15), 2010.

[62] A. Israeli and A. Itai. A Fast and Simple Randomized Parallel Algorithm for Maximal Matching. *Information Processing Letters*, 22:77–80, 1986.

[63] Ö. Johansson. Simple distributed Δ+1-coloring of graphs. In *Inf. Process. Lett.*, volume 70, 1999.

[64] T. Jurdziński, M. Kutylowski, and J. Zatopianski. Weak Communication in Radio Networks. In *Proceedings of Euro-Par,*, 2002.

[65] T. Jurdziński and G. Stachowiak. Probabilistic Algorithms for the Wakeup Problem in Single-Hop Radio Networks. In *Proceedings of the 13^{th} Int. Symposium on Algorithms and Computation (ISAAC)*, 2002.

[66] M. Karchmer and J. Naor. A fast parallel algorithm to color a graph with delta colors. *J. Algorithms*, 9(1):83–91, 1988.

[67] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *J. Parallel Distrib. Comput.*, 48(1), 1998.

[68] S. Khot. Improved Inapproximability Results for MaxClique, Chromatic Number and Approximate Graph Coloring. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, 2001.

[69] K. Kothapalli, C. Scheideler, M. Onus, and C. Schindelhauer. Distributed coloring in $O(\log^{1/2} n)$ bit rounds. In *International Parallel & Distributed Processing Symp. (IPDPS)*, 2006.

[70] D. R. Kowalski and A. Pelc. Broadcasting in undirected ad hoc radio networks. In *Distributed Computing*, volume 18, 2005.

[71] D. R. Kowalski and A. Pelc. Leader election in ad hoc radio networks: A keen ear helps. In *ICALP*, 2009.

[72] F. Kuhn. Weak Graph Coloring: Distributed Algorithms and Applications. In *Symp. on Parallelism in Algorithms and Architectures (SPAA)*, 2009.

[73] F. Kuhn, T. Moscibroda, T. Nieberg, and R. Wattenhofer. Fast Deterministic Distributed Maximal Independent Set Computation on Growth-Bounded Graphs. In *Symp. on Distributed Computing (DISC)*, 2005.

[74] F. Kuhn, T. Moscibroda, T. Nieberg, and R. Wattenhofer. Local Approximation Schemes for Ad Hoc and Sensor Networks. In *Workshop on Foundations of Mobile Computing (DIALM-POMC)*, 2005.

[75] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Initializing newly deployed ad hoc and sensor networks. In *Proceedings of the 10^{th} Annual Int. Conference on Mobile Computing and Networking (MOBICOM)*, pages 260–274, 2004.

[76] F. Kuhn, T. Moscibroda, and R. Wattenhofer. On the Locality of Bounded Growth. In *Proceedings of the 24^{th} ACM Symp. on Principles of Distributed Computing (PODC)*, pages 60–68, 2005.

[77] F. Kuhn, T. Moscibroda, and R. Wattenhofer. What Cannot Be Computed Locally! In *Symp. on Principles of Distributed Computing (PODC)*, 2005.

[78] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Local Computation: Lower and Upper Bounds. In *http://arxiv.org/abs/1011.5470*, 2010.

[79] F. Kuhn and R. Wattenhofer. Constant-Time Distributed Dominating Set Approximation. In *Journal for Distributed Computing, Volume 17*, 2005.

[80] F. Kuhn and R. Wattenhofer. On the Complexity of Distributed Graph Coloring. In *Symp. on Principles of Distributed Computing (PODC)*, 2006.

[81] E. Kushilevitz and Y. Mansour. An $\Omega(D \cdot \log(\frac{N}{D}))$ lower bound for broadcast in radio networks. In *Symp. on Principles of Distributed Computing (PODC)*, 1993.

[82] E. Kushilevitz and Y. Mansour. An omega(d log(n/d)) lower bound for broadcast in radio networks. In *Symp. on Principles of Distributed Computing (PODC)*, 1993.

[83] E. Kushilevitz and N. Nisan. Communication complexity. In *Cambridge University Press*, 1997.

[84] E. Lebhar and Z. Lotker. Unit disk graph and physical interference model: Putting pieces together. In *IPDPS*, 2009.

[85] C. Lenzen, P. Sommer, and R. Wattenhofer. Optimal Clock Synchronization in Networks. In *Proc. on Embedded Networked Sensor Systems (SenSys)*, 2009.

[86] C. Lenzen and R. Wattenhofer. Leveraging Linial's Locality Limit. In *Symp. on Distributed Computing (DISC)*, 2008.

[87] C. Lenzen and R. Wattenhofer. MIS on Trees. In *Symp. on Principles of Distributed Computing (PODC)*, 2011.

[88] N. Linial. Locality in Distributed Graph Algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[89] D. B. Lomet. Process structuring, synchronization, and recovery using atomic actions. In *Proceedings of an ACM conference on Language design for reliable software*, pages 128–137, 1977.

[90] M. Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. volume 15, pages 1036–1053, 1986.

[91] Y. Métivier, J. M. Robson, S.-D. Nasser, and A. Zemmar. An optimal bit complexity randomised distributed mis algorithm. In *Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 2009.

[92] G. D. Marco and A. Pelc. Fast distributed graph coloring with $O(\delta^2)$ colors. In *SODA*, 2001.

[93] T. Moscibroda and R. Wattenhofer. Coloring Unstructured Radio Networks. In *Symp on Parallelism in Algorithms and Architectures (SPAA)*.

[94] T. Moscibroda and R. Wattenhofer. Maximal Independent Sets in Radio Networks. In *Proceedings of the 24^{th} ACM Symp. on Principles of Distributed Computing (PODC)*, pages 148–157, 2005.

[95] T. Moscibroda and R. Wattenhofer. Maximal Independent Sets in Radio Networks. In *Symp.Principles of Distributed Computing (PODC)*, 2005.

[96] N. Nisan and A. Wigderson. Rounds in communication complexity revisited. *SIAM J. Comput.*, 22(1), 1993.

[97] A. Panconesi and A. Srinivasan. Improved distributed algorithms for coloring and network decomposition problems. In *Symp. on Theory of computing (STOC)*, 1992.

[98] S. Pandit and S. Pemmaraju. Finding facilities fast. In *Proceedings of the 10th International Conference on Distributed Computing and Networking (ICDCN)*, 2009.

[99] S. Parthasarathy and R. Gandhi. Distributed algorithms for coloring and domination in wireless ad hoc networks. In *FSTTCS*, 2004.

[100] I. A. Pirwani and M. R. Salavatipour. A ptas for minimum clique partition in unit disk graphs. *CoRR*, 2009.

[101] H. Ramadan, C. Rossbach, D. Porter, O. Hofmann, A. Bhandari, and E. Witchel. MetaTM/TxLinux: transactional memory for an operating system. In *Symp. on Computer Architecture*, 2007.

[102] T. Riegel, C. Fetzer, and P. Felber. Time-based Transactional Memory with Scalable Time Bases. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, 2007.

[103] N. Santoro. *Design and Analysis of Distributed Algorithms.* Wiley-Interscience, 2006.

[104] W. Scherer and M. Scott. Advanced contention management for dynamic software transactional memory. In *Symp. on Principles of distributed computing*, 2005.

[105] T. Schmid, P. Dutta, and M. B. Srivastava. High-resolution, low-power time synchronization an oxymoron no more. In *Proc. on Information Processing in Sensor Networks (IPSN)*, 2010.

[106] J. Schneider and R. Wattenhofer. A Log-Star Distributed Maximal Independent Set Algorithm for Growth-Bounded Graphs. In *Symp. on Principles of Distributed Computing(PODC)*, 2008.

[107] J. Schneider and R. Wattenhofer. Bounds On Contention Management Algorithms. In *ISAAC*, 2009.

[108] J. Schneider and R. Wattenhofer. Coloring Unstructured Wireless Multi-Hop Networks. In *Symp. on Principles of Distributed Computing (PODC)*, 2009.

[109] J. Schneider and R. Wattenhofer. A New Technique For Distributed Symmetry Breaking. In *Symp. on Principles of Distributed Computing(PODC)*, 2010.

[110] J. Schneider and R. Wattenhofer. Distributed Coloring Depending on the Chromatic Number or the Neighborhood Growth. In *Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 2011.

[111] Y. Shang, W. Ruml, Y. Zhang, and M. P. J. Fromherz. Localization from connectivity in sensor networks. *IEEE Trans. Parallel Distrib. Syst.*, 15(11), 2004.

[112] G. Sharma, B. Estrade, and C. Busch. Window-based greedy contention management for transactional memory. In *DISC*, 2010.

[113] D. Shasha and N. Goodman. Concurrent search structure algorithms. *ACM Trans. Database Syst.*, 13(1), 1988.

[114] N. Shavit and D. Touitou. Software transactional memory. *Distributed Computing*, 10, 1997.

[115] Y. Shi, Y. T. Hou, J. Liu, and S. Kompella. How to correctly use the protocol interference model for multi-hop wireless networks. In *MobiHoc*, 2009.

[116] A. Sterling. Self-assembling systems are distributed systems. *CoRR*, 2009.

[117] D. E. Willard. Log-logarithmic selection resolution protocols in a multiple access channel. In *SIAM Journal on Computing*, volume 15, 1986.

[118] R. M. Yoo and H. S. Lee. Adaptive transaction scheduling for transactional memory systems. In *SPAA*, 2008.

Curriculum Vitae

October 10, 1979	Born in Bludenz, Austria
1986–1998	Primary, secondary, and high schools in Bludenz, Austria
1998–1999	Obligatory military service (8 months)
1999–2004	Studies in computer science, ETH Zurich, Switzerland
October 2004	M.Sc. in computer science, ETH Zurich, Switzerland
2005– November 2007	Software engineer, AutoForm Zurich, Switzerland
November 2007 –2011	Ph.D. student, research and teaching assistant, Distributed Computing Group, Prof. Roger Wattenhofer, ETH Zurich, Switzerland
November 2011	Ph.D. degree, Distributed Computing Group, ETH Zurich, Switzerland Advisor: Prof. Roger Wattenhofer Co-examiner: Prof. Uzi Vishkin, University of Maryland USA Co-examiner: Prof. Rachid Guerraoui, EPFL Lausanne, Switzerland

Publications

The following list enumerates the publications[1] I co-authored during my PhD at ETH Zurich.

Journals

1. Bounds on Contention Management Algorithms Johannes Schneider and Roger Wattenhofer. Theoretical Computer Science (TCS) Journal, July 2011.

2. An Optimal Maximal Independent Set Algorithm for Bounded-Independence Graphs Johannes Schneider and Roger Wattenhofer. Journal of Distributed Computing, March 2010.

Conference Proceedings

3. Poster Abstract: Message Position Modulation for Power Saving and Increased Bandwidth in Sensor Networks Johannes Schneider and Roger Wattenhofer. 10th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), USA, April 2011.

4. Poster Abstract: Three Plane Localization Johannes Schneider and Roger Wattenhofer. 10th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), USA, April 2011.

5. Brief Announcement: Efficient graph algorithms without synchronization Johannes Schneider and Roger Wattenhofer. 29th Symposium on Principles of Distributed Computing (PODC), Zurich, Switzerland, July 2010.

6. Brief Announcement: Tree decomposition for faster concurrent data structures Johannes Schneider and Roger Wattenhofer. 29th Symposium on Principles of Distributed Computing (PODC), Zurich, Switzerland, July 2010.

[1] Technical reports accompanying conference or journal publications are not listed.

7. Trading Bit, Message, and Time Complexity of Distributed Algorithms Johannes Schneider and Roger Wattenhofer. 25th International Symposium on Distributed Computing (DISC), Rome, Italy, September 2011.

8. Distributed Coloring Depending on the Chromatic Number or the Neighborhood Growth Johannes Schneider and Roger Wattenhofer. 18th International Colloquium on Structural Information and Communication Complexity (SIROCCO), Poland, June 2011.

9. What Is the Use of Collision Detection (in Wireless Networks)? Johannes Schneider and Roger Wattenhofer. 24th International Symposium on Distributed Computing (DISC), Cambridge, Massachusetts, USA, September 2010.

10. A New Technique For Distributed Symmetry Breaking Johannes Schneider and Roger Wattenhofer. 29th Symposium on Principles of Distributed Computing (PODC), Zurich, Switzerland, July 2010.

11. Transactional Memory: How to Perform Load Adaption in a Simple And Distributed Manner David Hasenfratz, Johannes Schneider, and Roger Wattenhofer. The 2010 International Conference on High Performance Computing and Simulation (HPCS), Caen, France, June 2010.

12. Bounds On Contention Management Algorithms Johannes Schneider and Roger Wattenhofer. 20th International Symposium on Algorithms and Computation (ISAAC), Honolulu, USA, December 2009.

13. Coloring Unstructured Wireless Multi-Hop Networks Johannes Schneider and Roger Wattenhofer. 28th ACM Symposium on Principles of Distributed Computing (PODC), Calgary, Canada, August 2009.

14. A Log-Star Distributed Maximal Independent Set Algorithm for Growth-Bounded Graphs Johannes Schneider and Roger Wattenhofer. 27th ACM Symposium on Principles of Distributed Computing (PODC), Toronto, Canada, August 2008.