Exact bounds for distributed graph colouring

Joel Rybicki

Helsinki 18 May 2011

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Joel Rybicki

Työn nimi — Arbetets titel — Title

Exact bounds for distributed graph colouring

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
M. Sc. Thesis	18 May 2011	90 pages

Tiivistelmä — Referat — Abstract

A distributed system is a collection of networked autonomous processing units which must work in a cooperative manner. Currently, large-scale distributed systems, such as various telecommunication and computer networks, are abundant and used in a multitude of tasks. The field of distributed computing studies what can be computed efficiently in such systems.

Distributed systems are usually modelled as graphs where nodes represent the processors and edges denote communication links between processors. This thesis concentrates on the computational complexity of the distributed graph colouring problem. The objective of the graph colouring problem is to assign a colour to each node in such a way that no two nodes connected by an edge share the same colour. In particular, it is often desirable to use only a small number of colours. This task is a fundamental symmetry-breaking primitive in various distributed algorithms. A graph that has been coloured in this manner using at most k different colours is said to be k-coloured.

This work examines the synchronous message-passing model of distributed computation: every node runs the same algorithm, and the system operates in discrete synchronous communication rounds. During each round, a node can communicate with its neighbours and perform local computation. In this model, the time complexity of a problem is the number of synchronous communication rounds required to solve the problem.

It is known that 3-colouring any k-coloured directed cycle requires at least $\frac{1}{2}(\log^* k - 3)$ communication rounds and is possible in $\frac{1}{2}(\log^* k + 7)$ communication rounds for all $k \geq 3$. This work shows that for any $k \geq 3$, colouring a k-coloured directed cycle with at most three colours is possible in $\frac{1}{2}(\log^* k + 3)$ rounds. In contrast, it is also shown that for some values of k, colouring a directed cycle with at most three colours requires at least $\frac{1}{2}(\log^* k + 1)$ communication rounds. Furthermore, in the case of directed rooted trees, reducing a k-colouring into a 3-colouring requires at least $\log^* k + 1$ rounds for some k and possible in $\log^* k + 3$ rounds for all $k \geq 3$.

The new positive and negative results are derived using computational methods, as the existence of distributed colouring algorithms corresponds to the colourability of so-called neighbourhood graphs. The colourability of these graphs is analysed using Boolean satisfiability (SAT) solvers. Finally, this thesis shows that similar methods are applicable in capturing the existence of distributed algorithms for other graph problems, such as the maximal matching problem.

ACM Computing Classification System (CCS):
F.2.2 [Analysis of algorithms and problem complexity]: Nonnumerical algorithms and problems
C.2.4 [Computer-communication networks]: Distributed systems
F.1.1 [Computation by abstract devices]: Models of computation
F.4.1 [Mathematical logic and formal languages]: Mathematical logic – computational logic

Avainsanat — Nyckelord — Keywords
graph colouring, distributed algorithms, colour reduction, Cole–Vishkin techniques, SAT

Säilytyspaikka — Förvaringsställe — Where deposited
Kumpula Science Library, serial number C-

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction

A distributed system is a collection of networked autonomous processing units which must work in a cooperative manner. Distributed systems are ubiquitous in the modern world with applications ranging from telecommunication networks to distributed database systems and industrial control systems [Pel00]. The field of distributed computing studies what can be computed efficiently in a distributed system.

Traditionally, computability and complexity theory concentrates on questions such as "what can be computed" and "how much resources does the computation need" in a centralized setting: all the input data is seen by a single computational entity which sequentially processes the data. In a distributed system, the situation is different, as there are multiple autonomous processors active simultaneously, and all the data may not be available to every processor.

The computational problems in distributed networks often relate to managing the network such as scheduling the activity of the processors or finding good routing schemes within the network. When the size of the distributed system grows large, it becomes infeasible to use protocols and algorithms which essentially rely on global information about the network. Therefore, it is natural to ask whether the network can be efficiently managed if the processors utilize only local information, that is, information available on "nearby" processors.

A distributed system is usually modelled as a graph where the processors are nodes and communication links between two processors are edges. For example, Figure 1 illustrates the communication graph for a simple distributed system. Thus, various aspects of managing the network can be regarded as graph problems.

This work studies the computational complexity of distributed graph colouring

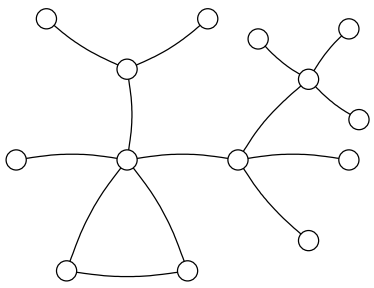


Figure 1: A communication graph of a distributed system. The nodes represent processors and the edges denote communication links between the nodes.

problems. More specifically, the thesis presents exact lower bounds and improved upper bounds for distributed colour reduction in directed cycles and directed rooted trees.

The usual measures for the computational complexity of distributed problems are related to the amount of information exchanged by the processors: *how often* must a processor communicate with others and *how many bits* must the processors exchange in order to solve a problem in the worst case? These measures can be regarded as distributed counter-parts for the time and space complexities in centralized computing [Sip06].

In this thesis, we will concentrate on the synchronous message-passing model of distributed computation. The *time complexity* is measured by the number of communication rounds required to solve the problem and the *message complexity* is measured by the size of the largest message sent. A distributed algorithm which has a running time independent of the size of the network is called a *local algorithm*. The models of distributed computing and related concepts such as locality are discussed further in Section 4.

Colouring the vertices of a graph is a fundamental concept in graph theory dating back to a cartographical problem stated in the 19th century [Cay79]: is it always possible to colour a map divided into separate regions with at most four colours such that no two adjacent regions have the same colour? The problem can be equivalently stated as a *vertex colouring problem* for graphs where nodes are embedded on a two-dimensional plane and edges intersect only at their end-points (the word vertex refers to the alternative name for nodes). Graphs with this property are known as planar graphs. For example, the communication graph illustrated in Figure 2 is planar.

In the vertex colouring problem, the task is to assign a colour for each node in the graph in such a way that nodes connected by an edge have different colours. The colours are usually denoted by non-negative integers. The decision version of the vertex colouring problem (does there exist a colouring that uses at most k colours) was one of the first computational problems shown to be NP-complete [Kar72]. That is, no efficient (polynomial-time) algorithm for optimally colouring graphs is known.

In addition to vertex colourings, other graph colouring problems also exist. For example, instead of colouring the nodes of a graph, the edges (i.e., communication links) can be coloured. In this case, edges with a common end-point must have different colours.

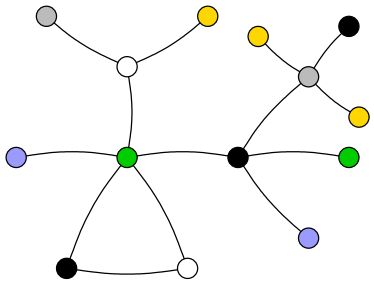


Figure 2: The communication graph given in Figure 1 with a proper 5-colouring of its vertices.

Graph colourings have numerous applications. For example, the vertex colouring problem is closely related to various scheduling and resource allocation problems in communication networks [ABLP89, BBH$^+$98, Ram99]. Vertex colourings also have applications in optimization problems related to compiler design [CAC$^+$81].

In the realm of distributed computing, graph colouring algorithms are also used as subroutines for other algorithms [PR01, ÅS10]. Furthermore, many algorithms designed for stronger distributed models can be used in weaker models if the communication graph is coloured. Of course, the graph colouring problems are also of theoretical interest. Therefore, it is natural that the graph colouring problem has received considerable attention in the field of distributed and parallel computing [CV86, GP87, GPS88, Nao91, Lin92, KW06, FGIP07, BE09, Kuh09, BE10a].

It is known that directed cycle graphs cannot be coloured with three colours in a constant number of communication rounds [Lin92]. However, a classic algorithm originally due to Cole and Vishkin [CV86] can reduce the number of colours from k to $\mathcal{O}(\log k)$ in a single communication round. Repeated applications of this algorithm allow fast colour reduction in cycles and trees.

This work studies how well a distributed algorithm can reduce the number of colours in a constant number of rounds. We derive *exact* values for various cases of distributed colour reduction parametrized by the type of colouring, the number of colours in the graph, and the number of communication rounds allowed for the distributed algorithm. Furthermore, these results are used to give an improved algorithm for 3-colouring two fundamental graph families in distributed computing: directed cycles and directed rooted trees.

The classic lower bound for distributed 3-colouring due to Linial [Lin92] states that 3-colouring a directed n-cycle requires at least $\frac{1}{2}(\log^* n - 3)$ communication rounds for all n. The corresponding upper bound established in the literature

is $\frac{1}{2}(\log^* n + 7)$ [CV86, Pel00]. The so-called log-star function $\log^*$ is a very slow growing function. The precise definition is given in Section 3.1.

The main result of this thesis improves both upper and lower bounds for 3-colouring k-coloured directed cycles: 3-colouring a k-coloured directed cycle requires at least $\frac{1}{2}(\log^* k + 1)$ communication rounds for some k and is possible in $\frac{1}{2}(\log^* k + 3)$ communication rounds for all k. Moreover, bounds for k-coloured directed rooted trees are given as well with a lower bound of $\log^* k + 1$ for some $k \geq 3$ and an upper bound of $\log^* k + 3$ for all $k \geq 3$.

The methods used in this work are an example of using computers to derive new upper and lower bounds for the complexity of computational problems. While this idea is not new, this line of research has recently been advocated in the theoretical computer science community [Aar07, Wil08]. For example, in the last few years, this approach has been applied in the context of circuit complexity [KKY09] and time-space lower bounds [Wil07, Wil10]. Prior work and related techniques are discussed in Section 2. This work extends the approach to the field of distributed computing, and in particular, the study of local algorithms [NS95, Suo11].

The thesis is divided into two parts. In the first part, we begin with an overview of related work in computer-aided proofs. After this, we focus on distributed computation: we recall some necessary graph theoretic concepts, define what is a distributed system and a distributed algorithm, describe the models of distributed computation, and give an overview of known algorithms and lower bounds for distributed graph colouring with an emphasis on directed cycles and directed rooted trees.

In the latter part of the thesis, we describe the methods used for deriving new upper and lower bound results for distributed colouring. These results are based on the correspondence between the colourability of certain graphs and the existence of distributed colouring algorithms. For computing the chromatic numbers, we utilize so-called SAT solvers which have been applied previously in solving graph colouring [GS02, Pre04, Van08] and other combinatorial problems [LMS08, JK10].

2 Related work in computer-aided proofs

The idea of using computers in solving mathematical problems is not new. In fact, one can argue that computer science itself studies how to solve such problems efficiently

with computers. A common feature for these problems is the fact that they are often *finite* in their nature. That is, the computer is given some input of finite length and the computer is expected to output an answer of finite length in finite time. Of course, this is inevitable if we wish to restrict ourselves to *effectively calculable* procedures.

However, many mathematical statements concern infinite sets and objects. For example, many statements have the form "for every natural number n there is an object with property A". There are also statements about non-existence of certain mathematical objects such as "there does not exist any algorithm that solves the halting problem for Turing machines". When studying computability, and in particular, what cannot be computed, we need to argue that there does not exist an algorithm with certain properties. For these statements, a naive approach yields a countably infinite search space as there are infinitely many algorithms. Therefore, to perform automated search effectively, the search space has to be limited in some manner.

In this work, we study methods for proving uncomputability results and finding new distributed algorithms for graph problems. While there does not seem to be any prior work in regards to distributed algorithms, there exists a significant amount of studies that use computers to solve problems of similar nature.

A common approach is to interpret some aspect of the problem as a combinatorial problem that can be solved with a (possibly exhaustive) computer search. Therefore, it is not surprising that the field of combinatorics provides many examples of proofs by computers. Perhaps the most well-known examples are the proofs of the non-existence of projective planes of order 10 [LTS89, Lam91] and the Four Colour theorem: every planar graph is four-colourable [AH89, RSST97, Gon08]. However, a large body of similar work predates these celebrated results.

Already during the 1950s computers were used to help in mathematical proofs. An example of such an early result is the proof that finite projective plane of order 8 is unique which was attained by enumeratively searching for 7×7 Latin squares [HSW56]. An early survey of combinatorial search techniques was written by Hall and Knuth in 1965 [HK65]. In the same year, Golomb and Baumert [GB65] wrote an exposition on backtrack programming which is an important tool for exhaustive search. Almost half a century later, Knuth published the fourth volume of his celebrated book series [Knu11]. The first section of the fourth volume contains a comprehensive presentation of basic techniques for enumerative combinatorial search.

In Ramsey theory, computing upper and lower bounds for the values of specific Ramsey numbers is an active area of research [Rad09]. Similarly, many other combinatorial systems have been studied using computational search methods [KÖ06]. Furthermore, algorithmic tools for automatic proving of combinatorial identities have been in use for quite some time [PWZ96].

Conjectures on major open problems, in e.g., number theory, such as Goldbach's conjecture and Riemann hypothesis, have been verified with the help of computer search up to large numbers.

As regards to complexity theory, computers have been used to construct and verify so-called *gadgets* which are often used in computational hardness proofs to reduce a problem into another one. Usually, the construction of these gadgets is guided by intuition and human insight. In the recent years, there have been examples of constructing and verifying these gadgets using computers. For example, Trevisan et al. used a linear programming approach to construct gadgets for hardness of approximation proofs for the maximum 3-SAT, maximum 2-SAT and maximum cut problems [TSSW96]. Mulzer and Rote showed that triangulating a point set such that the sum of edge lengths is minimized is NP-hard [MR08].

In the recent years, so-called SAT solvers — programs for deciding the satisfiability of formulas in propositional logic — have been used to solve various problems ranging from formal verification and model checking to various classical combinatorial problems [GS02, Pre04, Van08, JK10] and genome analysis such as haplotype inference [LMS08]. The positive results seem to stem from the fact that in the past twenty years, SAT solvers have been intensively studied and new solvers are constantly engineered. For example, there exist annual competitions[1] for new state-of-the art SAT solvers.

In addition, SAT solvers have been used to search upper and lower bounds for various logical design problems in contrast to other exhaustive search algorithms [Knu11, Ch. 7.1.2]. For example, Kamath et al. [KKRR93] introduced a way to translate a problem of finding minimal canonical forms (algebraic sum-of-product expressions) of certain Boolean functions into a SAT instance. Estrada [Est03] later experimented with the transformations given by Kamath et al. and proposed a procedure for synthesizing and minimizing logical circuits with SAT solvers.

Williams together with Woo [Wil08] proposed a research program for computational search of small circuits in an effort to understand Boolean circuit complexity better:

[1] http://www.satcompetition.org/ (February 20th, 2011)

catalog the smallest known circuits for various problems on small input sizes hoping to gain new insight on the circuit complexity of some problems such as matrix multiplication. Williams argues that while truly minimal examples are most likely hard to come by, concrete examples of almost-minimal circuits could still be useful by either as inspiration for theoreticians, or for computers as example patterns for machine learning techniques. This approach has been dubbed as *experimental complexity theory* [Aar07].

Of course, finding small circuits computationally has been studied already before. During the 1960s, provably optimal circuits were attained via exhaustive search using computers [Hel63]. In addition, classic text books on circuit complexity [Weg87] have emphasized the importance (and difficulty) of designing small circuits. However, with the increase in computational power and availability of high-performance computers, there are hopes that such difficult problems can be attacked in a more systematic way and on a much larger scale than before.

Kojevnikov et al. [KKY09] contributed by studying the use of SAT solvers for constructing small Boolean circuits for symmetric Boolean functions. In particular, they studied the class of MOD-functions. The $\mathrm{MOD}^n_{m,r}$ functions are defined as

$$\mathrm{MOD}^n_{m,r}(x_1, \ldots, x_n) = 1 \text{ iff } \sum_{i=1}^{n} x_i \equiv r \pmod{m}.$$

Kojevnikov et al. found better upper bounds for circuits computing the $\mathrm{MOD}^n_{3,r}$ functions on two different circuit models.

Another benefit of the SAT solver approach is that it also yields lower bounds. The formulas are often satisfiable if and only if a circuit (or an algorithm) with the given constraints, e.g., certain number of logic gates, exists. If the formulas are unsatisfiable, then we know that no such circuit exists. Unfortunately, it seems that the unsatisfiable cases are very difficult to solve with current SAT or QBF (quantified Boolean formula solvers) solvers [Wil08, KKY09].

In addition to SAT, linear programming solvers have also been used to construct proofs for time lower bounds. Williams used a so-called alternation-trading method [Wil07, Wil08, Wil10] to show both deterministic and non-deterministic time-space lower bounds for NP- and coNP-hard problems such as SAT, Hamiltonian path, vertex cover, and tautologies [GJ79]. The method is based on solving a series of linear programs. For example, a proof that SAT problem cannot be solved by an algorithm running in $\mathcal{O}(n^{1.6})$ time and $n^{o(1)}$ space was found using a computer search. For a detailed discussion of this type of automated search for proofs, see [Wil07, Ch. 5].

Computer-aided searches and proofs have also been used to improve the analyses of running times for exact exponential time algortihms [FK10, Ch. 6]. For example, Fomin et al. have analysed recursive backtracking algorithms (also known as search-tree algorithms) for maximum independent and minimum dominating sets [FGK05, FGK09]. These backtracking algorithms are recursively applied to solve subproblems using two different types of rules. Reduction rules are used to simplify the problem instance, whereas branching rules are used to choose a suitable subproblem to be solved.

The upper bound analyses of such algorithms are usually based on bounding the number of nodes in the search tree generated by the algorithm. The running time of the algorithm is given by a collection of linear recurrences for each different rule. Given such a collection of recurrences for the running time of a backtracking search algorithm, Eppstein's method [Epp06] for analysing multivariate recurrence equations can be used to find suitable coefficents for the recurrences which minimize the running time. Eppstein's method is based on solving generalization of linear programs called quasiconvex programs [Epp05].

Computers have not just been used to analyse but also to create new algorithms. For example, Gramm et al. [GGHN04] and Fedin and Kulikov [FK06] give programs for finding and analysing new search tree algorithms for NP-hard problems. Both frameworks utilize the observation that an upper bound can be proved by considering all bounded size subproblems. Thus, the search space of algorithms can be restricted to a finite set.

In the case of local distributed algorithms and colour reduction algorithms, we can limit the search of algorithms into a finite problem, as distributed colouring algorithms correspond to colourability of certain finite graph structures. This correspondence is discussed in Section 6.3.

3 Preliminaries

In this section, we introduce the notation used throughout this work and give the definitions for various basic graph theoretic concepts, such as graphs, subgraphs, neighbourhoods, independent sets and colourings. We also review the computational complexity of graph colouring. Some knowledge of basic complexity theory is assumed. For a reference on complexity theory, see for example the introductory textbook by Sipser [Sip06].

3.1 Sets and functions

We begin with defining some set theoretic notation. The set of natural numbers is denoted by $\mathbb{N} = \{0, 1, \dots\}$ and the set of positive natural numbers is $\mathbb{N}^+ = \mathbb{N}\backslash\{0\}$. For any $k \in \mathbb{N}^+$ the set of the first k natural numbers is denoted by $[k] = \{0, 1, \dots, k-1\}$. The size of a finite set A is denoted by $|A|$. The set of k-subsets of set A is

$$\binom{A}{k} = \{B \colon B \subseteq A \text{ and } |B| = k\}$$

and the set of k-tuples of A is

$$A^k = \{(a_0, a_1, \dots, a_{k-1}) \colon a_i \in A \text{ where } i \in [k]\}.$$

Multisets are a generalization of sets. In a *multiset* A, every element in the multiset is assigned a multiplicity $m \colon A \to \mathbb{N}$. The value $m(a)$ denotes how many times the item a is contained within the set A. For example, the multiset $\{x, x, y, y, z\}$ contains elements x and y two times while the element z occurs once. A *k-colouring of a set A* is a mapping $c \colon A \to [k]$. An n-ary *relation* over A is a set $R \subseteq A^n$.

Furthermore, there are two important functions that arise frequently when analysing the running times of distributed algorithms: the *iterated logarithm* and the *log-star* functions. All logarithms are to base 2 unless otherwise specified. That is, $\log x = \log_2 x$.

Definition 3.1. Let $i \in \mathbb{N}$. The *iterated logarithm* $\log^{(i)} x$ is defined inductively by

$$\log^{(0)} x = x,$$
$$\log^{(i+1)} x = \log(\log^{(i)} x).$$

Definition 3.2. The *log-star* function $\log^* x$ is defined as

$$\log^* x = \begin{cases} 0 & \text{if } x \leq 1, \\ 1 + \log^* \log x & \text{otherwise.} \end{cases}$$

The two functions are closely related. Observe that if $i = \log^* x$ then $\log^{(i)} x \leq 1$. In addition, the log-star function grows extremely slowly; for all practical values of k the value $\log^* k$ is at most 7. For example, $\log^* 2 = 1$, $\log^* 4 = 2$, $\log^* 16 = 3$, $\log^* 65536 = 4$, and $\log^* 2^{65536} = 5$.

Similarly to the iterated logarithm, we use the following notation for other iterated functions.

Definition 3.3. Let A be a non-empty set, $f\colon A \to A$, and $i \in \mathbb{N}$. The *iterated function* $f^{(i)}$ is defined inductively by

$$f^{(0)}(a) = a,$$
$$f^{(i+1)}(a) = (f^{(i)} \circ f)(a).$$

3.2 Graphs

We will now define some central concepts of graph theory used throughout this work. Most of the following graph theoretic terminology follows the ones presented in standard textbooks of graph theory [Die10].

A *graph* is denoted by a pair $\mathcal{G} = (V, E)$ where V is the set of *nodes* (also called *vertices*) and E is the set of *edges*. The *order* of the graph is $|\mathcal{G}| = |V|$, that is, the number of nodes in the graph. While it is often assumed that the nodes are a subset of natural numbers, the nodes can be of any type, such as sets or tuples.

In this work, we consider both undirected and directed graphs. The edge set for an *undirected graph* is a set $E \subseteq \binom{V}{2}$ of 2-subsets of the set V of nodes. Two nodes u and v are said to be adjacent if $\{u, v\} \in E$.

In a *directed graph*, the edge set $E \subseteq V \times V$ is a set of ordered pairs. In a directed graph, nodes u and v are adjacent if either $(u, v) \in E$ or $(v, u) \in E$ holds. An edge (u, v) is said to be *outgoing* for u and *incoming* for v. In addition, the node u is sometimes called the *predecessor* of v, while the node v is the *successor* of u.

Observe that an undirected graph $\mathcal{G} = (V, E)$ always corresponds to a directed graph $\mathcal{G}' = (V, E')$ where $\{u, v\} \in E \iff (u, v), (v, u) \in E'$. Thus in both directed and undirected graphs, the edge set E can be considered as a binary adjacency relation over the set of nodes. For undirected graphs, the relation is always symmetric.

The nodes u and v connected by an edge e are said to be the *end-points* of e. Two edges e and f are said to be adjacent if there exists a node $v \in V$ such that both e and f have v as an end-point. A node v is *incident* to the edge e if node v is an end-point of e.

The *degree* of a node $v \in V$ is the number of edges that have v as an end-point and is denoted by $\deg(v)$. In an undirected graph, the degree of a node v is the size of the set $\{\{u, v\} \in E\}$. For directed graphs the *in-degree* of a node v is the size of the set of incoming edges $\{(u, v)\colon (u, v) \in E\}$ and the *out-degree* of v is the number of outgoing edges connected to v. In directed graphs, the degree of a node is the sum

of the in- and out-degrees of the node. A node v is said to be *isolated* if $\deg(v) = 0$. The *maximum degree* $\Delta(\mathcal{G})$ of a graph $\mathcal{G}$ is the maximum number of edges incident to any node $v \in V$, that is, $\Delta(\mathcal{G}) = \max\{\deg(v) : v \in V\}$. For brevity, we denote the maximum degree simply as Δ when the graph $\mathcal{G}$ is clear from the context.

All the graphs considered in this work are *simple*: there are no self-loops (an edge from a node to itself) or multiple edges between nodes. In other words, the adjacency relation E is irreflexive and E is a set in contrast to a multiset.

Since any undirected graph corresponds to a symmetric directed graph, we only give the following definitions for directed graphs. Let $\mathcal{G} = (V, E)$ and $\mathcal{G}' = (V', E')$ be graphs. The graphs $\mathcal{G}$ and $\mathcal{G}'$ are said to be *isomorphic* if there exists a bijection $\pi\colon V \to V'$ such that $(u, v) \in E$ if and only if $(\pi(u), \pi(v)) \in E'$. If $V' \subseteq V$ and $E' \subseteq E$, then $\mathcal{G}'$ is a *subgraph* of $\mathcal{G}$. A graph $\mathcal{G}$ is said to be $\mathcal{G}'$*-free* if $\mathcal{G}$ does not contain a graph isomorphic to $\mathcal{G}'$ as a subgraph. For example, *triangle-free graphs* do not contain a complete graph of 3 vertices as a subgraph.

A subgraph $\mathcal{G}'$ of graph $\mathcal{G}$ is said to be *induced* by the node set $V' \subseteq V$ if for all $u, v \in V'$ there is an edge $(u, v) \in E'$ if and only if $(u, v) \in E$. That is, if there is an edge between two nodes in the original graph, then there is also an edge between the nodes in the induced subgraph. Moreover, no new edges are introduced. A subgraph of $\mathcal{G}$ induced by the node set $A \subseteq V$ is denoted by $\mathcal{G}[A]$.

A path of length k is a non-repeating sequence $(v_0, \ldots, v_k) \in V^{k+1}$ such that for all $i \in \{0, 1, \ldots, k-1\}$ it holds that $\{(v_i, v_{i+1}), (v_{i+1}, v_i)\} \cap E \neq \emptyset$. The distance $\mathrm{dist}_{\mathcal{G}}(u, v)$ of two nodes u and v in graph $\mathcal{G}$ is the length of the shortest path from u to v. The *diameter* of graph $\mathcal{G}$ is the maximum distance of any two nodes in the graph and is denoted by $\mathrm{diameter}(\mathcal{G})$.

A node u is said to be a neighbour of node v if they are connected by an edge. The the set of neighbours for u is $\{v\colon (u, v) \in E \text{ or } (v, u) \in E\}$.

Definition 3.4. Let $\mathcal{G} = (V, E)$ be a graph and $r \in \mathbb{N}$. The *radius-r neighbourhood* of a node $v \in V$ is defined as the set

$$B_{\mathcal{G}}(v, r) = \{u : \mathrm{dist}_{\mathcal{G}}(u, v) \leq r\}.$$

When the graph is clear from the context, we omit the subscript $\mathcal{G}$ in $B_{\mathcal{G}}$ and dist_G. Finally, we say that a graph is *connected* if for all $u, v \in V$ such that $u \neq v$, there is a path between u and v.

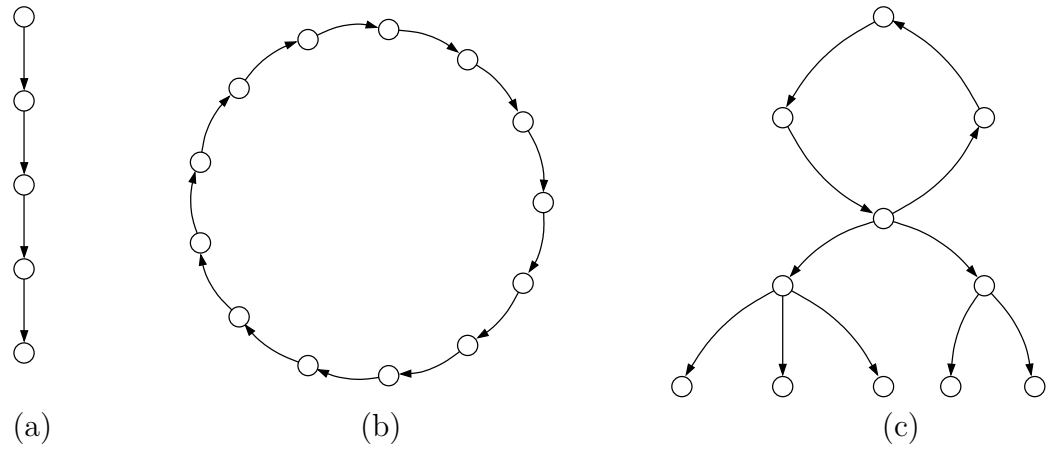


Figure 3: Various graphs. (a) A directed path $\mathcal{P}_4$. (b) A directed cycle $\mathcal{C}_{13}$. (c) A directed pseudotree. Notice that removing any edge from the directed cycle yields a directed rooted tree. The union of these three graphs is a pseudoforest.

3.3 Fundamental graph families

We now give the definitions of the graph families considered throughout this work.

Definition 3.5. Let $\Delta \in \mathbb{N}$. The family of *bounded-degree graphs* of degree Δ is

$$\{\mathcal{G} = (V, E) : \text{for all } v \in V \text{ it holds that } \deg(v) \leq \Delta\}.$$

Definition 3.6. A *directed n-cycle* $\mathcal{C}_n = (V, E)$ is a graph on n nodes such that

$$V = \{v_0, v_1, \ldots, v_{n-1}\}$$

and the edges have a globally consistent orientation such that

$$E = \{(v_i, v_{i+1}) : i \in \{0, 1, \ldots, n-2\}\} \cup \{(v_{n-1}, v_0)\}.$$

A cycle is *even* if n is even and *odd* otherwise.

Definition 3.7. A *tree* is a connected graph without cycles. A *directed rooted tree* is a tree where each node has an in-degree of at most one. The *root* is a node with in-degree of zero.

Definition 3.8. A *directed pseudotree* is a connected, directed graph $\mathcal{G}$ such that the in-degree of each node is at most one. That is, $\mathcal{G}$ may be a directed rooted tree, directed cycle or a directed rooted tree with at most one directed cycle.

A graph consisting of vertex-disjoint trees is called a *forest*. Similarly, a pseudoforest consists of one or more vertex-disjoint pseudotrees. A *directed path* is a directed rooted tree where each node has an in-degree and out-degree of at most one. Figure 3 illustrates a directed path, cycle, and a pseudotree. Finally, a graph $\mathcal{G} = (V, E)$ is *d-regular* if for all $v \in V$ it holds that $\deg(v) = d$. For example, cycles are 2-regular.

3.4 Graph-theoretic problems

In combinatorics, a colouring refers to a mapping from a set S of *objects* to a set $[r]$ of *colours*. For graphs, the set of colourable objects is usually either the set of vertices, the set of edges, or both, Furthermore, graph colourings are often assumed to have some additional properties. To emphasize these additional properties, the word "colouring" is used in conjunction with suitable descriptive adjectives, such as proper, defective, total, and so on.

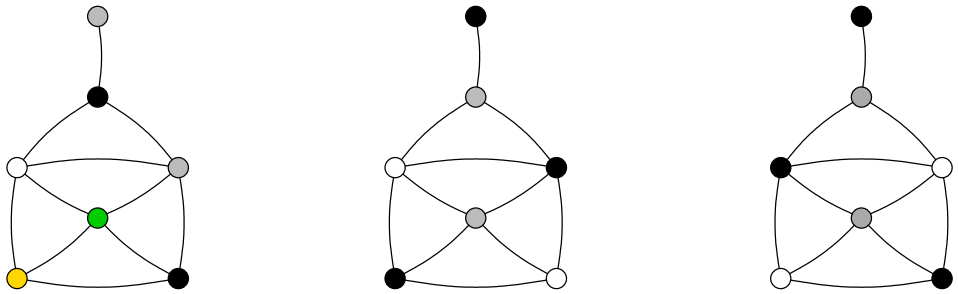
Definition 3.9. A *proper vertex k-colouring* of a graph $\mathcal{G} = (V, E)$ is a mapping $\varphi\colon V \to [k]$ such that for each edge $(u, v) \in E$ it holds that $\varphi(u) \neq \varphi(v)$. That is, φ assigns a colour for each node $v \in V$ such that no two adjacent nodes are given the same colour.

Definition 3.10. An *independent set* in a graph $\mathcal{G} = (V, E)$ is a set $I \subseteq V$ such that for all $u, v \in I$ it holds that $(u, v) \notin E$. That is, no two nodes in the independent set are adjacent. An independent set I is *maximal* if it is not a proper subset of any other independent set. A largest independent set is called a *maximum* independent set. The size of a maximum independent set in a graph $\mathcal{G}$ is denoted by $\alpha(\mathcal{G})$.

The *chromatic number* $\chi(\mathcal{G})$ of a graph $\mathcal{G}$ is the smallest number of colours required to properly colour the vertices of graph $\mathcal{G}$. A graph is k-colourable if the chromatic number is at most k, that is, $\chi(\mathcal{G}) \leq k$. In a k-colourable graph, for each colour $c \in \{0, 1, \ldots, k-1\}$, the colour class $I_c = \{v \in V\colon \varphi(v) = c\}$ is an *independent set* in the graph.

Figure 4 illustrates some examples of proper vertex colourings. The graph depicted in the figure has a chromatic number of 3. Figures 4b and 4c show that the graph can be properly vertex coloured with three colours. However, the graph contains odd cycles which are not two-colourable.

Cliques are a dual concept to independent sets. Cliques can be used, for example, to lower bound the chromatic number of a graph.



(a) (b) (c)

Figure 4: Examples of vertex colourings. (a) A proper 5-colouring. (b) A proper 3-colouring. (c) Another proper 3-colouring.

Definition 3.11. A *k-clique* in an undirected graph $\mathcal{G} = (V, E)$ is a set $U \subseteq V$ such that $|U| = k$ and for all nodes $u, v \in U$ there exist edges $\{u, v\} \in E$ if $u \neq v$. The *complete graph* $\mathcal{K}_n = (V', E')$ on n vertices is a graph such that V' is an n-clique. The size of a largest clique in a graph $\mathcal{G}$ is denoted by $\omega(\mathcal{G})$.

The computational problems related to vertex colourings, independent sets, and cliques are hard in general. Given a graph $\mathcal{G}$ and a value k as input, it is NP-complete to decide whether a proper vertex k-colouring of $\mathcal{G}$ exists, if $\mathcal{G}$ has an independent set of size at least k, or if $\mathcal{G}$ has a clique of size at least k [GJ79]. This corresponds to deciding whether $\chi(\mathcal{G}) \leq k$, $\alpha(\mathcal{G}) \geq k$, or $\omega(\mathcal{G}) \geq k$, respectively.

It is easy to see that if a graph contains a k-clique, then the graph cannot be coloured with less than k colours. Unfortunately, lower bounding the chromatic number by finding large cliques is not easy. Finding relatively large cliques efficiently may not be possible as the maximum clique problem is NP-complete and the optimization version is hard to approximate [Vaz01, Ch. 29.6].

Moreover, $\omega(\mathcal{G})$ does not have a direct relation to $\chi(\mathcal{G})$ as there are graph families where the largest clique has a constant size but the chromatic number is arbitrarily large. For example, there exist so-called *Mycielski graphs* such that for every k there is a graph $\mathcal{G}$ such that the graph is not k-colourable and the size of the largest clique is two [Myc57]. The neighbourhood graphs presented in Section 6.3 are another example of a graph family where the size of the largest clique remains constant while the chromatic number grows as a function of the order of the graph.

On the other hand, every graph is $(\Delta + 1)$-colourable as seen by a simple centralized greedy algorithm [Die10, Ch. 5]: iterate over all nodes $V = \{v_0, v_1, \ldots, v_{n-1}\}$ and for each node v_i choose the smallest free colour not assigned to any node in $B(v_i, 1)$.

This produces a proper colouring as after i steps for every $1 \leq i \leq n$, the subgraph induced by the node set $\{v_0, v_1, \ldots, v_{i-1}\}$ is properly coloured. To see that there are situations where the above algorithm has to use $\Delta + 1$ colours, consider for example a complete graph or an odd cycle. If a graph is neither of these, then by Brooks' theorem the graph is Δ-colourable [Bro41].

In addition to proper vertex and edge colourings, various other graph colourings also exist such as total colourings (properly colour both nodes and edges), radius-r colourings (each node v has a unique colour in their radius-r neighbourhood), p-defective colourings (each colour class induces a subgraph with maximum degree at most p), and so on.

Definition 3.12. Let $\mathcal{G} = (V, E)$ be a graph with a proper vertex k-colouring $\varphi\colon V \to [k]$. The colouring φ is a *radius-r colouring* if for all nodes $v \in V$ it holds that

$$|B_{\mathcal{G}}(v, r)| = |\{\varphi(u) : u \in B_{\mathcal{G}}(v, r)\}|.$$

That is, in every radius-r neighbourhood of $\mathcal{G}$, the nodes have unique colours.

Unless otherwise mentioned, the phrase "graph colouring" refers to a proper vertex colouring. The computational complexity of graph colouring problems is further discussed in Section 3.5. In addition to proper vertex colourings, we also discuss the following graph problems which are illustrated in Figure 5. We give the definitions for undirected graphs.

Definition 3.13. A *vertex cover* of a graph $\mathcal{G} = (V, E)$ is a subset $C \subseteq V$ of the nodes such that $\{u, v\} \cap C \neq \emptyset$ for every edge $\{u, v\} \in E$. That is, each edge is covered by some node in C.

Definition 3.14. A *dominating set* of a graph $\mathcal{G} = (V, E)$ is a subset $D \subseteq V$ of the vertices such that for all $v \in V$ either $v \in D$ or there exists a node $u \in D$ such that $\{u, v\} \in E$. That is, all nodes are in the dominating set or adjacent to a node in the dominating set.

Definition 3.15. A *weak k-colouring* of $\mathcal{G} = (V, E)$ is a colouring $c\colon V \to [k]$ such that for each non-isolated node $v \in V$ there exists $\{u, v\} \in E$ such that $c(v) \neq c(u)$. That is, every node has at least one neighbour with a different colour.

If a graph has no isolated nodes, then a weak 2-colouring can be regarded as a partitioning of the node set into two disjoint dominating sets.

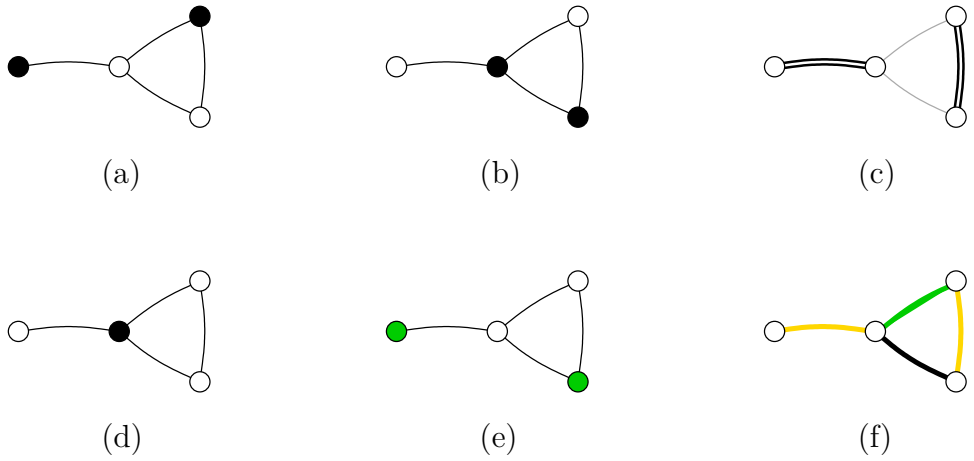


Figure 5: Examples of various graph problems. (a) A maximum independent set. (b) A minimum vertex cover. (c) A maximum matching. (d) A minimum dominating set. (e) A weak 2-colouring. (f) An edge 3-colouring.

Definition 3.16. A *matching* in a graph $\mathcal{G} = (V, E)$ is a subset $M \subseteq E$ of the edges such that no two adjacent edges $e, f \in E$ are both in M. A *maximal matching* M is a matching such that every edge $e \in E$ is either in the matching M or is adjacent to another edge in M.

Definition 3.17. An *edge k-colouring* of graph $\mathcal{G} = (V, E)$ is mapping $c\colon E \to [k]$ such that no two adjacent edges $e, f \in E$ have the same colour.

Observe that matchings are edge sets analogous to independent sets. Moreover, while independent sets were related to vertex colourings, matchings are closely related to edge colourings.

Somewhat surprisingly, while the vertex chromatic number of graphs can range from 1 (a graph with no edges) to $\Delta + 1$ (a complete graph), the smallest number of colours to properly colour the edges of a graph is always either Δ or $\Delta + 1$. This is known as Vizing's theorem [Die10, Ch. 5.3]. Despite this fact, deciding whether a graph can be edge coloured with Δ colours or not is still NP-complete [Hol81].

Observe that a trivial independent set or a matching is just an empty set whereas a trivial vertex cover or a dominating set is the set of all vertices. Therefore, from a computational perspective, finding vertex covers and dominating sets are minimization (or covering) problems, whereas finding independent sets and matchings are maximization (or packing) problems. For these problems, finding the size of the minimum (or maximum) solution is NP-hard [GJ79] with the exception of maximum matchings which can be computed in polynomial time [PS98, Ch. 10].

3.5 Computational complexity of graph colouring

This section gives an overview on the results regarding the complexity of centralized vertex colouring problems. We begin by defining the three important variants of the vertex colouring problem: the decision, optimisation, and search problems.

Problem 1 (k-colourability problem)**.** Given a graph $\mathcal{G}$ and $k \in \mathbb{N}^+$ as input, decide whether there exists a proper vertex k-colouring for $\mathcal{G}$.

Problem 2 (Chromatic number problem)**.** Given a graph $\mathcal{G}$ find the smallest k such that graph $\mathcal{G}$ is k-colourable. That is, compute the chromatic number $\chi(\mathcal{G})$.

Problem 3 (Minimum colouring problem)**.** Given a graph $\mathcal{G}$ as input, output a proper vertex $\chi(\mathcal{G})$-colouring φ of the graph $\mathcal{G}$.

An α-approximation algorithm for the minimum colouring problem outputs a proper vertex colouring which uses at most $\alpha \cdot \chi(\mathcal{G})$ colours where $\alpha \geq 1$ is the approximation factor. Similarly, an α-approximation algorithm for the chromatic number problem outputs a value $k \in \mathbb{N}$ where $\chi(\mathcal{G}) \leq k \leq \alpha \cdot \chi(\mathcal{G})$.

The computational complexity of various graph colouring problems has been studied intensively during the last forty years. In fact, determining whether a graph is k-colourable for $k \geq 3$ was one of the first problems shown to be NP-complete [Kar72]. Thus, no polynomial-time algorithms for solving k-colourability or the chromatic number problems are currently known. On the other hand, there exists an efficient algorithm for deciding whether a graph is 2-colourable (or *bipartite*). Testing bipartiteness and finding a 2-colouring can be done in $\mathcal{O}(|V| + |E|)$ time with a simple breadth-first search algorithm [CLRS01]: start at an arbitrary node $v \in V$ and proceed in a breadth-first manner marking alternating levels as black and white.

While graph colouring is hard for general graphs, some restricted graph families admit polynomial-time algorithms for computing the chromatic number. For example, there are polynomial-time algorithms for solving the search problem in restricted graph families, such as perfect graphs [GLS93, Ch. 9.4] and graphs with bounded tree-width and clique-width (a generalization of tree-width). Unfortunately, these algorithms are often impractical as they rely on the numerically unstable ellipsoid method or require hard to compute algebraic expressions for describing the input graph.

Kobler and Rotics [KR03] give polynomial-time (in the size of the graph) algorithms for a generalization of the k-colouring problem known as list k-colouring and the chromatic number problem in bounded clique-width graphs. However, the running

time is exponential in the clique-width of the graph and approximating clique-width is known to be hard [FRRS09].

Whereas colouring planar graphs optimally is NP-hard already when restricted to planar graphs with maximum degree of four [GJS74], planar graphs admit polynomial-time algorithms for computing almost-optimal solutions. A planar graph can be 4-coloured in $\mathcal{O}(n^2)$ time [RSST96] and 5-coloured in linear time [CNS81].

Currently the algorithms for k-colourability and the chromatic number problems in general graphs with best asymptotic running times require both exponential time and space [Koi06, BH06]. These algorithms are based on evaluating algebraic inclusion-exclusion expressions which count the number of ways to partition $\mathcal{G}$ in to k different independent sets. This can be done in $\mathcal{O}(2^n n^2)$ time and $\mathcal{O}(n2^n)$ space. However, it is possible to trade time for space and compute the chromatic number in $\mathcal{O}(2.247^n)$ time and polynomial space [BH06]. Recently, Björklund et al. further refined the result and showed that it is possible to compute the chromatic number in 2^n time up to polynomial factors and in space $\mathcal{O}(1.443^n)$ [BHKK10]. A recent exposition of these techniques can be found in the textbook by Fomin and Kratsch [FK10, Ch. 4].

In the past decades, the field of approximation algorithms [Vaz01] has attained wide interest in the hopes of finding algorithms producing relatively good or almost-optimal solutions to NP-hard problems. Not unlike other hard combinatorial optimization problems, the approximability of minimum colouring problem has received a lot of attention. Garey and Johnson provided one of the first inapproximability results regarding the chromatic number problem: unless P = NP, there does not exist a polynomial-time constant-factor approximation algorithm for the problem [GJ76]. However, it is possible to compute a 2-approximation of $\chi(\mathcal{G})$ in $\mathcal{O}(1.3998^n)$ time and a $(1+\varepsilon)$-approximation for all $\varepsilon > 0$ in time and space $\mathcal{O}(1.221^n + 2.247^{e^{-\varepsilon}n})$ [BH06].

During the 1990s, new inapproximability results gradually emerged. Lund and Yannakakis [LY94] reduced the problem of approximating inpependent sets into approximating chromatic number and proved that there exists a positive constant $\varepsilon > 0$ such that the chromatic number cannot be approximated to within factor $|V|^{\varepsilon}$ unless P = NP. Later, Bellare et al. [BGS98] showed that if P $\neq$ NP, the chromatic number cannot be approximated within a factor of $|V|^{\frac{1}{7}-\varepsilon}$ for any positive constant $\varepsilon > 0$. Furthermore, if coRP $\neq$ NP holds, then the chromatic number cannot be approximated within a factor of $|V|^{\frac{1}{5}-\varepsilon}$. Feige and Kilian [FK98] showed that if coRP $\neq$ NP, the chromatic number cannot be approximated within factor $|V|^{1-\varepsilon}$ for any constant $\varepsilon > 0$.

4 Distributed computing

In this section, we define what is a *distributed system* and a *distributed algorithm* in the context of this thesis. In this work, the basic model of distributed computation is the synchronous message-passing model. This model is also known as the local model or Linial's model [Lin92, Pel00].

We begin with a discussion on the relationship between the network structure and the input of the distributed system. We then examine some important variants of the basic message-passing model: the model with unique identifiers, the port-numbering model, and the broadcast model. For the latter two models, we will also discuss the benefits gained by assuming a proper vertex colouring as part of the input. Finally, we formalise the notions of distributed algorithms and the locality of computation.

4.1 The structure of a distributed system

A distributed system is represented by a graph $\mathcal{G} = (V, E)$ where V is the set of nodes and E the set of edges. The nodes correspond to computational entities and the edges represent communication links between two nodes. The graph $\mathcal{G}$ is called the *communication graph* of the distributed system.

A node $v \in V$ can directly send a message to another node u only if they are connected by an edge. Unless otherwise specified, the communication graph is simple and communication links are always symmetric: if v can send a message to u then u can send a message to v.

All nodes in the network run the *same* algorithm. In addition, each node v may receive local input, such as a unique identifier, weight, colour, some other information, or any combination of these. While the communication graph $\mathcal{G}$ represents the *structure* of the distributed system, it also acts as *input* for the distributed algorithm. As we will see, many problems in distributed computing are related to the structure of the graph. Usually, distributed algorithms compute and output some property of the communication graph.

The distributed system operates in discrete synchronous rounds each consisting of three steps where each node (i) sends messages to its neighbours, (ii) receives messages and (iii) performs local computation. All communication and local computation is both reliable and fault-free. That is, all sent messages always arrive, no processor leaves the network or malfunctions, and no communication link breaks down.

Since the rounds are synchronous, all messages that are sent during step (i) are propagated through the communication links and received by the appropriate nodes before step (ii) begins. The details on how neighbours are addressed is discussed in Section 4.2. On the first round, all the nodes in the network start running the algorithm simultaneously.

These rounds are repeated until every node has computed and declared its output. The output may be, for example, the colour of the node, or a binary value indicating whether the node decided to be a part of a set of nodes (e.g. independent set, dominating set, vertex cover). The output may also relate to edges; in these cases the encoding of the output depends on the exact variant of the model at hand.

As regards to the model of local computation used by the nodes, we assume that the nodes are capable of computing any recursive function. That is, while we do not explicitly fix any particular model of computation for the nodes, any Turing-complete model of computation suffices. However, it is mandatory that the local computation of nodes *always halts* at the end of each round. In addition, the nodes can send arbitrary (finite) messages to adjacent nodes. All algorithms discussed in this work are *deterministic*.

In this model, the standard measure of *time complexity* is the maximum number of synchronous communication rounds required for all nodes to compute and declare their output. Any local computation performed by a node in step (iii) is considered free and thus it does not affect the time complexity of a distributed algorithm. Thus, the running time of a distributed algorithm refers to the number of synchronous communication rounds taken by the algorithm to compute the output on all nodes. The *message complexity* of an algorithm is the maximum size of any sent message in bits. However, for the purpose of this work, we assume that the message sizes are unbounded.

Initially, the assumption that the system operates synchronously may seem unrealistic, as real-world distributed networks are usually asynchronous in nature. However, this assumption is reasonable. First, in the worst case, an asynchronous system behaves in a totally synchronous manner. Thus, any lower bound results for synchronous systems directly yield lower bound results for asynchronous systems.

Second, there exist several techniques for converting asynchronous algorithms into synchronous without considerable overhead [Pel00, Ch. 6]. The most common synchronization techniques are the α- and β-synchronizers [Awe85]. The α-synchronizer only has a constant factor overhead when measuring the time complexity and thus

is asymptotically time-optimal. The β-synchronizer is asymptotically optimal with regards to the message complexity but it may increase the running time by a factor proportional to the diameter of the communication graph.

4.2 Models of computation

There are many variations to the basic synchronous message-passing model. Most notable of these are the model with unique identifiers, the port-numbering model, and the broadcast model. Each model differs mainly in how much information is available to the nodes.

Unique identifiers. The model with unique identifiers is often referred to either as Linial's model (after the seminal paper [Lin92]) or as the *local* model [Pel00]. Each node in the graph is given a unique non-negative integer identifier as local input. Usually, the set of used identifiers is the subset of natural numbers $\{0, 1, \ldots, n-1\}$ where $n = |V|$ is the number of nodes in the communication graph.

This model is rather strong as it renders many problems trivial. A good example is the *leader election problem* where the task is to select a single node $v \in V$ as a leader such that the node v knows it has been selected while all other nodes $u \in V \setminus \{v\}$ know that they have not been selected. Solving the leader election problem in this model requires no communication: simply choose the node with identifier 0 as the leader. For avoiding such trivialities, the set of identifiers is sometimes assumed to be any n-subset of the set $\{0, 1, \ldots, \mathrm{poly}(n)\}$. That is, there is only a polynomial bound on the size of the identifiers.

Even with this restriction, the model remains strong. If the communication graph is connected, any computable function f of $\mathcal{G}$ can be computed deterministically in $\mathcal{O}(\mathrm{diameter}(\mathcal{G}))$ communication rounds: each node can collect complete knowledge of $\mathcal{G}$, compute $f(\mathcal{G})$ locally, and output its result. Therefore, it is also interesting to examine more restricted models.

Port-numbering model. In this model, the nodes do not receive any identifiers, and thus, all nodes are *anonymous*. However, each node v can refer to its neighbours by unique numbers $\{1, 2, \ldots, \deg(v)\}$ where $\deg(v)$ is the number of edges connected to the node. With port-numbering, any node can direct messages to a specific neighbour and distinguish which neighbour sent a particular message.

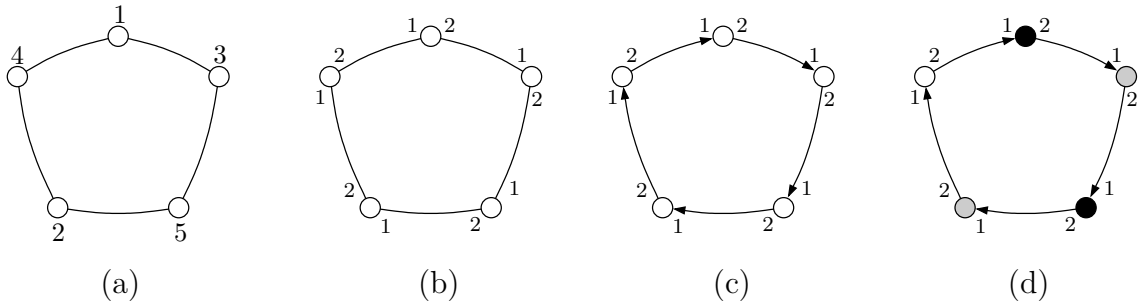


Figure 6: A communication graph isomorphic to a 5-cycle. (a) The graph with unique identifiers. (b) A port-numbering. (c) A port-numbering and an orientation. (d) A proper vertex 3-colouring, port-numbering, and an orientation.

The port-numbering model is known to be restricted; problems such as leader election cannot be solved deterministically [Ang80]. The same applies to graph colouring unless additional information is provided, as we will see later.

A slightly stronger variant of the port-numbering model is the port-numbering model with orientation. An *orientation* is simply an assignment of a direction to each edge in the communication graph. Each node v can partition the port-numbers into two sets: incoming edges and outgoing edges. An oriented edge from u to v is denoted by (u, v). The edge (u, v) is *outgoing* for u and *incoming* for v. Note that while the edges have direction, the communication links remain symmetric. That is, for an oriented edge (u, v) the node v can send a message to node u and vice versa.

Both variants of the port-numbering model are weaker than the model with unique identifiers. For example, consider an anonymous directed cycle $\mathcal{C} = (V, E)$ with port number 1 assigned to the incoming edge and 2 to the outgoing edge as illustrated in Figures 6b and 6c. The graph is highly symmetric: the local neighbourhood $B_{\mathcal{C}}(v, r)$ of each node $v \in V$ is identical for all $r \in \mathbb{N}^+$. Thus, any deterministic algorithm must output the same result for every node in the graph. For this reason, no deterministic algorithm can output a non-trivial subset of nodes, that is, something other than the empty set or the set of all nodes.

While orientation does not help in the case of a symmetric cycle, orientation does partially break the symmetry in graphs where every node has an odd degree. In such graphs, nodes have different number of incoming and outgoing edges. A classical example of utilizing orientation and odd degrees is the weak-colouring algorithm by Naor and Stockmeyer [NS95].

Broadcast model. A natural restriction to the port-numbering model is to leave out the port numbers and orientation. In this setting, the only way of communicating is *broadcasting*: a node must send the same message to all neighbours as there is no way to distinguish different neighbours. Thus, the messages received by a node v is a multiset of size $\deg(v)$ rather than an ordered tuple of length $\deg(v)$.

In contrast to the other models, the broadcast model is quite weak as symmetry cannot be broken without additional information. Perhaps due to this reason, the broadcast model has not received as much attention as the previous models. Nevertheless, there exist some non-trivial positive results when the nodes are given additional information such as a proper colouring [KW06] or an improper colouring [BV01, ÅS10].

The computational power of distributed models. The use of the various models is abundant in the field of distributed computing, and the computational power of different models is an active topic of research. While the broadcast and port-numbering models are in general weaker than the model with unique identifiers, somewhat surprisingly, there are examples of non-trivial problems that are solvable in the broadcast model as well as in the model with unique identifiers.

A recent example is the constant-time approximation of minimum vertex cover in bounded-degree graphs. Given unique identifiers, there does not exist a constant-time distributed algorithm for the minimum vertex cover probelm with approximation factor $2 - \varepsilon$, as this would contradict the fact that there are no constant-factor approximation algorithms for the maximal independent set in cycles [CHW08]. Moreover, no $(2 - \varepsilon)$-approximation algorithm is known in the centralized setting either. However, in bounded-degree graphs, there exists a constant-time 2-approximation algorithm for the minimum vertex cover in the broadcast model [ÅS10].

Anonymous port-numbered k-coloured networks. In this work, we concentrate primarily on the vertex colouring problem which is not deterministically solvable in anonymous networks with port-numbering. Therefore, we assume a stronger variant of the port-numbering model where the all nodes are anonymous but the graph has been properly k-coloured.

More precisely, in this model the distributed system consists of the communication graph $\mathcal{G} = (V, E)$, a port-numbering, and a colouring $\varphi\colon V \to [k]$. Each node $v \in V$ knows only its own colour and the local port-numbering given as input. Some algorithms presented in this work assume that the nodes also know the exact value or

an upper bound on k. Observe that a graph with unique identifiers can be regarded as properly vertex coloured.

Before discussing coloured networks further, let us refine the notion of input and output in a distributed system and give precise definitions for distributed algorithms and local neighbourhoods. After this, we will continue with a short discussion on the benefits of coloured networks in Section 4.5.

4.3 The input and output of a distributed system

As stated in Section 4.1, the communication graph is part of the input for a distributed algorithm. The communication graph may contain information, such as local inputs for nodes, unique identifiers, port-numberings, or orientations for the edges. Input regarding an edge is given to both end-points of the edge.

Frequently, distributed algorithms assume that some global information is also given as input. For example, this may include the number of colours in a coloured graph, the maximum degree, or an upper bound or the exact number of nodes in the graph. Algorithms oblivious to these bounds are called *uniform*. Uniform distributed algorithms and the necessity of this type of global information has been studied by Musto [Mus11].

Furthermore, the distributed problem is often restricted in some way. For example, the algorithms may assume that the communication graph belongs to a particular graph family. Common graph families are trees, bounded-degree graphs, regular graphs, unit-disk graphs or other geometric graphs. The algorithms in Section 5.2 assume that the input graphs are properly coloured directed pseudoforests.

The output of a distributed system is also distributed. For example, a centralized algorithm that solves a decision problem needs to output only one bit to indicate whether to accept or reject the input. Of course, in the distributed setting we cannot assume that all the nodes unanimously agree on the output as this would require network-wide, and hence global, cooperation.

Therefore, the output of a distributed decision problem is defined as follows: the distributed algorithm accepts the input if and only if all nodes accept. Thus, if a single node rejects the input, we say that the distributed algorithm rejects the input. This model for distributed decision problems remedies the situation when the the input is locally feasible but not globally.

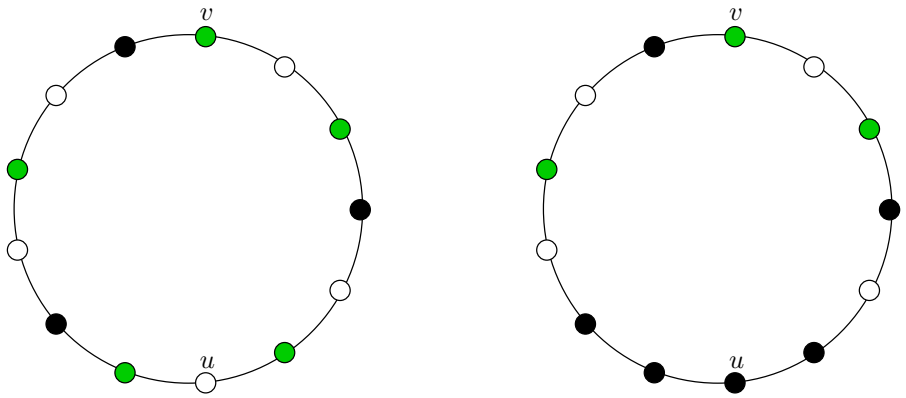


Figure 7: Distributively deciding the validity of a colouring given as input. Assume that the running time of the distributed decision algorithm is at most 5. (a) All nodes accept as the colouring is proper. (b) Node v accepts just as in the previous case but node u must reject as it has equally-coloured neighbours.

As an example of distributed decision problem, consider the problem of verifying whether a given vertex colouring is proper. A one-round algorithm can solve the problem: each node sends its colour to its neighbours and in turn receives the colours of its neighbours. Then the node checks that no neighbour has the same colour.

Figure 7 illustrates the verification scenario. In Figure 7a all the nodes are satisfied with the given colouring. On the other hand, in Figure 7b, most of the nodes remain satisfied as for them the colouring is *locally* proper. However, node u notices that the given colouring is improper, and thus rejects the input whereas node v and its neighbours cannot differentiate the two situations.

In the case of optimization problems, such as finding a large independent set or a proper vertex colouring with few colours, each node outputs its role in the solution. For the independent set problem, the node decides whether it is part of the set or not, and in the vertex colouring problem, the node outputs its new colour.

When the output is related to an edge $\{u, v\}$, it is assumed that both end-points of the edge unanimously agree on the output of the edge. For example, consider the matching problem. Let p_u and p_v be the port-numbers assigned to the edge $\{u, v\}$ by nodes u and v, respectively. If $\{u, v\}$ is part of the matching, then u knows that the edge connected to port p_u is in the matching, and node v knows that the edge p_v is in the matching.

4.4 Distributed algorithms and local neighbourhoods

We will now give some necessary definitions for reasoning about distributed algorithms and their properties. Let $\mathcal{G} = (V, E)$ be a communication graph. Initially, each node $v \in V$ knows *only* its local input. By communicating with other nodes, the node v can gather input from its local neighbourhood.

Let us recall the definition of radius-r neighbourhoods. The radius-r neighbourhood of a node $v \in V$ is the set $B(v, r) = \{u : \mathrm{dist}(u, v) \leq r\}$. The graph $\mathcal{G}[v, r]$ is the subgraph of $\mathcal{G}$ induced by $B(v, r)$ with edges (x, y) and (y, x) removed if $\mathrm{dist}(v, x) = \mathrm{dist}(v, y) = r$. Figure 8 illustrates an example of radius-2 neighbourhoods in two different port-numbered graphs.

If T is the running time of a distributed algorithm, then each node can gather information only from the nodes that are within distance T. Therefore, the output of node $v \in V$ may only depend on the inputs of nodes in $B(v, T)$ and the structure of $\mathcal{G}[v, T]$. Any node $u \notin B(v, T)$ cannot affect the output of v. The running time T of a distributed algorithm is often a function of the number of nodes, the maximum degree, or some other property of the communication graph. In some cases, the running time may be a constant.

Definition 4.1 (Radius-r local view)**.** In a port-numbered graph $\mathcal{G} = (V, E)$, the radius-r *local view* of node v is a tree $\mathcal{V}_{\mathcal{G}}(v, r)$ of depth $r+1$. The root of $\mathcal{V}_{\mathcal{G}}(v, r)$ corresponds to node v and its input. If $r > 0$, then for each neighbour $u_1, u_2, \ldots, u_{\deg(v)}$ of v, the root node has a tree $\mathcal{V}_{\mathcal{G}}(u_i, r-1)$ as a child. The edges leading to children are labeled according to the port numbering of v and the direction of the edge. The tree $\mathcal{V}_{\mathcal{G}}(v, 0)$ consists only of the root node.

Note that in the port-numbering model, it is not necessary to branch again on node v in the subtree $\mathcal{V}_{\mathcal{G}}(u_i, r-1)$. It suffices to recursively branch only on nodes in $B_{\mathcal{G}}(u_i, 1) \setminus \{v\}$. For example, the views in cycles can then be regarded as paths.

Using a simple flooding procedure, a node in the graph can construct its radius-r local view in r communication rounds. The flooding procedure is given in Algorithm 1. During each round, the algorithm gradually updates the local view and sends the updated local view to all neighbours.

While most algorithms are explicitly presented using the send-receive-compute framework, algorithms in this model can also be viewed as mappings from local views to the set of possible outputs. That is, for any deterministic distributed algorithm $\mathcal{A}$ running on node v, there is a corresponding function $f_{\mathcal{A}}$ which maps the local view

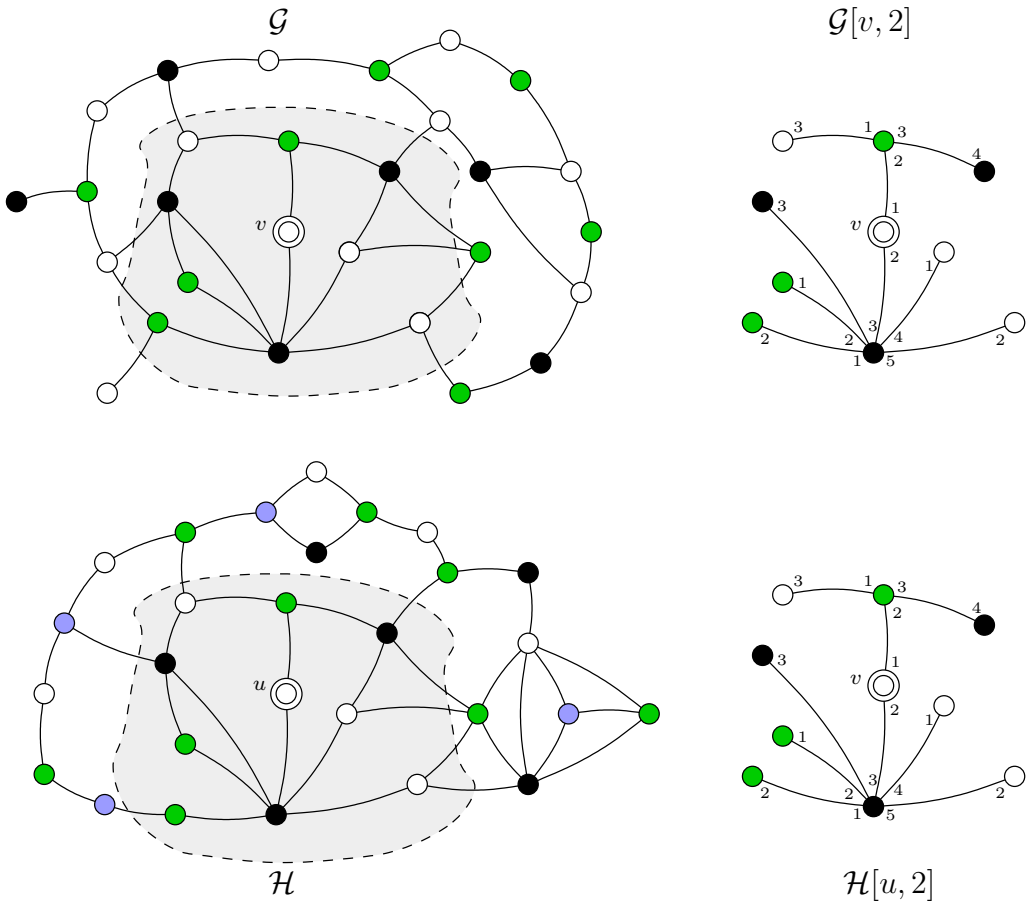


Figure 8: Two different port-numbered graphs $\mathcal{G}$ and $\mathcal{H}$. The radius-2 neighbourhoods of nodes u and v are identical while the graphs $\mathcal{G}$ and $\mathcal{H}$ are not. The neighbourhoods of u and v are highlighted. Observe that $B_{\mathcal{G}}(v,2) = B_{\mathcal{H}}(u,2)$ and $\mathcal{G}[v,2] = \mathcal{H}[u,2]$. An algorithm that runs for $T = 2$ rounds outputs the same result at both nodes u and v.

Algorithm 1 Gather the radius-r neighbourhood

1: Initialize the local view of the node v by setting $x \leftarrow \mathcal{V}(v, 0)$.

2: In sequence, repeat the following for each round $t \in \{1, 2, \ldots, r\}$:

3: Send the message (x, p) to port p for each $p \in \{1, 2, \ldots, \deg(v)\}$.

4: Set $x \leftarrow \mathcal{V}(v, 0)$.

5: For each received message (y, p) where $p \in \{1, 2, \ldots, \deg(v)\}$:

6: Update x: Add y as a child of x. Label the edge from x to y with p.

of v into the output of the algorithm at v. When given unique identifiers, the local view at v can be bijectively folded back into the graph $\mathcal{G}[v, r]$ but for anonymous or coloured graphs this is not generally possible. In a properly coloured graph, different neighbourhoods can map into the same view: consider a 3-coloured infinite path or a large 3-coloured cycle as in Figures 9b and 9c.

In Section 6, we will see that local neighbourhoods and local views prove to be useful when reasoning about the existence of distributed algorithms.

4.5 Coloured networks

Let us now briefly return to the discussion on coloured networks. One goal of the theoretical study of distributed computing is to find the boundaries and limits of different models. Just as in traditional computational complexity theory, often the interesting question is not what can be computed but how much resources are needed to solve a given problem. In centralized computing, the focus is often on the time and space resources required to solve the problem. However, in a distributed setting, the focus is on how much information does the algorithm need in order to solve a problem.

Sometimes all the necessary information can be obtained through communication. In such cases, it is customary to measure how many communication rounds or how many bits need to be sent in order to solve the problem at hand. But communication is not always sufficient. For example, as discussed in Section 4.2, there are many problems that are not possible to solve in anonymous networks with the port-numbering using deterministic algorithms.

To overcome this, some form of symmetry breaking is required. One way to solve this problem is to assign each node a unique identifier. However, unique identifiers are a

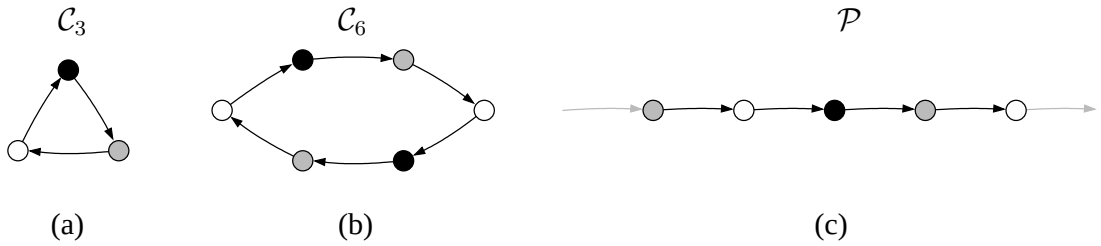


Figure 9: In a coloured communication graph, nodes cannot decide whether the graph is triangle-free. The output must be the same in all the illustrated graphs. (a) A triangle. (b) A 6-cycle. (c) An infinite path.

very strong form of symmetry breaking as all computable graph problems become trivially solvable in time proportional to the diameter of the graph. In this model, it is often interesting to study how much communication is required at the minimum.

Instead of giving globally unique identifiers to all nodes, it often suffices to break the symmetry partially by using locally unique identifiers or almost unique labels such as vertex colours. In a properly k-coloured graph, the distributed algorithm can then use the natural total order of $[k] \subset \mathbb{N}$ to break symmetries since every node has a different colour than its neighbours.

Vertex colourings provide enough information to break symmetry in most cases. Many problems that are not possible to solve in port-numbered anonymous networks can be solved if a vertex colouring is given as part of the input. For example, a proper edge colouring can be computed in one communication round: First, all nodes send their colours and port-numbers to their neighbours. Then for each edge $\{u, v\}$, the nodes u and v choose the colour as follows. Assume that u has a larger colour than v. Let $\varphi(u)$ be the colour of u and $\varphi(v)$ colour of v. Respectively, let p_u and p_v be the port-numbers for the edge $\{u, v\}$. Both nodes then colour the edge with colour $(\varphi(u), \varphi(v), p_u, p_v)$.

The edge colouring is proper since for any two adjacent edges $\{u, v\}$ and $\{v, w\}$ the colour must differ at some component: even if the colours of u and w are the same, the edges have different port numbers in v. Observe that it is critical that the end-points of each edge have different colours. More examples of algorithms in coloured graphs are given in Section 5.

In contrast, while a k-coloured network with port-numbers is a strictly stronger model, it is still weaker than the model with unique identifiers. That is, the model is not *too* strong. Of course, problems depending on fully global information, such as counting the number of nodes in a graph, cannot be solved in coloured networks.

Nevertheless, there are also natural "local problems" that separate the two models. For example, consider the problem of deciding if the communication graph $\mathcal{G}$ is triangle-free. Clearly, the problem is local in the sense that in the unique identifier model, each node u simply gathers its radius-2 view and checks if there is a 3-cycle (u, v, w, u) in $\mathcal{G}[u, 2]$.

However, in a coloured network the nodes cannot distinguish between a 3-coloured $\mathcal{C}_3$ and $\mathcal{C}_{3n}$ for any $n \in \mathbb{N}^+$. We can simply assign the colouring as depicted in Figure 9. Now, in each colour class, all the local radius-r neighbourhoods are identical for all $r \geq 0$. Therefore, if a distributed algorithm rejects the graph illustrated in Figure 9a then it must also reject the other two graphs. Hence, we can argue that coloured networks are strictly weaker model of distributed computation than the model with unique identifiers.

Moreover, in k-coloured networks, the size of the local input can be made independent of the size of the communication graph assuming that the number of colours is independent of the size of the graph. For example, in infinite but bounded-degree graphs, it would not be possible to use algorithms where the running time depends on unique identifiers. Also, the size of the local input would be unbounded.

In a k-coloured graph, the label of each node requires only $\lceil \log k \rceil$ bits. Furthermore, algorithms designed to use unique identifiers often work as-is in properly coloured graphs. This can also improve the running time of the algorithm as the number of colours is usually significantly smaller than the number of unique identifiers.

5 Distributed vertex colouring

In this section, we discuss how to find vertex colourings in a distributive manner. Since a vertex colouring cannot be found in anonymous networks, we assume that some vertex k-colouring is given as input, for example, in the form of unique identifiers. Therefore, to be more precise, we look at *colour reduction* algorithms, that is, algorithms that find a new vertex colouring that uses fewer colours than the original colouring.

We will present three basic algorithmic ideas. First, we present a greedy colour reduction algorithm which is often used as a subroutine in more elaborate colour reduction algorithms. The second algorithm is the so-called Cole–Vishkin colour reduction algorithm [CV86]. The algorithm can be iteratively applied in k-coloured

pseudoforests to find a colouring 3-colouring in $\mathcal{O}(\log^* k)$ rounds. While the Cole–Vishkin algorithm works only in pseudoforests, it can be used as a subroutine to find a $(\Delta+1)$-colouring in bounded-degree graphs in $\mathcal{O}(\Delta^2+\log^* k)$ rounds [GPS88, PR01].

Finally in Section 5.4, we briefly overview some recent positive results for distributed graph colouring which have faster running times with regard to Δ.

5.1 The greedy approach

The greedy algorithm is listed as Algorithm 2. The greedy algorithm sequentially considers each colour class and colours nodes with the smallest unused colour in their neighbourhood. The algorithm can be used in both graphs with unique identifiers and in k-coloured graphs. In addition, the algorithm can also be applied in infinite bounded-degree graphs with a k-colouring as the running time depends only on the value of the highest colour. Figure 10 illustrates the algorithm on a simple graph.

Algorithm 2 Greedy $(\Delta+1)$-colouring from k-colouring

1: In sequence, for each colour c from $k-1$ to $\Delta+1$ do

2: Send colour $\varphi(v)$ to each neighbour.

3: Receive the colour of each neighbour; $A = \{\varphi(u) \colon u \in B(v,1)\}$.

4: If $\varphi(v) = c$ then $\varphi(v) \leftarrow \min\{[k] \setminus A\}$.

To see that the greedy algorithm properly colours the graph, we can look at each colour class in the graph. As each colour class $c \in [k]$ forms an independent set I_c, every node $v \in I_c$ can in parallel choose a new colour in a way that does not conflict with the colour of its neighbours. Once all colour classes have chosen a new colour, the graph has been properly coloured. If the maximum degree Δ of the graph is known, $k-\Delta-1$ communication rounds suffice. Otherwise, the algorithm can be run for k rounds. In each case, the asymptotic running time is $\Theta(k)$.

A similar greedy technique can be applied to other problems as well. Examples of these are various maximal or minimal subsets of nodes or edges, such as maximal independent sets, maximal matchings, minimal dominating sets, and minimal vertex covers.

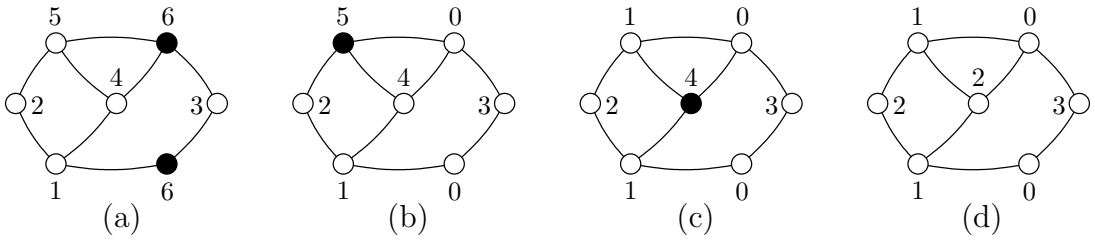


Figure 10: The greedy algorithm running three rounds with $k = 7$ and $\Delta = 4$. (a) The independent set I_6 consisting of nodes with colour 6 activates and every node $v \in I_6$ chooses a new colour. (b) Nodes in the set I_5 choose a new colour. (c) Nodes in the set I_4 choose a new colour. (d) The resulting $(\Delta + 1)$-colouring.

5.2 Cole–Vishkin techniques

While the greedy algorithm works, it is inefficient in large graphs with unique identifiers. In this section, we will introduce a technique for 3-colouring directed cycles originating to Cole and Vishkin [CV86]. This technique was soon extended by Goldberg et al. to $(\Delta + 1)$-colour bounded-degree graphs and to 3-colour rooted trees [GP87, GPS88]. Although the algorithms were developed for parallel processors, they easily translate into the distributed model. We will first describe the basic Cole–Vishkin technique in pseudotrees.

The Cole–Vishkin technique is based on an iterative colour reduction algorithm. The same technique can be used to colour directed cycles, directed rooted trees, and in general, directed pseudoforests. During each communication round, the colouring is reduced from k to $\mathcal{O}(\log k)$ colours by manipulating the bit representations of the colours. Algorithm 3 exploits both the orientation of the edges and the binary encoding of the colours.

In Algorithm 3, every node sends its colour to all its successors. Since the graph is properly coloured, the colour of the predecessor must be different. If the node v has no predecessors, it can simulate a predecessor coloured with the smallest colour in the set $\{0, 1\} \setminus \{\varphi(v)\}$.

After receiving the colour of its predecessor, each node v compares its own colour $\varphi(v)$ to the colour of its predecessor u. As the colours are natural numbers, each colour can be treated as a bit string of length $\lceil \log k \rceil$. We denote such strings as $(b_0, b_1, \ldots, b_{\lceil \log k \rceil - 1})$, where b_0 is the least-significant bit. Now, each node v finds the smallest index i where the bit b_i differs in $\varphi(v)$ and $\varphi(u)$. Then node v chooses the binary encoding of the tuple (i, b_i) as its new colour.

Algorithm 3 $\mathcal{O}(\log k)$-colouring in directed pseudoforests.

1: Send the colour $\varphi(v)$ to all neighbours.

2: Receive the colour $\varphi(u)$ from predecessor u.

3: Compare the binary encodings of $\varphi(v)$ and $\varphi(u)$. Let i be the index of the least-significant bit that differs and b_i be the ith bit of $c(v)$.

4: Set (i, b_i) as the new colour of v.

Lemma 5.1. *In one communication round, a k-coloured directed pseudoforest can be coloured with $2 \cdot \lceil \log k \rceil$ colours.*

Proof. To see that Algorithm 3 computes a proper colouring in a directed pseudoforest, consider the node v and its predecessor u. In the comparison stage, if u chooses the same index i as v then their respective bit values must differ as both u and v have different colours. Otherwise, both nodes choose different indices and thus different colours.

The algorithm reduces a k-colouring into an $\mathcal{O}(\log k)$-colouring in a single round. When comparing two colours, there are $\lceil \log k \rceil$ choices for the differing index i. In addition, the differing bit b_i may have two distinct values. Therefore, the largest colour any node may choose is $2 \cdot \lceil \log k \rceil - 1$. □

Observe that if $k \leq 6$ the above method cannot be used to reduce the number of colours. For example, consider the case that $k = 6$, node u is the predecessor of node v, $\varphi(v) = 5 = 101_2$ and $\varphi(u) = 1 = 001_2$. Since the differing index is $2 = 10_2$ and the differing bit of $\varphi(v)$ has value 1, node v chooses $101_2 = 5$ as its new colour. Therefore, while all pseudotrees are 3-colourable, a different method is needed to reduce the number of colours further if $3 < k \leq 6$.

Nevertheless, the Cole–Vishkin method can be iteratively applied to reduce any k-colouring in a directed pseudotree into a 6-colouring in $\log^* k$ communication rounds.

Lemma 5.2. *A k-coloured directed pseudoforest can be 6-coloured in time $\log^* k$.*

Proof. In one iteration, the Cole–Vishkin algorithm reduces the number of used colours from k to at most $f(k) = 2\lceil \log k \rceil$ colours. Let $6 \leq k \in \mathbb{N}$ and $\ell = \log^* k$. The assumption implies that $\ell \geq 3$. If $\ell = 3$ then the value of k is at most $k \leq 2^{2^2} = 16$. A 16-coloured pseudotree can be 6-coloured in two rounds as $f(f(16)) = f(8) = 6$, thus proving the claim for the case $\ell = 3$.

Assume that $\ell \geq 4$ and let $i \leq \ell - 3$. We will prove by induction an upper bound for the number of colours used after i iterations of the Cole–Vishkin algorithm. The claim is

$$f^{(i)}(k) < 4 \log^{(i)} k.$$

The base case $i = 1$ is trivial as by definition $f(k) = 2\lceil \log k \rceil < 4 \log k$. Assume the induction hypothesis $f^{(i)}(k) < 4 \log^{(i)} k$ holds. For the induction step, we observe that

$$\begin{aligned} f^{(i+1)}(k) &= f(f^{(i)}(k)) \\ &< f(4 \log^{(i)} k) \\ &= 2\lceil \log(4 \log^{(i)} k) \rceil \\ &\leq 2(1 + \log 4 + \log^{(i+1)} k) \\ &= 6 + 2 \log^{(i+1)} k. \end{aligned}$$

Since $i \leq \ell - 3$ we know that $\log^{(i+1)} k \geq \log^{(\ell-2)} k \geq 4$. Therefore,

$$6 < 2 \cdot 4 \leq 2 \log^{(i+1)} k$$

holds and we can finish the induction step with $6 + 2 \log^{(i+1)} k < 4 \log^{(i+1)} k$.

After $\ell - 3$ iterations the number of colours is at most

$$f^{(\ell-3)}(k) < 4 \log^{(\ell-3)} k \leq 4 \cdot 16 = 64.$$

The remaining three rounds suffice to drop the number of colours to

$$f^{(3)}(64) = f^{(2)}(12) = f(8) = 6. \qquad \square$$

Iterating the basic Cole–Vishkin method $\log^* k$ times yields a 6-colouring in any k-coloured directed pseudotree. To drop the number of colours to three, we utilize a colour shifting algorithm and a variant of the greedy colour reduction algorithm.

The shifting algorithm given in Algorithm 4 shifts the colours from the root towards the leaves. This ensures that for any node v the neighbours of v use at most two different colours. That is, $\mathcal{G}[v, 1]$ uses at most three colours. As before, if node v has no predecessor, it can simulate one by choosing the smallest colour that is different from $\varphi(v)$.

After one iteration of the Algorithm 4, the children of every node v have the same colour. The new colouring is also proper since before shifting, the parent $p(v)$ of

Algorithm 4 Colour shifting

1: Send $\varphi(v)$ to all children.

2: Receive a colour from the parent and store it into $p(v)$.

3: Store the old colour: $\varphi'(v) \leftarrow \varphi(v)$.

4: Update the current colour by setting $\varphi(v) \leftarrow p(v)$.

Algorithm 5 Reduce colours by one

1: Send $\varphi(v)$ to all children.

2: Receive a colour from the parent and store it into $p(v)$.

3: If $\varphi(v) = k - 1$, choose a new colour from the set $\{0, 1, 2\} \setminus \{\varphi'(v), p(v)\}$.

node v had a different colour for all v. Thus after shifting, the children of v take the old colour of v while v takes the colour of its parent. After shifting, Algorithm 5 can be applied to reduce the colours by one.

Now we have all the pieces for 3-colouring pseudotrees. First, reduce the number of colours from k to 6 using Algorithm 3. Then alternate Algorithm 4 and Algorithm 5 for three iterations. Each iteration reduces the number of colours by one. Thus, we have the following result.

Theorem 5.3. *Every k-coloured directed pseudotree can be 3-coloured in $\log^* k + 6$ communication rounds using a deterministic distributed algorithm.*

For directed cycles the shifting step is not necessary as this produces only an isomorphic coloured cycle where all the colours have been shifted by one step. By omitting the shifting step, we can reduce the 6-colouring into a 3-colouring in three rounds: node v gathers the colours of its predecessor u and successor w and runs Algorithm 5 with $p(v) = \varphi(u)$ and $\varphi'(v) = \varphi(w)$. This results in a speed-up of factor two during the final steps of the cycle colouring algorithm. It is possible to get a similar speed-up in the iterative Cole–Vishkin phase when restricting to directed cycles.

Theorem 5.4. *A k-coloured directed cycle can be 3-coloured in $\left\lceil \frac{1}{2} \log^* k \right\rceil + 3$ rounds.*

Proof. Let $\mathcal{C} = (V, E)$ be a directed k-coloured cycle and $f(\varphi(u), \varphi(v))$ be the value computed by Algorithm 3 for an edge (u, v). First, each node $v \in V$ sends its colour $\varphi(v)$ to both its predecessor u and successor w. Next, node v computes its new

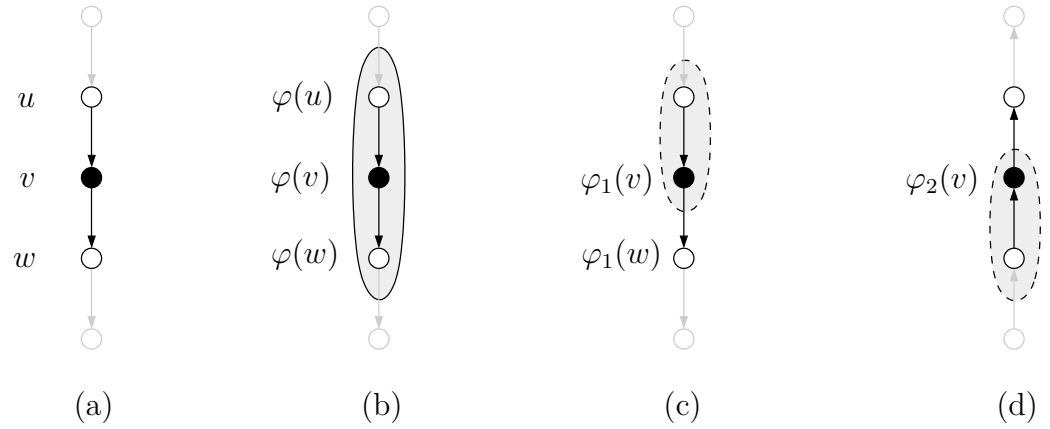


Figure 11: Simulating two rounds of the colour reduction step in cycles. (a) A segment of a cycle $\mathcal{C}$. The node v running the algorithm is highlighted. (b) Node v collects the colours in its radius-1 neighbourhood. (c) Node v computes a new colour $\varphi_1(v)$ by simulating Algorithm 3 with u as predecessor. The node v also computes the new colour $\varphi_1(w)$ of node w. (d) The orientation is reversed and node v simulates the Algorithm 3 with w as predecessor and outputs the colour $\varphi_2(v)$.

colour $\varphi_1(v) = f(\varphi(u), \varphi(v))$ and the colour $\varphi_1(w) = f(\varphi(v), \varphi(w))$ of its successor w. By Lemma 5.1 φ_1 is a proper $\mathcal{O}(\log k)$-colouring.

Now, the node v knows its own colour and the new colour of its successor w. To further reduce the colour without resorting to any additional communication, node v can simulate the Cole–Vishkin colour reduction as if w was the predecessor of v as illustrated in Figure 11. Essentially, the orientation of the edges is reversed. This is possible since the graph is a cycle: each node has exactly the same number of incoming and outgoing edges. Thus, node v can again simulate the Cole–Vishkin colour reduction step and compute a new colour

$$\varphi_2(v) = f(\varphi_1(w), \varphi_1(v)).$$

Therefore, in a single communication round, it is possible to simulate the Cole–Vishkin colour reduction step for two iterations in directed cycles. By Lemma 5.2 it follows that 6-colouring a directed cycles requires at most $\left\lceil \frac{1}{2} \log^* k \right\rceil$ communication rounds. The 6-colouring can be reduced into a 3-colouring greedily in 3 rounds. □

Both results for directed pseudotrees and directed cycles are optimal up to additive terms. That is, 3-colouring directed pseudotrees requires at least $\log^* k + c$ communication rounds while 3-colouring directed cycles requires $\frac{1}{2} \log^* k + c'$ communication

rounds for some (possibly negative) constants c and c'. These lower bound results are discussed in Section 6.

5.3 Colouring bounded-degree graphs

So far we have only considered a very restricted set of graph families as the Cole–Vishkin style algorithms described work only in directed pseudoforests. However, many other distributed graph problems can be solved by first partitioning a more complex graph into pseudoforests [GPS88, PR01, ÅS10]. As an example, we consider the problem of finding a $(\Delta + 1)$-colouring in bounded-degree graphs.

The following distributed $(\Delta + 1)$-colouring algorithm for bounded-degree graphs follows the presentation given by Panconesi and Rizzi [PR01] although Goldberg et al. also utilized a similar strategy [GPS88]. We begin by showing how to distributively decompose the graph into disjoint trees.

Definition 5.5. Let $\mathcal{G} = (V, E)$ be a undirected graph. A *forest decomposition* of the graph $\mathcal{G}$ is a collection $\mathcal{F} = \{F_1, F_2, \ldots, F_\Delta\}$ of pairwise disjoint edge sets such that for all $i \in \{1, 2, \ldots, \Delta\}$, the subgraph $\mathcal{G}_i = (V, F_i)$ is a forest and

$$\bigcup_{i=1}^{\Delta} F_i = E.$$

An *oriented forest decomposition* is a forest decomposition $\mathcal{F}$ where all the edges have been oriented such that for each $i \in \{1, 2, \ldots, \Delta\}$, the graph $\mathcal{G}_i = (V, F_i)$ is a forest of directed rooted trees.

To $(\Delta + 1)$-colour a k-coloured graph $\mathcal{G}$, the general idea is to first construct an oriented forest decomposition $\mathcal{F} = \{F_1, F_2, \ldots, F_\Delta\}$ of the graph $\mathcal{G}$. After this, each forest $\mathcal{G}_i = (V, F_i)$ is 3-coloured in parallel. Finally, all the forest colourings are iteratively combined into a single colouring of the graph $\mathcal{G}$ such that the total number of used colours is at most $(\Delta + 1)$.

The procedure in Algorithm 6 constructs an oriented forest decomposition in the port-numbering model given a k-coloured graph $\mathcal{G}$: every edge is oriented according to the vertex colouring and for all $i \in \{1, 2, \ldots, \Delta\}$ all incoming edges from port i are added to the set F_i. Figure 12 illustrates the construction of the forest decomposition.

Proposition 5.6. *Given a k-coloured graph $\mathcal{G} = (V, E)$, Algorithm 6 computes an oriented forest decomposition $\mathcal{F} = \{F_1, F_2, \ldots, F_\Delta\}$ of graph $\mathcal{G}$ in one communication round.*

Algorithm 6 Oriented forest decomposition

1: Node $v \in V$ sends its colour to each of its neighbours along with the port number associated to the edge.

2: Orient each edge $\{u, v\} \in E$ from the node with a *smaller* colour to the node with the larger colour.

3: Node $v \in V$ adds each *incoming* edge $e = (u, v)$ with the port number i into the set F_i. Similarly, outgoing edges are ordered according to port number assigned by the neighbouring node u.

Proof. First, since the graph is properly k-coloured and the set of colours have a natural total ordering, the orientation is well-defined for each edge. To see that all the edge sets in $\mathcal{F}$ are pairwise disjoint, suppose the opposite holds: some oriented edge (u, v) has been assigned to two sets F_i and F_j where $i \neq j$. Now, the port number p of edge (u, v) at node v would have to satisfy $p = i$ and $p = j$ which is a contradiction. Moreover, each node has at most one predecessor in the subgraph $\mathcal{G}_i$.

Observe that no subgraph $\mathcal{G}_i$ contains a cycle as each edge in F_i is oriented towards a node with a higher colour value. Finally, the number of communication rounds required by the algorithm is clearly one, as all communication is performed during the first step. □

The $(\Delta + 1)$-colouring procedure is given in Algorithm 7. Once an oriented forest decomposition has been constructed, each node simulates the 3-colouring algorithm for rooted trees in each $\mathcal{G}_i = (V, F_i)$ in parallel. This takes at most $\log^* k + 6$ communication rounds. The $(\Delta + 1)$-colouring for graph $\mathcal{G}$ will be constructed iteratively by merging all the 3-colourings of each forest in $\mathcal{F}$.

Algorithm 7 Computing a $(\Delta + 1)$-colouring

1: Construct an oriented forest decomposition $\mathcal{F} = \{F_1, \ldots, F_\Delta\}$.

2: In parallel, compute a 3-colouring φ_i in $\mathcal{G}_i = (V, F_i)$ for each $i \in \{1, \ldots, \Delta\}$.

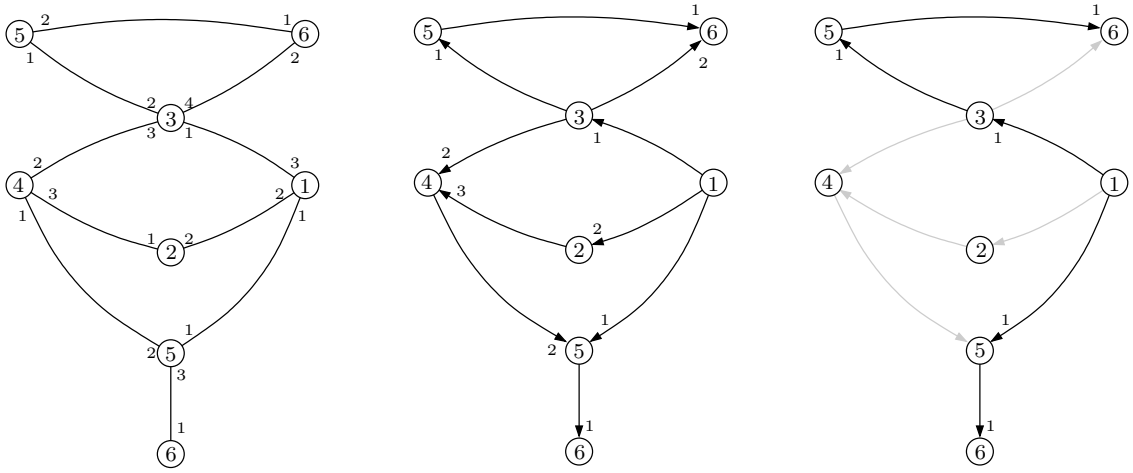
3: Let $A_1 = F_1$ and $\pi_1 = \varphi_1$.

4: For each $i \in \{2, \ldots, \Delta\}$:

(a) Merge the colourings π_{i-1} and φ_i into a $3(\Delta + 1)$-colouring π_i^*.

(b) Greedily reduce the $3(\Delta + 1)$-colouring into a $(\Delta + 1)$-colouring π_i.

5: Output π_Δ as the proper $(\Delta + 1)$-colouring of $\mathcal{G}$.



(a) (b) (c)

Figure 12: Computing an oriented forest decomposition in a coloured graph $\mathcal{G} = (V, E)$. (a) The graph $\mathcal{G}$ with a port-numbering. (b) The graph $\mathcal{G}$ after orienting and assigning the edges into disjoint sets. The port-numbers next to the arrow heads determine the set into which each edge is added. (c) The subgraph $\mathcal{G}_1$. Edges in F_1 are drawn black and edges in $E \setminus F_1$ are gray.

Let $\varphi_1, \varphi_2, \ldots, \varphi_\Delta$ be the 3-colourings for each subgraph $\mathcal{G}_i = (V, F_i)$ where $i \in \{1, 2, \ldots, \Delta\}$. Define $A_1 = F_1$, $A_{i+1} = A_i \cup F_{i+1}$ and $\pi_1 = \varphi_1$. Algorithm 7 satisfies the following invariant for each $i \in \{2, 3, \ldots, \Delta\}$: the colouring π_i of the subgraph $\mathcal{H}_i = (V, A_i)$ will use at most $\Delta + 1$ colours. Consider the node v when $i > 1$. The node v will set the pair $(\pi_{i-1}(v), \varphi_i(v))$ as its new colour π_i^*. The pair can be presented as an integer

$$\pi_i^*(v) = 3 \cdot \pi_{i-1}(v) + \varphi_i(v).$$

Using induction we can see that the colouring π_i^* is a proper vertex colouring for $\mathcal{H}_i$ when $1 \leq i \leq \Delta$. The claim holds for the base case $\pi_1^* = \varphi_1$ since φ_1 is a proper colouring of $\mathcal{H}_1 = (V, F_1)$. Assume that the colouring π_i^* properly colours $\mathcal{H}_i$ for some i such that $1 \leq i \leq \Delta - 1$. For the induction step, we show that π_{i+1}^* properly colours $\mathcal{H}_{i+1}$. Let $\{u, v\} \in A_{i+1} = A_i \cup F_{i+1}$. If $\{u, v\} \in A_i$, then by the induction hypothesis it holds that $\pi_i^*(u) \neq \pi_i^*(v)$. Otherwise, if $\{u, v\} \in F_{i+1}$, then $\varphi_{i+1}(u) \neq \varphi_{i+1}(v)$. Therefore, at least one component in $(\pi_i(v), \varphi_{i+1}(v))$ will differ and π_{i+1}^* is a proper colouring of graph $\mathcal{H}_{i+1} = (V, A_{i+1})$.

We also maintain the invariant that the colouring π_{i-1} uses at most $\Delta + 1$ colours. Since φ_i is a 3-colouring, the new colouring π_i^* satisfies $\pi_i^*(v) < 3(\Delta + 1)$ for all $v \in V$. To satisfy the invariant for the next step, each node v iteratively runs the

greedy colour reduction algorithm given in Section 5.1 for $2(\Delta + 1)$ rounds until a $(\Delta + 1)$-colouring π_i has been computed in $\mathcal{H}_i = (V, A_i)$.

Theorem 5.7. *Algorithm 7 finds a $(\Delta + 1)$-colouring in $\mathcal{O}(\Delta^2 + \log^* k)$ rounds in any k-coloured bounded-degree graph with maximum degree Δ.*

Proof. We already showed that the algorithm produces a proper $(\Delta + 1)$-colouring. To establish the running time, observe that the first two steps of the algorithm require $\mathcal{O}(\log^* k)$ communication rounds. Step 4 is iterated for $\Delta - 1$ times where each iteration requires at most $2\Delta + 2$ communication rounds. In total, step 4 requires at most $2(\Delta^2 - 1)$ communication rounds. □

One might ask why settle for a $(\Delta + 1)$-colouring as a $(\Delta + 1)$-colouring can be arbitrarily bad compared to an optimal colouring. For example, the maximum degree of a *star graph* (a tree of height 2) may be arbitrarily large. But all trees are bipartite and hence 2-colourable.

For one, finding an optimal colouring or even approximating it is a notoriously difficult task even in the centralized setting as discussed in Section 3.5. Furthermore, a classic result due to Erdős states that for every integer k there exists a graph such that the optimal colouring uses k colours and the shortest cycle in the graph has more than k nodes [Erd59]. That is, locally the graph resembles a tree. In a sense this implies that the number of colours used by an optimal colouring can be a strictly *global* phenomenon. The existence proof of these graphs was first given by Erdős using the probabilistic method. However, a constructive proof was later given by Lovász [Lov68].

5.4 The current state of deterministic distributed colouring

We conclude this section with a brief overview on other work regarding deterministic distributed colouring algorithms. Essentially all algorithms either produce a $(\Delta + 1)$-colouring or an $\mathcal{O}(\Delta)$-colouring. For this section, we assume the model with unique identifiers and all the running times are given as a function of the maximum degree Δ and the number of nodes n.

Linial showed that it is iteratively possible to find a $\mathcal{O}(\Delta^2)$-colouring in $\mathcal{O}(\log^* n)$ communication rounds for bounded-degree graphs [Lin87, Lin92]. Unfortunately, no explicit algorithm was given as the proof given essentially relies on a non-constructive existence proof of certain set systems. However, it is pointed out that a slightly weaker

result (with regards to constant factors) can be constructively proven. This algorithm is often used as a subroutine in other distributed graph colouring algorithms.

At the same time, Goldberg, Plotkin and Shannon [GP87, GPS88] studied parallel $(\Delta + 1)$-colouring algorithms. In addition to adapting the original Cole–Vishkin 3-colouring algorithm for directed rooted trees and pseudoforests, Goldberg and Plotkin also gave a $(\Delta + 1)$-colouring algorithm for bounded-degree graphs in the PRAM model [GP87]. The algorithm uses the basic Cole–Vishkin bit manipulation technique to first colour the graph with $2^{\mathcal{O}(\Delta \log \Delta)}$ colours. A $(\Delta + 1)$-colouring is then attained by greedily computing maximal independents sets in sequence and colouring each such set with a different colour.

Goldberg, Plotkin and Shannon [GPS88] soon gave a significantly faster $(\Delta + 1)$-colouring algorithm that utilises the 3-colouring algorithm for pseudoforests. The algorithm first partitions the graph into disjoint pseudoforests each of which are 3-coloured in parallel. These colourings are then used to recolour the graph with $(\Delta + 1)$-colours. In the distributed model, this algorithm requires $\mathcal{O}(\Delta^2 + \log^* n)$ communication rounds.

Later, Panconesi and Rizzi [PR01] used similar ideas to derive distributed algorithms for maximal matchings, edge colourings, maximal independent sets, and vertex colourings in bounded-degree graphs. These algorithms compute maximal matchings and $(2\Delta - 1)$-edge colourings in $\mathcal{O}(\Delta + \log^* n)$ communication rounds, while $(\Delta + 1)$-vertex colourings and maximal independent sets are computed in $\mathcal{O}(\Delta^2 + \log^* n)$ communication rounds.

While the $(\Delta + 1)$-colouring algorithms of Goldberg et al. [GPS88] and Panconesi and Rizzi [PR01] are rather simple and relatively fast in graphs of low maximum degree, the algorithms are considerably slower in graphs with high maximum degrees. For a faster algorithm for small graphs with high maximum degrees, a standard algorithm can be used to find a $(\Delta + 1)$-colouring in $\mathcal{O}(\Delta \log n)$ communication rounds [Pel00, Ch. 7.4].

In 2006, Kuhn and Wattenhofer [KW06] improved the upper bounds by showing that an iterative colour reduction algorithm can compute a $(\Delta + 1)$-colouring in $\mathcal{O}(\Delta \log \Delta + \log^* n)$ communication rounds. Intriguingly, Szegedy and Vishwanathan [SV93] had earlier conjectured that iterative colour reduction algorithms that find a $(\Delta + 1)$-colouring require $\Omega(\Delta \log \Delta)$ communication rounds. Thus, it was not surprising that the consequent improvements on the Δ-term required rather different techniques.

During 2009, a new line of positive results emerged by Kuhn [Kuh09] and Barenboim and Elkin [BE09]. The two groups independently published $(\Delta + 1)$-colouring algorithms with a running time of $\mathcal{O}(\Delta + \log^* n)$. Unlike the previous iterative colour reduction methods, the new algorithms relied on a different approach. Both algorithms first find certain *improper* vertex colourings.

Instead of maintaining a proper colouring as an invariant like the previous algorithms, the new algorithms compute a variant of weak colouring known as d-defective colouring: each colour class is guaranteed to induce a subgraph with bounded maximum degree of d. Finally, the defective colourings are recursively mended into a single proper $(\Delta + 1)$-colouring.

Recently, Barenboim and Elkin [BE10a] generalized the notion of defective colourings and developed a family of algorithms that can find small colourings in polylogarithmic time, that is, time polynomial in the logarithm of n and Δ. In particular, $\mathcal{O}(\Delta^{1+\eta})$-colourings can be computed in time $\mathcal{O}(\log \Delta \log n)$ for any positive constant $\eta > 0$ and a $\mathcal{O}(\Delta)$-colouring can be computed in $\mathcal{O}(\Delta^{\epsilon} \log n)$ rounds for any positive constant $\varepsilon > 0$.

It seems that defective and weak colourings are sufficient to break symmetry in many cases instead of proper vertex colourings. As the current algorithms for fast vertex colouring use improper colourings, a promising area of investigation is to study if algorithms for improper colourings can be applied to attain better algorithms for other distributed graph problems as well. As a very recent example, defective colourings were applied in new upper bounds for the distributed edge colouring problem [BE10b].

6 Lower bounds for distributed colouring

The first lower bound results regarding the complexity of distributed vertex colouring date back to the 1980s. Linial [Lin87, Lin92] proved that in a directed n-cycle with unique identifiers, any deterministic colouring algorithm requires at least $\frac{1}{2}(\log^* n - 3)$ communication rounds. Later, Naor [Nao91] showed that randomisation does not help: any probabilistic 3-colouring algorithm running fewer than $\frac{1}{2}(\log^* n - 4)$ rounds has a high probability of producing an improper colouring. Of course, this directly implies a lower bound for $(\Delta + 1)$-colouring as cycles are bounded-degree graphs with $\Delta = 2$.

Fraigniaud et al. have studied the information sensitivity of graph colouring in directed cycles and rooted trees [FGIP07], that is, how many bits of additional advice must be given to the nodes such that the time required to colour a graph with a constant number of colours reduces. It turns out that colouring cycles and trees is information insensitive in the sense that colouring a cycle with a constant number of colours in constant time essentially requires advice consisting of the whole solution. Similarly, colouring rooted directed trees and d-regular trees is information insensitive.

In this section, we prove that there are no constant-time algorithms for colouring directed cycles with a constant number of colours. Furthermore, we show an asymptotic lower bound of $\Omega(\log^* n)$ for colouring cycles with a constant number of colours. Finally, we show that colouring directed rooted trees requires twice as many communication rounds as colouring directed cycles.

6.1 Ramsey-theoretic lower bound arguments

Ramsey theory has shown to be a useful tool in the study of distributed algorithms. In particular, many results regarding local algorithms [Lin92, NS95, CHW08, ÅFP+09] have been or can be proved using *Ramsey's theorem*. Ramsey's theorem dates back to the early 1930s and was originally used in the context of mathematical logic [Ram30]. Essentially, Ramsey's theorem states that certain substructures are unavoidable in large enough structures. For a modern proof of the theorem, see for example the textbook by Graham et al. [GRS80].

Before showing the lower bound results, we will introduce Ramsey's theorem and the notation used in this section. Recall that an r-colouring of a set S is a mapping $c\colon S \to [r]$. For $m, \ell \in \mathbb{N}$ such that $2 \leq \ell \leq m$ and a colouring $c\colon \binom{S}{\ell} \to [r]$, we say that an m-subset $A \in \binom{S}{m}$ is *monochromatic* (under c) if for all $B, B' \in \binom{A}{\ell}$ it holds that $c(B) = c(B')$, that is, c colours all the ℓ-subsets of the m-subset $A \subseteq S$ with the same colour.

Theorem 6.1 (Ramsey's theorem)**.** *For all positive integers $2 \leq \ell \leq m$ and $2 \leq r$, there exists a positive integer N such that for any colouring $c\colon \binom{[N]}{\ell} \to [r]$ there exists a monochromatic m-subset $A \subset [N]$.*

For any given $2 \leq \ell \leq m$, the value $R(\ell, m)$ denotes the smallest N such that any 2-colouring of ℓ-subsets of $[N]$ produces a monochromatic m-subset of $[N]$.

We prove the lower bounds for the following model of distributed computation: each node is given a unique identifiers from the set $\{0, 1, \ldots, n-1\}$ and the edges have a clockwise orientation, that is, the input graph is a directed cycle.

We begin with a proof that a *maximal* independent set cannot be computed in a constant number of communication rounds in a directed cycle with unique identifiers. This implies that there does not exist a deterministic constant-time algorithm that finds a proper vertex colouring using a constant-number of colours.

Lemma 6.2. *There is no constant-time deterministic distributed algorithm for computing a maximal independent set in a directed n-cycle with unique identifiers.*

Proof. Assume that $\mathcal{A}$ is an algorithm that computes a maximal independent set in $T \in \mathbb{N}$ communication rounds. We will show that there exists an input such that the output of $\mathcal{A}$ is not a maximal independent set.

Consider radius-T neighbourhoods with increasing identifiers $i_0 < i_1 < \cdots < i_{2T}$. Due to the order of the identifiers, such a cycle neighbourhood can be regarded as a set $\{i_0, i_1, \ldots, i_{2T}\}$. Furthermore, in such neighbourhoods, the output of node i_T when running the algorithm $\mathcal{A}$ corresponds to a function $f_{\mathcal{A}} \colon \binom{[n]}{2T+1} \to \{0, 1\}$. Node i_T joins the independent set if $f_{\mathcal{A}}(\{i_0, i_1, \ldots, i_{2T}\}) = 1$ and otherwise not. The function $f_{\mathcal{A}}$ is in fact a 2-colouring of all $(2T+1)$-subsets of the identifier set $[n]$.

We can now apply Theorem 6.1 by setting $\ell = 2T+1$, $m = \ell+2$ and $c = 2$. As stated in the theorem, for a large enough n (the size of the identifier space), *any* 2-colouring of the ℓ-sets must produce a monochromatic m-subset $A = \{i_0, i_1, \ldots, i_{2T+2}\} \subset [n]$.

Let $n \geq R(m, \ell)$ and fix the monochromatic ℓ-subset A. To yield a contradiction, we will construct an n-cycle with a segment consisting of the identifiers in A in increasing order as illustrated in Figure 13. The remaining identifiers $[n] \setminus A$ can be assigned arbitrarily for the remaining nodes.

In the segment constructed from A, the algorithm $\mathcal{A}$ must have the same output in the three nodes with identifiers i_T, i_{T+1}, and i_{T+2}. If the output is 1, the algorithm does not output an independent set. On the other hand, if the output is 0, the independent set is not maximal: the node with identifier i_{T+1} could then be added to the independent set. Therefore, the algorithm $\mathcal{A}$ does not produce a maximal independent set. □

This technique can be extended to argue that it is not possible find a constant factor approximation of a maximal independent set in a directed cycle in constant

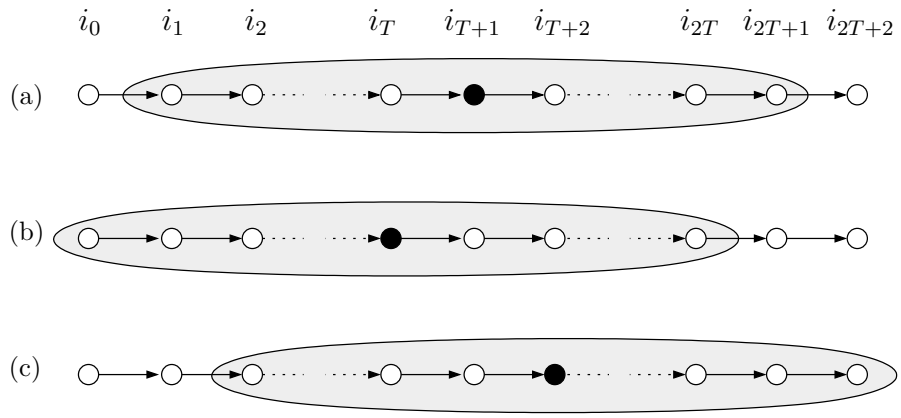


Figure 13: The Ramsey-theoretic lower bound construction using the monochromatic set $A = \{i_0, i_1, \ldots, i_{2T+2}\}$ as unique identifiers for a segment of length $2T+3$. (a) The local T-neighbourhood of node i_{T+1}. (b) The local T-neighbourhood of node i_T. (c) The local T-neighbourhood of node i_{T+2}.

time [CHW08]. As a corollary of Lemma 6.2, we also obtain a lower bound for colouring cycles with a constant number of colours.

Corollary 6.3. *There is no constant-time deterministic distributed algorithm for $\mathcal{O}(1)$-colouring a directed n-cycle with unique identifiers.*

Proof. Let $k \in \mathbb{N}$. Assume that a k-colouring algorithm $\mathcal{A}$ exists with a constant running time $T \in \mathbb{N}$. Now the algorithm $\mathcal{A}$ can be used to compute a maximal independent set in $T + k \in \mathcal{O}(1)$ communication rounds: first colour the cycle with k colours, and then in sequence, greedily try to add nodes in each colour class to the independent set. This is a contradiction with Lemma 6.2. □

We will now give a Ramsey-theoretic proof of Linial's $\Omega(\log^* n)$ lower bound. For this task, we apply an upper bound for the Ramsey numbers $R(\ell, m)$. Using this upper bound, we can analyse how large must the running time T of an algorithm be in regard to the number of nodes. We express the lower bound using the following exponent tower function.

Definition 6.4. The *tower function* $\operatorname{twr}(i, x)$ for $i \in \mathbb{N}^+$ is defined inductively by

$$\operatorname{twr}(1, x) = x,$$
$$\operatorname{twr}(i+1, x) = 2^{\operatorname{twr}(i,x)}.$$

Observe the relation between tower function and the iterated logarithm. In particular, it holds that $\log^{(i-1)} \operatorname{twr}(i, x) = x$.

Lemma 6.5. *For $2 \leq \ell \leq m$ and $c_\ell = 2(\ell-1)!$ the Ramsey numbers satisfy*

$$R(\ell, m) \leq \operatorname{twr}(\ell, c_\ell m).$$

Proof (Sketch). The Ramsey numbers for the special case $\ell = 2$ are known as graph Ramsey numbers [GRS80, Ch. 1]. This special case is upper bounded by

$$R(2, m) \leq \binom{2(m-1)}{m-1} \leq 2^{2m-1} - 1.$$

Furthermore, a proof of Ramsey's theorem [GRS80, Ch. 4.7] for fixed ℓ yields

$$\log_2 R(\ell, m) \leq R(\ell-1, m)^{\ell-1}.$$

With the above inequality, we show by induction that the tower bound

$$R(\ell, m) \leq \operatorname{twr}(\ell, c_\ell m)$$

holds where $2 \leq \ell \leq m$ and $c_\ell = 2(\ell-1)!$. The base case $\ell = 2$ is covered by the upper bound for graph Ramsey numbers:

$$R(2, m) \leq 2^{2m-1} - 1 < 2^{2m} = \operatorname{twr}(2, c_2 m).$$

Using the induction hypothesis we get

$$\begin{aligned} R(\ell+1, m) &\leq 2^{R(\ell,m)^\ell} \\ &\leq 2^{\left(\operatorname{twr}(\ell, c_\ell m)^\ell\right)} \\ &\leq 2^{\operatorname{twr}(\ell, \ell c_\ell m)} \\ &= \operatorname{twr}(\ell+1, c_{\ell+1} m). \end{aligned}$$

□

While the following Ramsey-theoretic argument has been known about as long as Linial's lower bound [Lin87, GP87] (even Linial referred to the result in his original paper), the proof itself does not seem to appear explicitly in the literature.

Theorem 6.6. *There is no deterministic algorithm for computing a maximal independent set in a directed n-cycle with unique identifiers in time $o(\log^* n)$.*

Proof. Due to Theorem 5.4 we know that for large enough n, a maximal independent set can be greedily computed in time at most $\log^* n$ by first 3-colouring the cycle and then greedily computing a maximal set. This takes time $\frac{1}{2}\log^* n + 5$.

Let $\mathcal{A}$ be an algorithm that computes a maximal set in any directed n-cycle in $T(n)$ communication rounds such that $T(n) \leq \log^* n$. Set $\ell = 2T(n) + 1$ and $m = \ell + 2$. Due to Lemma 6.2 we know that if $n \geq R(\ell, m)$ holds, then the algorithm must fail. Therefore, we will consider the case $n < R(\ell, m)$ and apply the upper bound for $R(\ell, m)$ given in Lemma 6.5.

Now $n < R(\ell, m) < \mathrm{twr}(\ell, c_\ell m)$. Using the definition of the iterative logarithm and the tower function, it follows that

$$\begin{aligned} \log^{(\ell-1)} n &< c_\ell m \\ &= 2(\ell-1)!m \\ &< (\ell+2)! \end{aligned}$$

holds. Therefore, for large enough n,

$$\begin{aligned} \log^* n &< \ell - 1 + \log^* \left((\ell+2)!\right) \\ &\leq \ell - 1 + \log^* \Big(\log \Big((\ell+2)!\Big) \Big) + 1 \\ &= \ell + \log^* \left(\sum_{i=1}^{\ell+2} \log i \right) \\ &\leq \ell + \log^* \left(\ell \log(\ell+2) + 1\right) \\ &\leq \ell + 1 + \log^* \left(\log(\ell) + \log(\log(\ell))\right). \end{aligned}$$

Since we set $\ell = 2T(n) + 1$ we have

$$\log^* n < 2T(n) + 2 + \log^* \Big(\log(\ell) + \log^{(2)}(\ell)\Big).$$

For large enough n it holds that

$$T(n) > \frac{1}{2}\Big(\log^* n - 2 - \log^* \Big(\log T(n) + \log^{(2)} T(n) + 2\Big)\Big)$$

and moreover

$$T(n) \geq \log T(n) + \log^{(2)} T(n) + 2.$$

Therefore, since we know that $T(n) \leq \log^* n$ we can derive the following inequalities when n is large enough

$$\begin{aligned} T(n) &> \frac{1}{2}\Big(\log^* n - 2 - \log^* \Big(\log T(n) + \log^{(2)} T(n) + 2\Big)\Big) \\ &\geq \frac{1}{2}\Big(\log^* n - 2 - \log^*(T(n))\Big) \\ &\geq \frac{1}{2} \log^* n - 1 - \frac{1}{2} \log^*(\log^* n) \\ &\geq \frac{1}{2} \log^* n - \log^*(\log^* n). \end{aligned}$$

□

The above analysis yields *almost* the same result as Linial's bound of $\frac{1}{2}(\log^* n - 3)$. However while the log-star additive term in the above analysis is small, it seems to be difficult to achieve Linial's exact bound using upper bounds for Ramsey numbers. All the known bounds for general Ramsey numbers are tower bounds containing a function of either ℓ or m at the topmost level. Because both parameters depend on $T(n)$, any similar analysis will yield an additive term depending on $T(n)$.

6.2 Lower bound for tree colouring

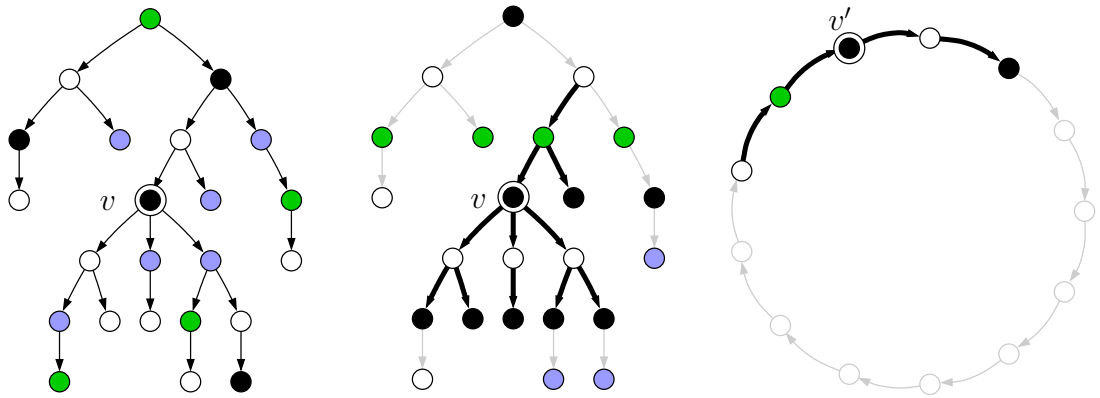
In this section, we show that any deterministic distributed algorithm for colouring rooted trees requires twice as much time as an algorithm for colouring directed cycles. Together with the lower bound for colouring directed cycles we get a lower bound for colouring directed rooted trees.

Lemma 6.7. *Let $c, k, T \in \mathbb{N}^+$ such that $c \leq k$. If there exists a deterministic distributed algorithm for c-colouring a k-coloured directed cycle in T communication rounds, then any k-coloured directed rooted tree can be c-coloured in $2T$ communication rounds.*

Proof. Let $\mathcal{A}$ be a c-colouring algorithm for k-coloured directed cycles with running time T. The output of the algorithm $\mathcal{A}$ at any node v depends only on the radius-T neighbourhood of node v. For directed cycles the T-neighbourhood is a directed path of length $2T + 1$.

Let $\mathcal{G} = (V, E)$ be a k-coloured directed rooted tree. We will now devise an algorithm for c-colouring $\mathcal{G}$ that runs for $2T$ rounds. First, each node v the shifts the colours downwards by running Algorithm 4 for T rounds and simultaneously storing all the previous colours in memory. Now, for any node $v \in V$ every branch connected to v is equally-coloured up to depth T; that is, every directed T-path from v towards the leaves is isomorphic up to port-numbering. If some path is shorter than T, say of length $t < T$, then the corresponding leaf node u will use the previously received colours $c_0, c_1, \ldots, c_{T-t-1}$ and simulate a path of length $T - t$ coloured with the corresponding colours.

After the first step, each node v knows the colours of its children up to depth T. It remains for node v to collect the colours of its T predecessors. If v has less than T predecessors, for example only $t < T$ predecessors, the root node r simulates a properly k-coloured t-path towards r.



(a) (b) (c)

Figure 14: Colouring directed trees with a T-time cycle colouring algorithm. (a) A coloured rooted tree and a node v running the algorithm. (b) The tree with colours shifted downwards T times. (c) The virtual cycle constructed by v. The node v simulates the cycle colouring algorithm on the virtual node v' and outputs the new colour of v'.

After $2T$ rounds each node v knows the colours $d_0, d_1, \ldots, d_{T-1}$ of its T predecessors, its own colour d_T, and the colours $d_{T+1}, d_{T+2}, \ldots, d_{2T}$ of its successors up to depth T. Node v can then construct a segment of a directed cycle $\mathbf{x} = (x_0, x_1, \ldots, x_{2T})$ with colours $d_0, d_1, \ldots, d_{2T}$ and port-numbers assigned according to the orientation. Finally, node v simulates $\mathcal{A}$ with input $\mathbf{x}$ and outputs the colour of x_T. Since $\mathcal{A}$ outputs a c-colouring colour chosen by v is also at most c. Figure 14 illustrates the simulation.

To see that node v outputs a different colour than any of its neighbours, suppose the opposite: either the the end-points of (u, v) or (v, w) have been coloured with the same colours. Notice that v simulates $\mathcal{A}$ on a path with colours $d_0, d_1, \ldots, d_{2T}$ whereas node u simulates $\mathcal{A}$ on a path with colours $a, d_0, d_1, \ldots, d_{2T-1}$ where $a \neq d_0$. If after simulating $\mathcal{A}$ both u and v have the same colour, then the algorithm $\mathcal{A}$ must fail in a properly k-coloured directed cycle with a path consisting of colours $a, d_0, d_1, \ldots, d_{2T}$ which contradicts the correctness of $\mathcal{A}$. The case (v, w) is analogous. □

For proving the other direction we will use a family of k-coloured rooted trees. Figure 15 illustrates the following construction for $k = 4$.

Definition 6.8. For all positive integers $k \geq 2$, the tree $\mathcal{T}_k$ is the infinite directed rooted k-coloured tree with the following properties: each node has $k - 1$ successors and all nodes but the root have a predecessor $p(v)$. Any node v with colour $c \in [k]$

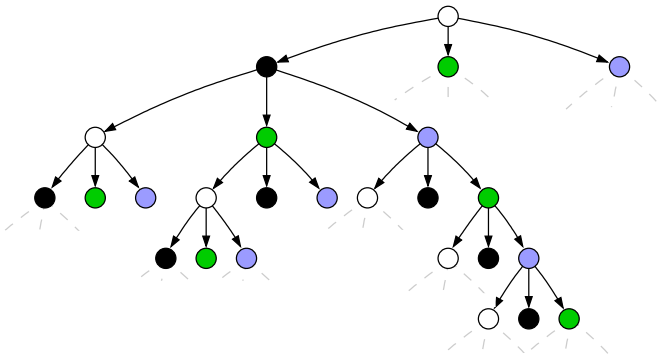


Figure 15: A partially drawn k-regular tree $\mathcal{T}_k$ where $k = 4$. Colour 0 is white, colour 1 black, colour 2 green and colour 3 blue.

has $k - 1$ children each with a distinct colour from the set $[k] \setminus \{c\}$ such that the tree is properly k-coloured. The port numbering for node v is assigned such that the children have the first $k - 1$ ports and the parent $p(v)$ is given the port number $\deg(v)$. The colour of the root is 0.

Proposition 6.9. *Let $k \geq 2$ and T be positive integers and $\mathcal{A}$ be a distributed algorithm with running time T in the tree $\mathcal{T}_k$. For any node v, the algorithm $\mathcal{A}$ only needs to gather input from the T predecessors of v.*

Proof. First, note that while $\mathcal{T}_k$ is infinite, the running time of $\mathcal{A}$ depends only on T which is a fixed constant that may depend on k. Moreover, $\mathcal{T}_k$ contains all the possible directed k-coloured paths.

Observe that any node v with colour $\varphi(v) \in [k]$ knows the colours of its children without resorting to any communication. This is possible due to the construction of $\mathcal{T}_k$. In fact, as the tree is infinite, v can recursively deduce the T-neighbourhood consisting of its successors. Furthermore, after v learns the colour of its parent $p(v)$, then v can similarly deduce the whole subtree of $p(v)$. Therefore, the node v can construct its complete T-neighbourhood with only the information regarding its T successors, that is, the directed path of length $T + 1$ ending in v. □

Lemma 6.10. *Let $c, k, T \in \mathbb{N}^+$ such that $c \leq k$. If there exists a deterministic distributed algorithm for c-colouring a k-coloured directed rooted trees in $2T$ communication rounds, then any k-coloured directed cycle can be c-coloured in T communication rounds.*

Proof. Let $\mathcal{C} = (V, E)$ be a directed k-coloured graph and $\mathcal{A}$ be a $2T$-time algorithm for colouring trees. Colouring the cycle $\mathcal{C}$ proceeds as follows. First, every node $v \in V$ gathers its radius-T neighbourhood in T communication rounds; again the neighbourhood is a directed path $\mathbf{x} = (x_0, x_1, \ldots . x_{2T})$. Node $v = x_T$ can then map the path $\mathbf{x}$ into a equally-coloured path $\phi(\mathbf{x})$ in the tree $\mathcal{T}_k$. Consider the node $u = \phi(x_{2T})$ in $\mathcal{T}_k$. As noted in Proposition 6.9, the output of u depends only on its predecessors and itself which are given in $\phi(\mathbf{x})$. Therefore, v can simulate $\mathcal{A}$ on node u and output the new colour of u. See Figure 16 for illustration.

To see that the method produces a proper c-colouring, assume the opposite: let $(v, w) \in E$ be an edge with both end-points having the same colour. This implies that also $\phi(v) = \phi(w)$ which is a contradiction since $\mathcal{A}$ properly colours $\mathcal{T}_k$. Finally, the colouring uses at most c colours as by assumption $\mathcal{A}$ was a c-colouring algorithm. □

Lemma 6.7 and Lemma 6.10 together imply the following theorem.

Theorem 6.11. *Let $c, k, T \in \mathbb{N}^+$ such that $c \leq k$. There is a T-time algorithm for c-colouring k-coloured directed cycles if and only if k-coloured directed rooted trees are c-colourable in time $2T$.*

The lower bound $\frac{1}{2}(\log^* n - 3)$ for 3-colouring directed cycles and Theorem 6.11 establish that the Cole–Vishkin style algorithms presented in Section 5.2 for 3-colouring directed cycles and directed rooted trees are optimal up to additive constant terms. Section 8 introduces improvements to these additive constant terms.

Therefore, while not strictly local, directed rooted trees can be coloured rather efficiently in $\Theta(\log^* n)$ time. For non-oriented trees the case is quite different. Linial showed that a d-regular tree of height r cannot be coloured with less than $\frac{1}{2}\sqrt{d}$ colours in time $\frac{2}{3}r$ [Lin92]. A d-regular tree of height r has $n = d^{r-1}$ nodes. For fixed $d \geq 2$ the running time is

$$\frac{2}{3}r = \frac{2}{3}\log_d d^r > \frac{2}{3}\log_d d^{r-1} = \frac{2}{3}\log_d n \in \Omega(\log n).$$

Therefore, without an orientation from parents to children, $\frac{1}{2}\sqrt{d}$-colouring d-regular trees cannot be done in $o(\log n)$ rounds. That is, a colouring algorithm has to use time proportional to the diameter of the tree.

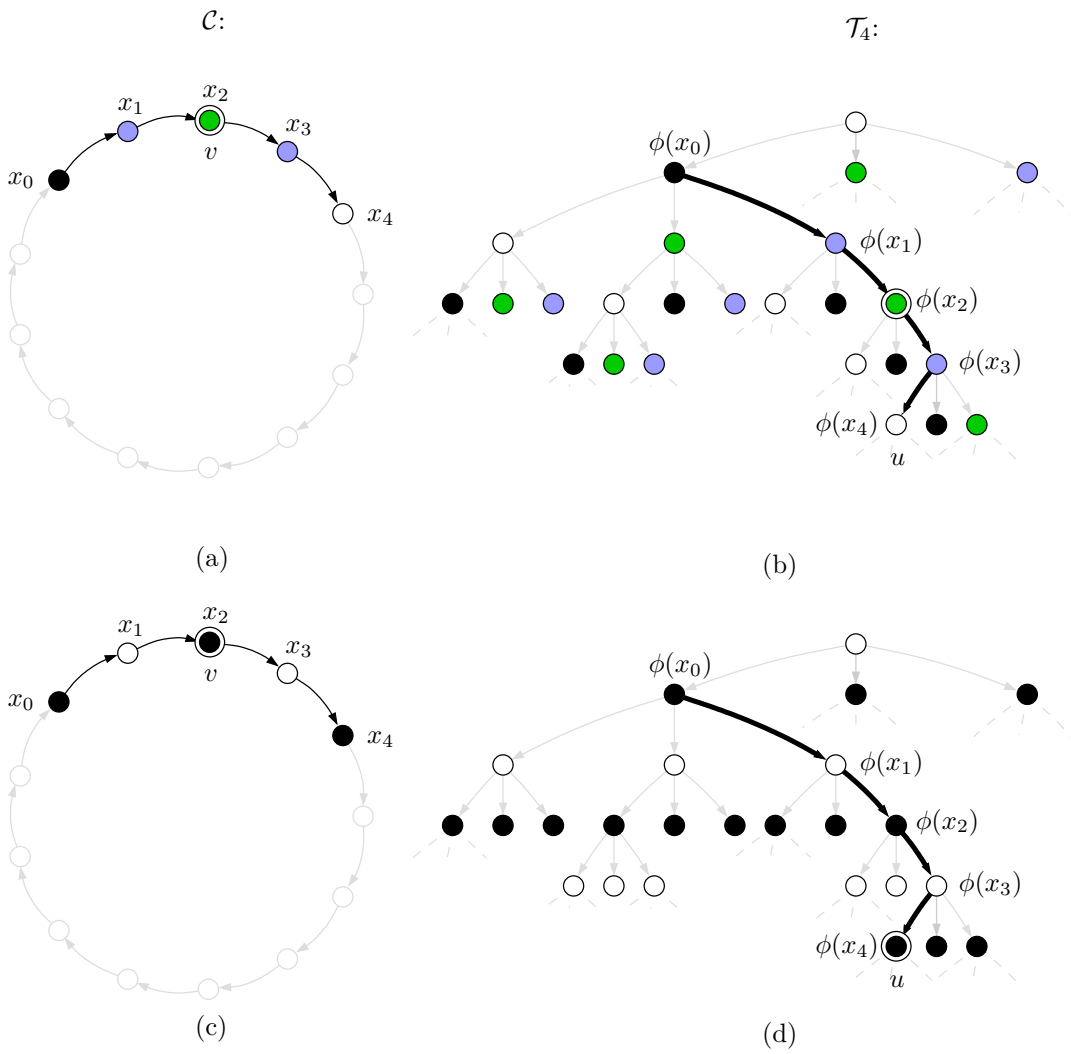


Figure 16: Colouring a 4-coloured directed cycle $\mathcal{C}$ by simulating a $2T$-time tree colouring algorithm $\mathcal{A}$ where $T = 2$. (a) A node v in the input cycle and its radius-T neighbourhood $\mathbf{x} = (x_0, x_1, x_2, x_3, x_4)$. (b) The 4-regular tree $\mathcal{T}_4$ and the mapped radius-T neighbourhood $\phi(\mathbf{x})$. (c) The new colouring mapped from the recoloured $\mathcal{T}_4$. Node v chooses the colour of $\phi(x_4)$ as its new colour in the cycle. (d) The tree $\mathcal{T}_4$ after running the $2T$-time colouring algorithm $\mathcal{A}$ for trees.

6.3 Neighbourhood graphs

We will now introduce the concept of neighbourhood graphs. Neighbourhood graphs contain all possible T-neighbourhoods of all communication graphs in a given graph family. In practice, the communication graph is from a restricted family of k-coloured graphs. We will focus on neighbourhood graphs for k-coloured directed cycles in the port-numbering model.

The colourability of neighbourhood graphs and the existence of distributed colouring algorithms are closely related. Linial observed this and used neighbourhood graphs to prove the $\Omega(\log^* n)$ lower bound result for distributed 3-colouring of directed cycles with unique identifiers [Lin87, Lin92]. Similar constructions have since been used in various other results related to distributed colouring [Nao91, KW06, FGIP07].

The neighbourhood graphs can be used to compute *exact* lower and upper bounds for parametrized distributed colouring. The parameters we study are the number of initial colours (or unique identifiers) denoted by k (or n), and the running time T which is equal to the radius of the local neighbourhood.

First, let us present a construction due to Linial [Lin92]: the neighbourhood graph for directed n-cycles with unique identifiers from the set $\{0, 1, \ldots, n-1\}$. The construction can also be interpreted as the neighbourhood graph for directed cycles with a radius-T colouring. Here n is the number of colours (or unique identifiers) and T is the running time of the distributed algorithm.

Definition 6.12. Let T be a non-negative integer. The *T-neighbourhood graph for directed cycles with unique identifiers* is denoted by $\mathcal{L}_{n,T} = (V, E)$. The set $V \subset [n]^{2T+1}$ of nodes consists of tuples $(x_0, x_1, \ldots, x_{2T})$ where each x_i is a distinct integer from the set $[n]$. All edges in the graph are between nodes $(x_0, x_1, \ldots, x_{2T})$ and $(y, x_0, x_1, \ldots, x_{2T-1})$ where $y \neq x_{2T}$.

The construction for properly k-coloured neighbourhoods is slightly different.

Definition 6.13. Let $k \geq 3$ and T be non-negative integers. The T-neighbourhood graph for k-coloured directed cycles is the graph $\mathcal{N}_{k,T} = (V, E)$ with the following properties:

(i) The set $V \subset [k]^{2T+1}$ of nodes is the collection of all possible T-neighbourhoods of directed k-coloured cycles. That is,

$$V = \{(x_0, x_1, \ldots, x_{2T}) : x_i, x_{i+1} \in [k] \text{ and } x_i \neq x_{i+1} \text{ where } 0 \leq i < 2T\}.$$

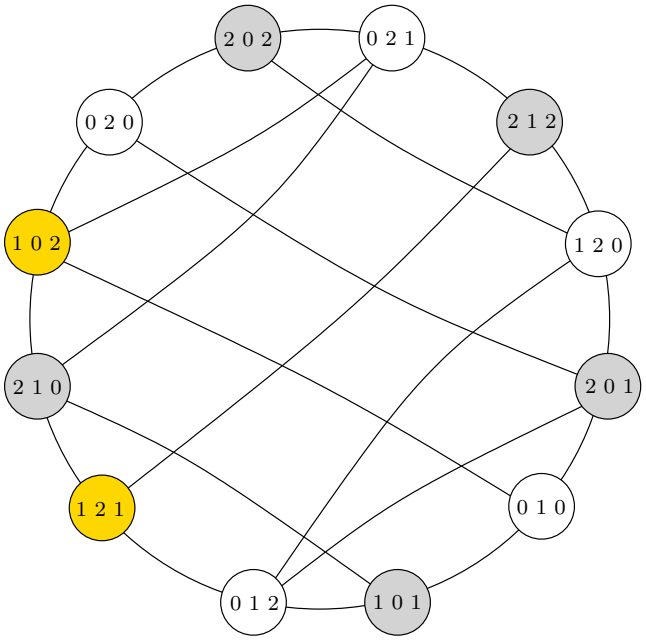


Figure 17: The neighbourhood graph $\mathcal{N}_{3,1}$.

(ii) There is an undirected edge between two nodes $\mathbf{v} = (v_0, v_1, \ldots, v_{2T})$ and $\mathbf{w} = (w_0, w_1, \ldots, w_{2T})$ if $\mathbf{w} = (v_1, v_2, \ldots, v_{2T}, z)$ for some $z \in [k] \setminus \{v_{2T}\}$.

The connection between the existence of distributed colouring algorithms and neighbourhood graphs is established by the following theorem.

Theorem 6.14. *Let c, k, and T be non-negative integers such that $3 \leq c \leq k$. The following claims are equivalent:*

(i) There exists a deterministic distributed algorithm for c-colouring a directed k-coloured cycle in T communication rounds.

(ii) The neighbourhood graph $\mathcal{N}_{k,T}$ is c-colourable.

Proof. Fix the integers c, k, and T.

(i) $\Rightarrow$ (ii): Let $\mathcal{A}$ be a distributed c-colouring algorithm for k-coloured directed cycles with a running time of T rounds. Let $\mathcal{N}_{k,T} = (V, E)$ be the neighbourhood graph for the given parameters k and T. Let $f_{\mathcal{A}}$ be the function corresponding to the distributed algorithm $\mathcal{A}$. The function $f_{\mathcal{A}}$ maps the radius-T local view of a node in a k-coloured cycle to a set $[c]$ of colours. Recall from Definition 4.1 that in directed cycles the local views are directed paths $(x_0, x_1, \ldots, x_{2T})$. Thus, for each

$\mathbf{x} = (x_0, x_1, \ldots, x_{2T}) \in V$, set the colour of $\mathbf{x}$ to $f_{\mathcal{A}}(\mathbf{x})$. By assumption, $f_{\mathcal{A}}$ is a proper colouring of $\mathcal{N}_{k,T}$.

(ii) $\Rightarrow$ (i): Let φ be a proper vertex c-colouring of $\mathcal{N}_{k,T}$ and (u, v, w) be a path in a k-coloured directed cycle. Consider the following algorithm running on node v. First v gathers its radius-T local view $\mathbf{x}_v = (x_0, x_1, \ldots, x_{2T})$ where $u = x_{T-1}$, $v = x_T$, and $w = x_{T+1}$. Then the node v outputs the value $\varphi(\mathbf{x}_v)$.

The algorithm produces a proper c-colouring as the local views $\mathbf{x}_u$ and $\mathbf{x}_w$ of nodes u and w must be connected by an edge to $\mathbf{x}_v$ in the graph $\mathcal{N}_{k,T}$ and φ was a proper colouring of $\mathcal{N}_{k,T}$. □

In a similar fashion, it is easy to derive the following variant of Theorem 6.14.

Theorem 6.15. *Let c, n, and T be non-negative integers such that $3 \leq c \leq n$. There exists a deterministic distributed algorithm for c-colouring an n-cycle with unique identifiers from the set $[n]$ in T communication rounds if and only if the chromatic number of $\mathcal{L}_{n,T}$ is at most c.*

However, as the properly k-coloured neighbourhood graphs are somewhat more general, we will primarily concentrate on them. They will also turn out to be useful in improving the Cole–Vishkin colour reduction algorithm. These results are discussed in Section 8.

Our aim is to compute the chromatic numbers of neighbourhood graphs for various values of k and T. Before this, we will first make some observations on the structure of these graphs. These are useful in analysing the colourability of the graphs.

First, we already know an upper bound for the chromatic number due to the Cole–Vishkin colour reduction techniques presented in Section 5.2. This means that the chromatic number of $\mathcal{N}_{k,T}$ is at most $f^{(2T)}(k)$ where $f(k) = 2 \cdot \lceil \log k \rceil$. However, we will see that in many cases, the actual chromatic number is considerably lower.

Second, while the chromatic numbers of neighbourhood graphs grow as a function of k and T, the maximum clique size still remains constant in all such graphs. In Section 7, we will briefly discuss the implications of this fact. In essence, large cliques cannot be used in the neighbourhood graphs to lower bound the chromatic number.

Proposition 6.16. *For $k \geq 3$ and $T \geq 1$, the maximum clique size $\omega(\mathcal{N}_{k,T})$ of the neighbourhood graph $\mathcal{N}_{k,T}$ is 3 and there always exists a 3-clique in the graph.*

Proof. Let $k \geq 3$ and $T \geq 1$ be positive integers and $\mathcal{N}_{k,T} = (V, E)$ the corresponding neighbourhood graph. For convenience, let us define a relation $R \subset [k]^{4T+2}$ such that

$$(\mathbf{x}, \mathbf{y}) \in R \iff \mathbf{y} = (x_1, x_2, \ldots, x_{2T}, z) \text{ where } z \in [k] \setminus \{x_{2T}\}.$$

The relation simply states that $\mathbf{y}$ can be constructed by "left-shifting" $\mathbf{x}$ and setting y_{2T} such that $y_{2T} \neq x_{2T}$. Observe that $\{\mathbf{x}, \mathbf{y}\} \in E$ if and only if $(\mathbf{x}, \mathbf{y}) \in R$ or $(\mathbf{y}, \mathbf{x}) \in R$. Thus, as E is by definition constructed using the relation R, it suffices to consider the directed graph $\mathcal{G} = (V, R)$. Observe that we do *not* leave out any edges, only direct the edges in such a way that some edges become bidirectional.

To show that $\omega(\mathcal{N}_{k,T}) \geq 3$, we can construct a clique of size three in $\mathcal{G}$ as follows. Let $c_0, c_1, c_2 \in [k]$ be pairwise distinct colours. Now, let $\mathbf{v}$ be a node that corresponds to the $(2T+1)$-length tuple consisting of the repeating pattern $c_0, c_1, c_2, c_0, c_1, c_2, \ldots$, that is, $v_i = c_j$ when $i \equiv j \pmod 3$. The node $\mathbf{u}$ can be constructed by left-shifting $\mathbf{v}$ and setting u_{2T} as the next element in the repeating pattern such that $(\mathbf{v}, \mathbf{u}) \in R$. Similarly, $\mathbf{z}$ can be constructed by left-shifting $\mathbf{y}$, and $\mathbf{z}$ can be shifted to acquire $\mathbf{v}$. We have $(\mathbf{v}, \mathbf{u}), (\mathbf{u}, \mathbf{w}), (\mathbf{w}, \mathbf{v}) \in R$, that is, the nodes $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$ form a 3-clique in the neighbourhood graph.

To show that there are no 4-cliques in the graph, we first observe that the only way to form a 3-clique in $\mathcal{G}$ is to find such a directed triangle in the relation R as above. Suppose that for some $\mathbf{x}$ there are nodes $\mathbf{y}$ and $\mathbf{z}$ such that $(\mathbf{x}, \mathbf{y}), (\mathbf{x}, \mathbf{z}) \in R$ and these three nodes form a 3-clique in $\mathcal{G}$. By definition, $(\mathbf{x}, \mathbf{y}) \in R$ implies that $\mathbf{y} = (x_1, x_2, \ldots, x_{2T}, a)$ and $(\mathbf{x}, \mathbf{z}) \in R$ implies that $\mathbf{z} = (x_1, x_2, \ldots, x_{2T}, a')$. Next, consider the case $(\mathbf{y}, \mathbf{z}) \in R$. This implies that $\mathbf{z} = (y_1, y_2, \ldots, y_{2T}, b) = (x_2, x_3, \ldots, x_{2T}, a', b)$ and $x_1 = x_2$ which is a contradiction as the neighbourhoods were properly coloured. The case for $(\mathbf{z}, \mathbf{y}) \in R$ is symmetric.

For the upper bound $\omega(\mathcal{N}_{k,T}) < 4$, assume that there exists a clique K of size 4 in the graph $\mathcal{G}$. In the subgraph induced by the clique K, the degree of each node is odd. Thus, there must be a node $\mathbf{x}$ with out-degree 2 in the clique. Let $\mathbf{y}$ and $\mathbf{z}$ be nodes such that $(\mathbf{x}, \mathbf{y}), (\mathbf{x}, \mathbf{z}) \in R$. That is, $\{\mathbf{x}, \mathbf{y}, \mathbf{z}\}$ is a 3-clique. Inspection shows that either $R(\mathbf{y}, \mathbf{z})$ or $R(\mathbf{z}, \mathbf{y})$ must hold which is a contradiction as such 3-cliques were not possible in the graph $\mathcal{G}$. □

Proposition 6.17. *The neighbourhood graph $\mathcal{N}_{k,T}$ has $k(k-1)^{2T}$ nodes and maximum degree of $2(k-1)$.*

Proof. There are exactly $k(k-1)^{2T}$ different ways to construct a $(2T+1)$-tuple such that two successive elements in the tuple have different values. As these tuples

directly correspond to the k-coloured neighbourhoods, there are $k(k-1)^{2T}$ different nodes in the graph $\mathcal{N}_{k,T}$.

Consider a node $\mathbf{x} = (x_0, x_1, \ldots, x_{2T})$. There are at most $(k-1)$ choices for $\mathbf{y}$ such that $R(\mathbf{x}, \mathbf{y})$ holds. In addition, there are again at most $(k-1)$ choices such that $R(\mathbf{y}, \mathbf{x})$ holds. Thus, there can be no more than $2(k-1)$ edges connected to any node. □

While we defined neighbourhood graphs in directed cycles, neighbourhood graphs can be extended to other regular and bounded-degree graphs just as well by constructing radius-T views given in Definition 4.1.

Fix positive integers Δ, k, and T. Construct all the radius-T views in k-coloured graphs with maximum degree Δ. Use this as the set of nodes for the neighbourhood graph. Connect any two views (nodes) $x = \mathcal{V}(v, T)$ and $y = \mathcal{V}(u, T)$ with an edge if x has a subtree of height $T-1$ that is isomorphic up to the depth $T-1$ with tree y up to depth $T-1$. That is, connect two views if the central nodes can be adjacent in some k-coloured graph with maximum degree Δ.

Observe that a colouring of a neighbourhood graph for d-regular graphs also implies a colouring algorithm for graphs of maximum degree $\Delta = d$: if a node v has $\deg(v) < \Delta$, then the node v can simply simulate the missing $\Delta - \deg(v)$ neighbours with colours different from the colour of v.

While the number of views for k-coloured regular graphs is very large, it is still finite and thus the graph is finite and the chromatic number can be computed. The size of such neighbourhood graphs grows rapidly due to the various combinations of colours and port-numbers. Thus, colouring the graphs (and storing them) seems infeasible for most modern computers already for relatively small values of d, k and T unless some effective symmetry reduction and encoding are applied.

Nevertheless, such general neighbourhood graphs have been applied in proving both analytical lower and upper bounds for graph colouring. For example, Kuhn considered one-round colour reduction algorithms in the broadcast model with d-regular graphs and showed both upper and lower bounds for these types of algorithms [KW06].

7 Computing the chromatic number

In this section, we will describe the methods used for computing the chromatic numbers of neighbourhood graphs and finding small colourings of the neighbourhood graphs.

In this work, we solve the graph colouring problems using so-called propositional satisfiability solvers, or SAT solvers for short. These programs solve the satisfiability problem for formulas in propositional logic (also known as the *SAT problem*). The decision versions of graph colouring problems given in Section 3.5 can be encoded as SAT instances.

Although there are algorithms designed explicitly for computing optimal colourings in graphs, the use of SAT solvers turned out to be a competitive alternative for these. In fact, recent studies have shown promising results regarding the use of SAT solvers for graph colouring [Pre04, Van08].

7.1 The propositional satisfiability problem

Let us begin by defining the necessary concepts for propositional logic and the SAT problem. A Boolean *variable* can take one of two values: either the variable is set to false or true. These are denoted by 0 and 1, respectively. A *formula* in propositional logic consists of a set of Boolean variables $\{x_0, x_1, \ldots, x_{n-1}\}$ which are connected using parentheses and two connectives $\neg$ (negation) and $\wedge$ (conjunction). In addition to these two connectives, it is customary to define various short-hands such as $\vee$ (disjunction), $\rightarrow$ (implication), and $\leftrightarrow$ (equivalence). All of these can be easily defined using only conjunction, negation, and parentheses as we will see.

An example of a propositional formula over the variables $\{x_0, x_1\}$ is the formula

$$\psi = \neg(\neg x_0 \wedge \neg x_1).$$

A *sentence* is a formula where each variable has been assigned a value. For example, assigning $x_0 = 0$ and $x_1 = 1$ in the previous formula ψ results in a sentence that is true. The formula given by ψ is often written in a more succinct form as $x_0 \vee x_1$ which is true if either the variables x_0, x_1, or both have the value 1.

In general, given two formulas ψ and ϑ, we denote disjunction, implication and equivalence as follows. The formula $\psi \vee \vartheta$ is a short-hand to

$$\neg(\neg\psi \wedge \neg\vartheta),$$

whereas implication $\psi \to \vartheta$ corresponds to the formula

$$\neg\psi \vee \vartheta,$$

and equivalence $\psi \leftrightarrow \vartheta$ is the same as

$$(\psi \to \vartheta) \wedge (\vartheta \to \psi).$$

The Boolean satisfiability problem for propositional logic (SAT) is defined as follows.

Problem 4 (SAT)**.** Given a formula ψ over the set $X = \{x_0, x_1, \ldots, x_{n-1}\}$ of variable symbols, decide whether there exists a truth assignment $\tau\colon X \to \{0,1\}$ such that formula ψ is true when the variable symbols are interpreted according to τ. If such an assignment exists, ψ is said to be *satisfiable* and otherwise *unsatisfiable*.

The truth assignment function τ can be also regarded as a binary vector $\mathbf{t} \in \{0,1\}^n$. In the function version of SAT the output is either the truth assignment or false if the given formula is unsatisfiable. In practice, most SAT solvers require that the input formula ψ is given in *conjunctive normal form*.

Definition 7.1 (CNF)**.** A propositional formula ψ over variables $\{x_0, x_1, \ldots, x_{n-1}\}$ is in *conjunctive normal form*, if $\psi = C_0 \wedge C_1 \wedge \cdots \wedge C_{m-1}$ is a conjunction of clauses. A *clause* is a disjunction of *literals* $C_i = \ell_0 \vee \ell_1 \vee \cdots \vee \ell_{k-1}$ where a literal ℓ_i is either a variable x_i or its negation $\neg x_i$.

A formula in conjunctive normal form is said to be a CNF formula for short. Any formula in propositional logic can be converted into a CNF formula. However, we will be encoding our graph colouring instances directly into CNF formulas as this is the most common input format for SAT solvers. A CNF formula ψ can be considered as a set of clauses $\psi = \{C_0, C_1, \ldots, C_{m-1}\}$ where each clause is a set of literals. An empty clause is *false* while an empty formula $\psi = \emptyset$ is *true*.

The Boolean satisfiability problem was the first problem proven to be NP-complete [Coo71]. The optimization version of SAT known as MAX-SAT is also hard to approximate [Vaz01, Ch. 29.3]. Despite this, there have been numerous advances in algorithms for SAT. Although the SAT problem remains intractable in theory, *in practice* the current algorithms and techniques seem to solve many "natural" instances arising from real-world problems relatively efficiently. This has led to an interest in applying SAT solver technology to hard computational problems ranging from traditional combinatorial problems [GS02, Van08, JK10] to model checking and to applications in bioinformatics [LMS08].

7.2 Encoding k-colourability as SAT

For the k-colouring problem, various CNF encodings have been studied. These differ in the number of variables, clauses and literals used in the produced formulas. In addition, the effectiveness (i.e., running time and memory consumption) of these encodings often varies depending on to the solver as different solvers use different algorithms and heuristics [Wal00, Pre04, Van08].

For all the following encodings, let $\mathcal{G} = (V, E)$ be the input graph and k the input parameter. In addition, we use the short-hands $n = |V|$ and $m = |E|$.

Direct encoding. We begin with the simplest encoding known as the *direct encoding* [Pre04] or as the *extended encoding* [SHvM09]. For this encoding, the set of variables will be $\{x_{v,i} : v \in V, i \in [k]\}$. When the variable $x_{v,i}$ is true, the node v is assigned the colour i.

Our goal is to construct a formula $\psi = \mathbf{direct}(\mathcal{G}, k)$ which is satisfiable if and only if (i) each node is given some colour, (ii) no node is assigned more than one colour, and (iii) adjacent nodes have different colours.

For condition (i), we add the following clause for all the nodes $v \in V$

$$\bigvee_{i \in [k]} x_{v,i} = x_{v,0} \vee \cdots \vee x_{v,k-1}.$$

In order to satisfy condition (ii), for every node $v \in V$ and each colour $i, j \in [k]$ where $i \neq j$, we add the clause

$$\neg x_{v,i} \vee \neg x_{v,j}.$$

Finally for condition (iii), we require that adjacent nodes have different colours. That is, for every edge $\{u, v\} \in E$ and each colour $i \in [k]$ we add the clause

$$\neg x_{u,i} \vee \neg x_{v,i}.$$

In total, the direct encoding requires nk variables and $n + nk^2 + mk$ clauses. Most of the clauses are relatively short; there are $nk^2 + mk$ two-literal clauses and n clauses of length k. The formula $\mathbf{direct}(\mathcal{G}, k)$ is satisfiable if and only if the variable assignment explicitly defines a valid k-colouring for the graph $\mathcal{G}$. Thus, if the formula is satisfiable, it is easy to extract the colouring from the output of the solver.

Multivalued encoding. It is easy to see that the set (ii) of clauses in the previous encoding does not affect the satisfiability of the formula. If a node v is assigned more than one colour, it suffices to choose any of them as the colour for v. The chosen colour cannot conflict with any of the possible colours given to the neighbours of v. Thus, we can omit the redundant clauses in the previous encoding. This variant of the direct encoding is sometimes called *multivalued encoding* [Pre04] or traditional encoding [Van08].

Logarithmic encoding. The logarithmic encoding [Van08, Pre04] was devised to reduce the number of required variables in the encoded SAT instance. As the colours are assumed to be non-negative integers from the set $\{0, 1, \ldots, k-1\}$, we can interpret each colour as a bit string of length $L = \lceil \log k \rceil$. Unlike in the two previous encodings, the logarithmic encoding does not have a variable $x_{v,c}$ for each node-colour pair (v, c) where $v \in V, c \in [k]$. Instead, there is a variable $x_{v,i}$ for each $v \in V$ and $i \in [L]$. The variable $x_{v,i}$ denotes whether the node v has a colour with the ith bit set to 1. This reduces the number of variables from nk to nL.

In the logarithmic encoding, the requirement that adjacent nodes u and v have different colours is stated as "the colours of u and v must have at least one differing bit". As a consequence of this formulation, it is not necessary to explicitly generate clauses that force u and v to have some colour. The requirement that adjacent nodes have at least one differing bit can be easily stated using *exclusive disjunction* denoted by the $\oplus$ operator (XOR). The exclusive disjunction is defined as

$$a \oplus b = (a \wedge \neg b) \vee (\neg a \wedge b).$$

Now the adjacency requirement for an edge $\{u, v\} \in E$ can be enforced with the formula

$$\vartheta(u, v) = \bigvee_{i \in [L]} x_{u,i} \oplus x_{v,i}.$$

By applying the definition of exclusive disjunction, the above formula expands into

$$\vartheta(u, v) = \bigvee_{i \in [L]} (x_{u,i} \wedge \neg x_{v,i}) \vee (\neg x_{u,i} \wedge x_{v,i}).$$

Unfortunately, the above formula is not yet in conjunctive normal form. On the other hand, it is in *negation normal form*: all the negations are in front of the literals and the formula uses only conjunction and disjunction. We can now use the standard procedure for transforming formulas in negation normal form into conjunctive normal

form by applying the distributivity law of propositional logic:

$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c).$$

This yields a logically equivalent CNF formula $\vartheta'(u, v)$ with 2^L clauses consisting of $2L$ literals.

For a graph $\mathcal{G}$ and a colourability parameter k, the encoded CNF formula $\mathbf{log}(\mathcal{G}, k)$ is then

$$\mathbf{log}(\mathcal{G}, k) = \bigwedge_{\{u,v\} \in E} \vartheta'(u, v).$$

For practical purposes, the following alternative characterisation for the CNF formula $\vartheta'(u, v)$ is useful. The formula $\vartheta'(u, v)$ can be considered as a conjunction of clauses $\{C_c \colon c \in [k]\}$ where clause C_c has literals $\neg x_{u,i}$ and $\neg x_{v,i}$ whenever the ith bit of c is 1. For all the 0 bits, literals $x_{v,i}$ and $x_{u,i}$ are added. For example, the clause C_3 for an edge $\{v, u\}$ is

$$\neg x_{u,0} \vee \neg x_{v,0} \vee \neg x_{u,1} \vee \neg x_{v,1} \vee x_{u,2} \vee x_{v,2} \vee \cdots \vee x_{u,L-1} \vee x_{v,L-1}.$$

Once the adjacency requirement has been encoded, we must also ensure that colours larger than k are not used. If k is a power of two, none of the $\log k$ bit combinations produce an invalid colour. Otherwise, we must add clauses to disallow the prohibited bit combinations. The simplest way is to list all the $2^L - k$ prohibited bit combinations for each node, though more efficient encodings exist. In addition, the adjacency requirement can be encoded using fewer clauses when the exclusive disjunction is encoded using additional variables. For a more thorough comparison of different variants of logarithmic encoding, see [Pre04, Van08].

Evaluation. The encodings described in this section have been studied in both experimental [Pre04, Van08] and theoretical [Wal00] settings. While the experimental studies have focused on graph colouring problems, the theoretical results consider encoding general constraint satisfaction problems into SAT.

Some experimental results have shown that the multivalued encoding generally performs better than the direct encoding on satisfiable instances [Pre04, Van08]. Prestwich conjectured that this is due to the fact that the multivalued encoding has usually a higher solution density [Pre04]. That is, the number of feasible solutions for formulas attained by multivalued encoding is larger than the number of feasible solutions for directly encoded formulas.

Another benefit of the multivalued encoding over the direct encoding is the smaller length of the produced formulas. While in both encodings the number of variables increases linearly with the parameter k, formulas using the direct encoding have $n + nk^2 + mk$ clauses, whereas the multivalued formulas have only $mk + n$ clauses. This is a considerable improvement for large but sparse graphs and for small values of k.

On the other hand, the logarithmic encoding aims to ensure a smaller search space by having less variables than in the direct and multivalued encodings. Van Gelder [Van08] reports that the variations of logarithmic encoding have a rather poor performance with unsatisfiable instances. For instances encoded with the direct or multivalued encodings, SAT solvers can show the unsatisfiability using unit clause propagation, while the logarithmic encodings seem to "conceal" this approach from the solver. This seems to be in line with Walsh's [Wal00] results stating that unit clause propagation is less efficient when using the logarithmic encoding for constraint satisfaction problems.

Furthermore, the experimental results of Van Gelder [Van08] indicate that the multivalued encoding outperforms logarithmic encodings most of the time in both unsatisfiable and satisfiable instances. However, in a different study conducted by Prestwich [Pre04], the author infers a somewhat different notion on the performance of the logarithmic encoding and concludes that the encoding "may not be as bad as expected".

7.3 Finding an optimal colouring

Now that we know how to solve the k-colourability problem via Boolean satisfiability, it is relatively straightforward to solve the chromatic number problem as well. For a given graph $\mathcal{G}$, we can simply solve the k-colourability problem iteratively for each $k = 1, 2, \ldots, \Delta + 1$ until the resulting formula is satisfiable. Once a satisfying truth assignment for the variables is found, the colouring function $\varphi\colon V \to [k]$ can be constructed by interpreting the variables according to the chosen encoding.

If the chromatic number is high, a considerable amount of work is spent on trying to solve the unsatisfiable instances which usually take a long time to solve. One reason for this is that in an unsatisfiable case, all permutations of colours may be tried unless this is prevented or a counter-example such as a $(k + 1)$-clique is identified by the solver. In addition, a high chromatic number can be a purely global phenomenon [Erd59, Lov68]. Hence, it may be that the solvers cannot

easily distinguish unsatisfiable instances without enumerating a large portion of the colourings.

Instead of an iterative search, it is also possible find the optimal k by applying binary search. On the other hand, for k-colouring instances where k is considerably larger than $\chi(\mathcal{G})$, it has been reported that the overhead generated by large formulas degrades performance [Van08]. Thus, the simple iterative method often works best if we can find good lower bounds on the chromatic number.

As discussed in Section 3.4, the chromatic number can be bounded in various ways by either exploiting the properties of the graph family at hand or finding large cliques. Unfortunately, even approximating the maximum clique size is hard [Vaz01, Ch. 29]. Nevertheless, greedily finding *some* clique already seems to improve the performance drastically [Van08, SHvM09].

Given a clique K in the graph $\mathcal{G}$, we can force a partial colouring for the graph $\mathcal{G}$ by designating a distinct colour for each node in the clique K. This prunes the search space explored by the SAT solver to some extent. For example in the direct and multivalued encodings, this can be done by adding the unit clause $(x_{v,i})$ for each $v \in K$ and distinct colour $i \in \{0, 1, \ldots, |K| - 1\}$. Moreover, it is possible to partially colour other dense subgraphs besides cliques as well [Van08].

In addition to these preprocessing techniques, there have been some recent work on so-called dynamic symmetry breaking methods. For example, Schaafsma et al. [SHvM09, Sch09] studied how to modify SAT solvers to work better on vertex colouring instances. They implemented a specialized conflict-driven SAT solver that essentially uses conflict clauses where one conflict clause can cover any permutation of a improper partial colouring.

The experimental results presented by Schaafsma et al. suggest that this type of dynamic symmetry breaking combined with the usual preprocessing techniques yields performance improvements. In particular, unsatisfiable instances may be faster to solve [SHvM09]. However, so far the dynamic symmetry breaking techniques are both problem- and solver-specific, and the only implementation as of the time of writing is the MiniMerge [Sch09] solver.

8 Improved bounds for cycle and tree colouring

This section presents the experimental results attained from computing the chromatic numbers of various neighbourhood graphs. First, we outline the new results regarding explicit upper and lower bounds for colour reduction in directed cycles. We also discuss some observations on the SAT approach.

Second, using the new explicit bounds, we give improved upper and lower bounds for the general case of colour reduction in directed k-coloured cycles. In addition, for all practical values of k, the new algorithms improve the performance of the standard Cole–Vishkin style algorithms by a factor of two or almost two. Finally, we analyse the tightness of these new results and discuss possible approaches for closing the remaining (small) gap between the upper and lower bounds.

8.1 Colourings for the neighbourhood graphs

The chromatic numbers for neighbourhood graphs were analysed using the techniques described in Section 7. A computer program was set up to generate the neighbourhood graphs and iteratively seek upper and lower bounds on the chromatic numbers using SAT solvers.

The c-colouring instances were encoded using both the multivalued and direct encodings. The logarithmic encoding was not used as it turned out that most satisfiable instances were not difficult to solve and it has been argued that multivalued encoding outperforms log encoding [Wal00, Van08]. In addition, the value of c is small in all the relevant cases and thus the impact of using fewer variables seems somewhat negligible.

As the maximum clique size remains constant for all neighbourhood graphs $\mathcal{N}_{k,T}$, colouring large cliques as a preprocessing step cannot be used to efficiently prune the search space. This may be the reason why the SAT solvers could not prove unsatisfiability of various c-colouring instances of $\mathcal{N}_{k,1}$ where $k \geq 25$ and $c \leq 4$. As a prospective research topic, it would be interesting to see if more efficient symmetry-breaking techniques can be successfully applied in order to find better lower bounds for the chromatic numbers.

Nevertheless, the SAT solvers managed to find solutions to most colouring instances with relative ease. In particular, parallel solvers, such as ManySAT, found solutions to the largest colouring instances, e.g., 5-colouring of the graph $\mathcal{N}_{70,1}$ with multivalued

Solver	Version	Website
clasp [GKNS07]	1.3.4	http://www.cs.uni-potsdam.de/clasp/
lingeling [Bie10]	276	http://fmv.jku.at/lingeling/
ManySAT [HJS09]	1.0	http://www.cril.univ-artois.fr/~jabbour/manysat.htm
MiniSAT [ES03]	2.2	http://minisat.se/
MiniMerge [SHvM09]	1.0	–
picosat [Bie10]	913	http://fmv.jku.at/picosat/
precosat [Bie10]	570	http://fmv.jku.at/precosat/
smallk	8/29/99	http://webdocs.cs.ualberta.ca/~joe/Coloring/
Minion [GJM06]	0.11	http://minion.sourceforge.net/

Table 1: The solvers used to compute chromatic numbers. All are SAT solvers with the exception of 'smallk' (a vertex colouring software) and 'Minion' (a CSP solver).

encoding, within reasonable time. The small unsatisfiable instances were solved by all SAT solvers quite rapidly.

Moreover, attempts with other solver techniques such as integer programming and general constraint satisfaction solvers with straightforward encodings did not seem to bear fruit in comparison to the SAT solver approach. Thus, while no thorough comparison study was made, it seems that SAT solvers are competitive with regard to solving graph colouring problems. The different solvers that were experimented with are listed in Table 1.

The solvers found exact chromatic numbers for all $\mathcal{N}_{k,1}$ where $4 \leq k \leq 24$. For larger values of k, near-optimal results were found up to $k = 70$; we could not prove that the results are *not* optimal. If $\chi(\mathcal{N}_{25,1}) > 4$ holds, then for $k \in \{25, \ldots, 70\}$ the chromatic numbers of $\mathcal{N}_{k,1}$ are indeed exact. The 4-colourability of $\mathcal{N}_{25,1}$ was attacked with multitude of solvers but none of the solvers managed to find a solution during several weeks of computation on an Intel Xeon E5540 processor with 32 gigabytes of RAM. In contrast, the satisfiable instances were usually solved within a few hours at most.

Table 2 summarises the known values for the chromatic numbers of neighbourhood graphs. For values of k where the exact value of $\chi(\mathcal{N}_{k,1})$ is not known, the corresponding bounds are given. The vertex colourings that yield the upper bounds are available online [Ryb11].

Using these computational results, it is possible to derive new lower and upper bound results on the complexity of distributed colour reduction in k-coloured directed

k	Lower bound	Upper bound	Exact
4	3	3	Yes
5	4	4	Yes
6	4	4	Yes
24	4	4	Yes
25	4	5	No
70	4	5	No

Table 2: Upper and lower bounds for the chromatic number of $\mathcal{N}_{k,1}$. Rows with matching upper and lower bounds are marked as exact.

cycles. We begin by observing the following lemmas which help us in analysing the complexity of distributed colouring algorithms. The first result follows directly from Theorem 6.14 and the fact that $\chi(\mathcal{N}_{5,1}) > 3$.

Lemma 8.1. *It is not possible to reduce the number of colours from* 5 *to* 3 *in a directed cycle in one communication round.*

Interestingly, it takes as many communication rounds to reduce the number of colours from 5 as it does from 24.

Lemma 8.2. *A* 24*-coloured directed cycle can be* 3*-coloured in* 2 *communication rounds.*

Proof. As $\chi(\mathcal{N}_{24,1}) = 4$ the cycle can be 4-coloured in one communication round. The 4-colouring can be reduced to a 3-colouring using the greedy colour reduction algorithm. □

8.2 The value of local information

The chromatic numbers of neighbourhood graphs $\mathcal{N}_{k,T}$ yield bounds for distributed colour reduction algorithms with running time T in directed k-coloured cycles. For comparison, we also explore two other cases where the nodes are given less information and one where the nodes are given more information.

In the first case, the nodes can only use information available from their immediate predecessor. In k-coloured directed cycles, this situation is captured by the neighbourhood graph $\mathcal{S}_k = (V, E)$ where $V = \{(u, v) : u, v \in [k]\}$ and there is an edge between $(u, v), (u', v') \in V$ if $v = u'$.

k	Lower bound	Upper bound	Exact
4	4	4	Yes
6	4	4	Yes
7	5	5	Yes
11	5	6	No
20	5	6	No

Table 3: Upper and lower bounds for the chromatic number of $\mathcal{S}_k$.

For example, the one-round Cole–Vishkin $\mathcal{O}(\log k)$-colour reduction used in Algorithm 3 corresponds to this case. While the Cole–Vishkin algorithm cannot achieve a 6-colouring when $k > 8$, it is still possible to achieve a 6-colouring when $k \leq 20$ with other algorithms that only utilize information available from the predecessor. The bounds for chromatic numbers for the graph $\mathcal{S}_k$ are given in Table 3.

In a sense, an algorithm that only utilizes the colour of a single predecessor is a "half-round" algorithm: we can use the same technique as in Theorem 5.4 to simulate two iterations of such an algorithm in a single communication round. For example, we attain a one-round colour reduction algorithm from 20 to 4 colours as follows.

In one round, any node v in a directed cycle can gather information from its local neighbourhood (u, v, w). Node v can first apply the colour reduction from 20 colours to 6 colours since the colour of its predecessor u is known. Similarly, node v can calculate the new colour of w as v itself is the predecessor of node w. After computing the new 6-colouring, node v can apply the colour reduction from 6 to 4 colours by simulating w as its predecessor.

From Table 3 we can also see that if a node only looks at its predecessor in a directed cycle, reducing the number of colours from 4 to 3 is not possible. Indeed, this conforms to the previous result that $\mathcal{N}_{5,1}$ is not 3-colourable: if this were not the case, an algorithm could reduce from 5 to 4 colours and then from 4 to 3 in one round. We can also see that the Cole–Vishkin colour reduction algorithm is not an optimal half-round algorithm as it stops working after the cycle has been 6-coloured. However, there is an algorithm that can reduce the colours from 6 to 4 just by using the information from the predecessor.

The "half-round" colour reduction algorithms are listed in Table 4 and Table 5. The tables have been constructed from the proper vertex colourings of the graphs $\mathcal{S}_{20}$ and $\mathcal{S}_6$. After a node v has received the colour $\varphi(u)$ of its predecessor u, node v can choose a new colour from the cell in row number $\varphi(v)$ and column number $\varphi(u)$.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	-	1	1	0	0	0	2	0	2	0	1	1	1	1	1	2	2	1	0	1
1	5	-	2	5	2	0	2	0	5	0	5	2	5	0	0	2	5	0	0	0
2	4	4	-	0	0	0	4	0	3	0	3	3	4	4	0	4	4	0	0	0
3	4	1	1	-	4	4	1	2	2	4	1	2	1	1	1	2	2	2	2	1
4	5	1	1	5	-	3	1	3	5	5	5	1	1	1	1	3	5	1	5	5
5	5	1	2	5	2	-	1	2	2	5	1	1	1	1	1	2	5	1	2	1
6	3	3	5	3	0	3	-	3	3	0	5	3	5	0	0	3	5	0	0	5
7	5	1	1	5	4	4	1	-	5	5	1	1	1	1	1	4	5	1	5	1
8	4	1	1	0	4	0	4	0	-	4	1	1	1	1	1	4	4	1	0	1
9	3	1	1	3	2	3	2	3	3	-	1	1	1	1	1	2	2	1	2	1
10	4	4	2	0	4	4	4	0	2	4	-	2	4	4	0	4	4	2	0	0
11	5	4	5	0	0	0	4	0	5	0	5	-	5	0	0	4	5	0	5	0
12	3	3	2	0	0	0	2	0	3	0	3	3	-	0	0	3	2	0	0	0
13	3	3	2	5	2	3	2	3	5	5	5	3	5	-	3	3	5	2	5	5
14	5	4	2	5	4	4	4	2	5	4	5	2	5	4	-	4	5	2	5	5
15	5	1	5	0	0	0	1	0	5	0	1	1	1	1	1	-	5	1	0	1
16	3	1	1	0	0	0	1	0	3	0	1	1	1	1	1	3	-	1	0	0
17	3	4	5	3	4	3	4	3	3	4	3	3	4	4	3	4	4	-	5	5
18	4	4	1	3	4	4	4	3	3	4	1	1	4	4	1	4	4	1	-	1
19	3	3	2	3	4	4	4	3	3	4	3	3	4	4	3	4	4	2	2	-

Table 4: The colour reduction scheme from 20 to 6 colours using only the information from the predecessor. For an edge (u, v) in a directed cycle, node v chooses the colour in the cell $(\varphi(v), \varphi(u))$.

	0	1	2	3	4	5
0	-	1	0	1	0	1
1	2	-	2	2	0	0
2	3	1	-	1	3	1
3	3	3	0	-	3	0
4	2	1	2	1	-	1
5	3	3	2	2	3	-

Table 5: The colour reduction scheme from 6 to 4 colours using only information available from the predecessor. For an edge (u, v) in a directed cycle, node v chooses the colour in the cell $(\varphi(v), \varphi(u))$.

Observe that the matrices give a proper colouring since for any $i \in [k]$, all the elements in the ith row are different from the elements in the ith column. This ensures that a colour i node always chooses a different colour than its successor. A very similar colour reduction algorithm was already used by Naor and Stockmeyer for computing weak colourings [NS95]. Their algorithm reduces the number of colours from k to $\log(k) + \mathcal{O}(\log \log k)$.

In contrast to restricting the information available to a node, the communication graph may be coloured in a way that allows the nodes to make stronger assumptions of their neighbourhoods. For example, consider the situation where the node knows the colours are unique in every T-neighbourhood, that is the graph has a radius-T vertex colouring. This corresponds to colouring the neighbourhood graph $\mathcal{L}_{k,T}$.

For the case $T = 1$, it turns out that the barrier of reducing from 5 colours to 3 can be broken. In fact, there is enough information to admit colour reduction from 6 colours to 3. However, for $k = 7$ this is not possible any more. Similarly to the case with a proper k-colouring, 4-colouring is possible in one round at least up to $k = 24$ in a directed cycle with a radius-1 vertex k-colouring.

8.3 Faster deterministic 3-colouring

We now derive faster deterministic 3-colouring algorithms for directed cycles and directed rooted trees using the colourings of cycle neighbourhood graphs.

For 3-colouring a directed cycle, the Cole–Vishkin algorithms presented in Section 5.2 require $\lceil \frac{1}{2} \log^* k \rceil + 3$ communication rounds. In the case of 3-colouring directed pseudotrees, the algorithms had a running time of $\log^* k + 6$ communication rounds.

While these algorithms are optimal up to additive terms, for small values of k, the additive term dominates the running time of the algorithm. In particular, the efficacy of the Cole–Vishkin algorithms greatly decreases when the number of colours is close to 6. In fact, after reducing the number of colours to 6, the Cole–Vishkin method cannot be used any longer. From this point onward, the algorithm switches to a linear time greedy algorithm in order to get rid of the last few colours.

In most practical cases, optimizing the additive terms result in a significant speed-up due the slow growth of the $\log^* k$ term. The additive terms can be improved by utilizing the colourings of neighbourhood graphs. For now, we will call the traditional iterative Cole–Vishkin style colour reduction techniques as *naive* Cole–Vishkin algorithms.

As an example, consider cycles that use 2048-bit identifiers. The 2048-bit identifiers correspond to a k-colouring where $k = 2^{2048}$. To 3-colour such a cycle, the naive version of Cole–Vishkin uses

$$\left\lceil \frac{1}{2} \log^* k \right\rceil + 3 = \left\lceil \frac{5}{2} \right\rceil + 3 = 6$$

communication rounds.

However, we can do much better. First, run the Cole–Vishkin algorithm for one communication round. This reduces the colouring from 2^{2048} to 24. Now, as $\chi(\mathcal{N}_{24,1}) = 4$, we can reduce the 24-colouring to a 4-colouring in one step and finally, in one more round reduce it to the 3-colouring. In total, this requires three communication rounds. We gained an improvement of factor two in the running time.

In practice, the identifiers used in current-day distributed systems have considerably smaller identifiers: IPv4 uses 32-bit addresses, MAC-addresses are 48-bit, and IPv6 uses 128-bit addresses. For 128-bit identifiers in cycles, the naive algorithm requires 5 rounds, whereas the improved algorithm requires only 3 communication rounds.

For general $k \geq 3$ we get the following upper bound for 3-colouring directed cycles.

Theorem 8.3. *Directed k-coloured cycles can be 3-coloured in $\left\lceil \frac{1}{2} \log^* k \right\rceil + 1$ communication rounds.*

Proof. In the proof of Lemma 5.2 it is shown that 12-colouring takes at most $\log^* k - 2$ communication rounds. Using the double-speed trick in cycles, the cycle can be 12-coloured in at most $\left\lceil \frac{1}{2} \log^* k \right\rceil - 1$ rounds. By Lemma 8.2, the 12-colouring can be reduced to a 3-colouring in two communication rounds. □

Alternatively, we can apply the one-round colour reduction algorithm from 20 to 4 colours given in Tables 4 and 5 instead of referring to Lemma 8.2 in the above proof.

Using any coloured graph $\mathcal{N}_{k,1}$ requires that the nodes have the graph stored in the memory. In practice, this means that the nodes are given precomputed lookup tables. However, the size of the lookup table is rather small as $\mathcal{N}_{k,1}$ contains $k(k-1)^2$ nodes as shown in Proposition 6.17. For example in the case $k = 24$, the lookup table for 4-colouring requires $\log(4)k(k-1)^2 = 2 \cdot 24 \cdot 23^2 = 25392$ bits which amount to 3174 bytes.

8.4 Closing the gap in lower and upper bound results

Let us now compare the result of Theorem 8.3 with Linial's lower bound of $\frac{1}{2}(\log^* n - 3)$. The major difference between the lower and upper bound results is in the additive constant and hence we will focus on analysing it. Moreover, Linial's lower bound holds for the model with unique identifiers whereas our upper bound holds in the weaker model of k-coloured networks. Thus, we examine the latter model.

For all $k \geq 3$ and for some constant c, it is known that $\frac{1}{2}(\log^* k + c)$ communication rounds suffice to colour directed k-coloured cycle with at most three colours. The results already established in the literature show that $c \in [-3, 7]$: the naive Cole–Vishkin algorithm provides an upper bound $c \leq 7$, whereas Linial's lower bound states that $c \geq -3$ for all n.

However, given the new computational results, we can tighten the gap. First of all, Theorem 8.3 gives us an upper bound of 3 for the constant c as

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

For the lower bound, we know by Lemma 8.1 that for $k = 5$ the algorithm needs at least two communication rounds. This in turn implies that $c \geq 1$ as

$$\frac{1}{2}(\log^* 5 + c) = \frac{1}{2}(3 + c) \geq 2 \implies c \geq 4 - 3.$$

Therefore, we can assert that $c \in [1, 3]$ for all k-coloured directed cycles. It seems that with these results alone, we cannot directly improve the bounds.

To improve the upper bound, one could try to show that it is possible to reduce a 63-colouring into a 3-colouring in two communication rounds, that is, show that $\chi(\mathcal{N}_{63,2}) = 3$. The analysis in Lemma 5.2 and the double-speed trick given Theorem 5.4 show that 63-colouring can be achieved in $\frac{1}{2}(\log^* k - 1)$ communication rounds. If $\chi(\mathcal{N}_{63,2}) = 3$ then $\frac{1}{2}(\log^* k + 1)$ rounds would suffice to 3-colour a k-coloured directed cycle yielding a tight result that the additive term $c = 1$. On the other hand, as there is no proof for that $\mathcal{N}_{63,1}$ is not 4-colourable, it may be possible to use $\mathcal{N}_{63,1}$ to bound the value of c to at most 2.

To tighten the gap from below, there are several alternatives. First, if for some $k \in [17, 2^{16}]$ 3-colouring requires at least 3 rounds, then $c \geq 2$. Second, if 3-colouring a directed cycle requires at least 4 rounds for some $k \in [2^{2^{16}} + 1, 2^{2^{65536}}]$, then $c \geq 2$. Finally, to show that $c = 3$, it suffices that for some $k \in [2^{16} + 1, 2^{2^{16}}]$ 3-colouring takes at least 4 communication rounds.

Unfortunately, for such large values of k, computing the chromatic numbers becomes infeasible using the methods described so far. Increasing the time parameter T does not seem to help either as the graph size grows exponentially with T. Already explicitly storing the graph itself would require considerable amounts of memory. For example, computing $\chi(\mathcal{N}_{25,2})$ was out of reach for the colouring applications used in this work. Furthermore, none of the solvers managed to show that $\chi(\mathcal{N}_{k,1}) > 4$ for any $k \in [25, 70]$.

It may be possible to improve the above result by computationally or analytically bounding the chromatic numbers of neighbourhood graphs where $T = 2$. That is, it may be possible that by inspecting a larger local neighbourhood, an iterative colour reduction algorithm may perform better.

8.5 Extensions to other distributed problems

So far, we have only considered the distributed graph colouring problem. In this section, we outline some ideas on prospective research regarding exact bounds for other distributed graph problems. Whereas the existence of distributed colour reduction algorithms easily translates into the chromatic number problem for neighbourhood graphs, other problems, such as maximal matchings, maximal independent sets, minimal dominating sets, and minimal vertex covers, seem to require a somewhat different approach. We now will examine the first two problems.

Consider the maximal independent set problem in k-coloured directed cycles. One might try to use the same neighbourhood graph constructions parametrized by the number of colours k and running time T as before. If we have a deterministic distributed algorithm for maximal independent sets in k-coloured cycles, it is relatively easy to see that this produces a maximal independent set in the neighbourhood graph.

However, a feasible solution to the maximal (or maximum) independent set problem in the neighbourhood graph does not imply a distributed algorithm for the maximal independent set problem. For example, consider the maximum independent set I for the cycle neighbourhood graph $\mathcal{N}_{3,1}$ given in Figure 18a. The solution I does not yield a distributed algorithm for maximal independent sets in cycles: a counter-example is given Figure 18c.

Therefore, the implication holds only for the other direction: a distributed algorithm produces a maximal independent set in the neighbourhood graph, but a maximal

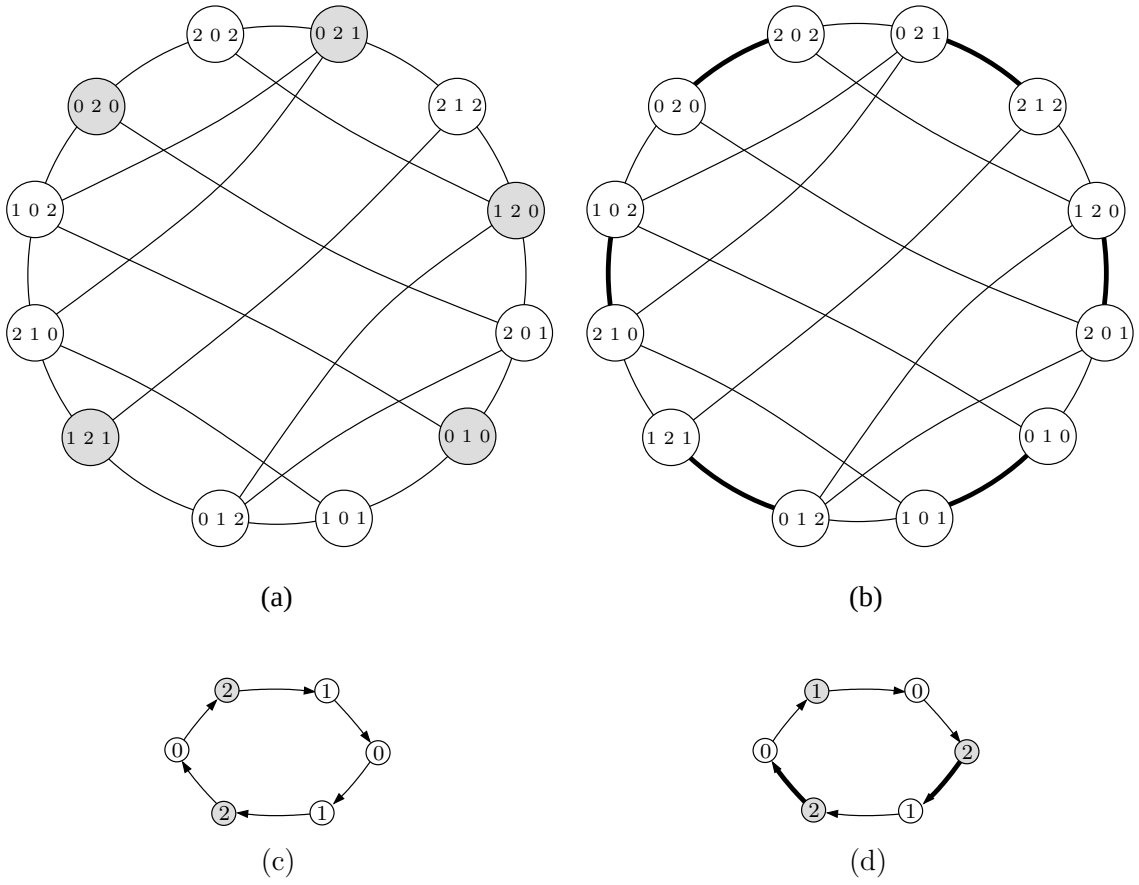


Figure 18: A maximum independent set in $\mathcal{N}_{k,T}$ does not guarantee a T-time algorithm that computes a maximal independent set in a k-coloured cycle. This also applies to the maximal matching problem. In the two bottom cycles, the numbers denote the colouring and the nodes in an independent set are grey. (a) A maximum independent set I in $\mathcal{N}_{3,1}$. (b) A maximum matching M in $\mathcal{N}_{3,1}$. (c) A 3-coloured cycle where I does not yield a maximal independent set and M produces an empty matching. (d) A 3-coloured cycle where I happens to produce a maximal independent set but the matching is not maximal.

independent in the neighbourhood graph does not yield a distributed algorithm. However, the solution yields a distributed algorithm that computes *some* independent set. Of course, if the neighbourhood graph is c-colourable, there exists an algorithm for computing maximal independent sets in $(T+k-1)$ rounds: the communication graph can be c-coloured and then a maximal independent set can be greedily constructed.

In the case of matchings, a maximal or maximum matching in the neighbourhood graph does not imply a distributed algorithm for the maximal matching problem. Figure 18b illustrates a maximum matching in the neighbourhood graph $\mathcal{N}_{3,1}$. Again, the solution produces an algorithm that finds a matching in a cycle but it is not maximal. For example, consider the graph depicted in Figure 18c: the matching is empty.

Nevertheless, there is a way to characterise these problems with the help of a slightly modified neighbourhood graph construction. This construction allows us to set constraints on the edges with relative ease. For simplicity, we will define *extended neighbourhood graphs* for directed cycles only. However, these graphs easily generalize to other graph families similarly as the neighbourhood graphs discussed in Section 6.3.

Definition 8.4. For two non-negative integer parameters $k \geq 3$ and $T \geq 1$, the *extended neighbourhood graph* $\mathcal{X}_{k,T}$ for directed cycles is a bipartite directed graph constructed as follows. The set of vertices is $V = S \cup K$ where

$$S = \{(x_0, x_1, \ldots, x_{2T}) : x_i, x_{i+1} \in [k] \text{ and } x_i \neq x_{i+1} \text{ where } 0 \leq i < 2T\}$$

is the set of *node neighbourhoods* and the set

$$K = \{(x_1, x_2, \ldots, x_{2T}) : (x_0, x_1, \ldots, x_{2T}) \in S\}$$

consists of *edge neighbourhoods*. The edge set E of the graph is given by the conditions $(\mathbf{s}, \mathbf{a}) \in E \iff \mathbf{a} = (s_1, s_2, \ldots, s_{2T})$ and $(\mathbf{a}, \mathbf{s}) \in E \iff \mathbf{a} = (s_0, s_1, \ldots, s_{2T-1})$ where $\mathbf{s} \in S$ and $\mathbf{a} \in K$.

We consider the distributed maximal matching problem. We will construct a formula in propositional logic that is satisfiable if and only if there exists a T-time distributed algorithm that computes a maximal matching in any k-coloured cycle. The satisfiability of the formula can be determined with a SAT solver.

Let $\mathcal{X}_{k,T} = (S \cup K, E)$ be the neighbourhood graph. For each $a \in K$ and $s \in S$ let x_a and x_s be Boolean variables. A variable x_a denotes whether the corresponding edge has been selected by the algorithm into the matching, whereas x_s is true if the corresponding node has been matched.

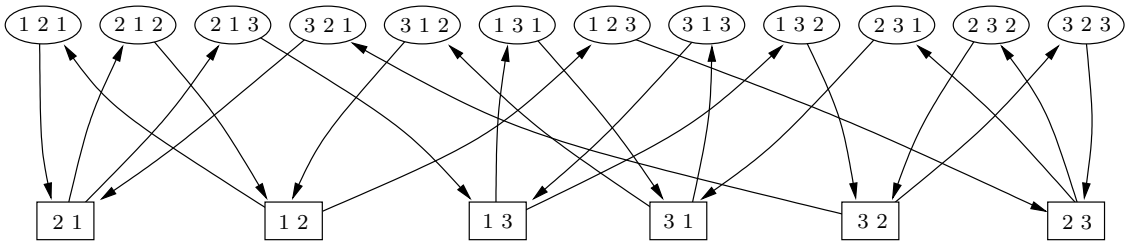


Figure 19: The extended neighbourhood graph $\mathcal{X}_{3,1}$ for cycles. The ellipse nodes represent the node neighbourhoods and the rectangular nodes are edge neighbourhoods.

We ensure that the following three properties hold in any communication graph: (i) if an edge has been selected, then none of the adjacent edges have been selected, (ii) if a node v has been matched, then some edge incident to v is in the matching, and (iii) if a node has not been matched, then it is adjacent to a matched node.

These conditions are captured by the formula $\psi(\mathcal{X}_{k,T}) = \psi_1 \wedge \psi_2 \wedge \psi_3$ where ψ_1, ψ_2, and ψ_3 correspond to the above constraints. Let

$$L = \{(a, b) \in K \times K : \operatorname{dist}(a, b) = 2\}$$

be the pairs of edge neighbourhoods with a common node neighbourhood. Now the formula ψ_1 is defined as

$$\psi_1 = \bigwedge_{(a,b)\in L} \big(x_a \to \neg x_b\big).$$

That is, edge neighbourhood a can be selected if no edge neighbourhood b which is "adjacent" to a is selected. This ensures that the selected edge set produces a matching. The case (ii) is covered by the formula

$$\psi_2 = \bigwedge_{s\in S} \Big(x_s \to \bigvee_{(s,a),(a,s)\in E} x_a\Big)$$

which states that a node neighbourhood can be matched only if some of its edge neighbourhoods is in the matching. Observe that together with the formula ψ_1, this implies that there will be exactly one such edge neighbourhood. Finally, let

$$U = \{(s, t) \in S \times S : (s, a), (a, t) \in E \text{ for some } a \in K\}$$

be the pairs of node neighbourhoods for nodes that may be adjacent in some communication graph. Now, the formula

$$\psi_3 = \bigwedge_{(s,t)\in U} x_s \vee x_t$$

ensures that the matching is maximal. If a node is not matched, then all of its neighbouring nodes must be matched.

Proposition 8.5. *Let $k, T \in \mathbb{N}^+$ such that $k \geq 3$ and $T \geq 1$. The following claims are equivalent:*

(i) There exists a deterministic distributed algorithm that finds a maximal matching in a directed k-coloured cycle in T communication rounds.

(ii) The formula $\psi(\mathcal{X}_{k,T})$ is satisfiable.

Proof. (i) $\Rightarrow$ (ii): Let $\mathcal{A}$ be an algorithm that computes a maximal matching in any k-coloured directed cycle in T rounds and $\mathcal{X}_{k,T} = (S \cup K, E)$ be the extended neighbourhood graph.

Simulate algorithm $\mathcal{A}$ for all node neighbourhoods s and t. Let v_s and v_t be the nodes in the communication graph corresponding to neighbourhoods s and t. If the algorithm selects an edge (v_s, v_t), then set x_s, x_t and x_a to true where $a \in K$ such that $(s, a) \in E$ and $(a, t) \in E$. That is, a is the common edge neighbourhood for s and t. Otherwise, do nothing.

To see that this assignment satisfies the formula $\psi(\mathcal{X}_{k,T})$, assume the opposite. If ψ_1 is false, then for some pair of edge neighbourhoods $(a, b) \in L$ both x_a and x_b are true: the algorithm $\mathcal{A}$ chose two incident edges which contradicts the correctness of $\mathcal{A}$. Neither ψ_2 can be false as we never set x_s true for any $s \in S$ without also setting the corresponding variable of some adjacent edge neighbourhood as true. Finally, if ψ_3 is false, then for some $(s, t) \in U$ both x_s and x_t are false. In this case, algorithm $\mathcal{A}$ would not produce a maximal matching in a graph where these two (unmatched) neighbourhoods are adjacent, and this would contradict the correctness of $\mathcal{A}$.

(ii) $\Rightarrow$ (i): Let τ be a truth assignment that satisfies $\psi(\mathcal{X}_{k,T})$. The algorithm first collects the radius-T neighbourhood of each node. Each node v checks the edge neighbourhoods $a_1, a_2, \ldots, a_{\deg(v)}$ for each adjacent edge in port i. If $\tau(x_{a_i}) = 1$ for some $i \in \{1, 2, \ldots, \deg(v)\}$, then v adds the edge in port i to the matching.

Let us show that the above algorithm produces a maximal matching in any k-coloured directed cycle. Let M be the set of edges selected by the algorithm. To see that M is a matching, suppose the opposite holds: $\{(u, v), (v, w)\} \subseteq M$ for some nodes u, v, and w in the communication graph. Let a and b be the edge neighbourhoods of edges (u, v) and (v, w), and s the node neighbourhood of v. Then $\tau(x_a) = \tau(x_b) = 1$. However, as there is a path (a, s, b) in $\mathcal{X}_{k,T}$, then $(a, b) \in L$ and ψ_1 is false since $x_a \to \neg x_b$ does not hold.

Finally, the maximality of matching M is shown in a similar manner. Assume that M is not maximal: there exists a matching M' such that $M \subsetneq M'$ is a proper subset.

Let $(u, v) \in M' \setminus M$ be an edge that could be added into the matching and a be the corresponding edge neighbourhood. Let s and t be the node neighbourhoods of u and v. Since u is not incident to an edge in the matching M and ψ_2 is satisfied by assumption, it follows that $\tau(x_s) = 0$. Similarly, since v is not incident to an edge in M, it also follows that $\tau(x_t) = 0$. But (s, a, t) is a path in $\mathcal{X}_{x,T}$ and hence ψ_3 cannot be satisfied since $x_s \vee x_t$ does not hold. □

Observe that while we shoved the result for k-coloured cycles only, the definition of the formula generalizes to other types of graphs as well. For example, given a neighbourhood graph for k-coloured d-regular graphs, the same method can be used to construct a formula that captures the existence of maximal matching algorithms in d-regular graphs. Furthermore, it seems that the neighbourhood graphs that also contain edge neighbourhoods can be used to characterise other distributed problems as well. However, the characterisation of other distributed graph problems is left for future work.

9 Conclusions

In the centralized setting it is convenient and often necessary to use asymptotic bounds on computational complexity, whereas in the distributed setting it is both possible and natural to report the exact complexity of a problem. The time complexity of a problem corresponds exactly to the size of the local neighbourhood each node must examine in order to compute the output.

In this work, it was shown that it is possible to study the concrete complexity of distributed graph problems using computational methods. In particular, we examined the distributed graph colouring problem in directed cycles and directed rooted trees. Computational search with SAT solvers revealed exact lower bounds and new fast colouring algorithms.

While the previously established analytical upper and lower bounds for distributed graph colouring in directed cycles are tight up to constant additive terms, analysing the chromatic numbers of neighbourhood graphs yielded new bounds. For all $k \geq 3$, the upper bound for reducing a k-colouring to a 3-colouring was improved from $\frac{1}{2}(\log^* k + 7)$ communication rounds to $\frac{1}{2}(\log^* k + 3)$ rounds in directed cycles, and in directed rooted trees, from $\log^* k + 6$ rounds to $\log^* k + 3$ rounds. Furthermore, the computational results indicate that for some values of k, reducing a k-colouring to

a 3-colouring requires at least $\frac{1}{2}(\log^* k + 1)$ rounds in directed cycles and $\log^* k + 1$ rounds in directed rooted trees.

Due to the slow growth of the log-star function, the additive term dominates the running time for most practical values of k. In some special cases, it is possible to gain significant speed-ups in colour reduction. As a concrete example, when $k \leq 2^{2048}$, it takes at most three rounds to properly 3-colour a k-coloured directed cycle in contrast to the six rounds required by the standard Cole–Vishkin style algorithms.

While the new bounds for 3-colouring directed cycles and directed rooted trees are almost tight, the exact value for the additive term $c \in [1, 3]$ remains open. A possible course of action is to target the colouring instances given in Section 8.4 and implement a program for exhaustive search with better symmetry-breaking features, such as isomorph rejection. Nevertheless, it seems that we are already quite close to understanding the exact complexity of graph colouring in directed cycles and trees.

As a prospective research topic, it would be interesting to study whether a similar approach can be successfully applied to also gain understanding of other distributed graph problems. For example, for the maximal matching problem in 2-coloured d-regular graphs, no matching lower and upper bounds are currently known. In 2-coloured bounded-degree graphs, a maximal matching can be computed in $\mathcal{O}(\Delta)$ communication rounds [HKP98], while computing a maximal matching is known to require at least $\Omega(\log \Delta)$ rounds [KMW10]. On the other hand, it seems that for d-regular graphs, no non-constant lower bounds are known. Thus, for various families of d-regular graphs, these bounds could be computationally derived by using, for example, the characterisation given in Section 8.5.

In this work the computer search was performed using SAT solvers. In addition to their good performance, one of the main advantages in the use of SAT solvers is the wide availability of different types of solvers and their ease of initial configuration. After encoding the problem instance once, it is easy to experiment with various solvers and fine-tune their parameters when necessary. This feature was particularly useful when some solvers could not solve the larger instances, while after some experimentation, other solvers managed to find solutions.

However, in addition to the SAT solver techniques, there exists a large body of work on various other combinatorial search algorithms and techniques. It may be useful to investigate and implement customised search algorithms that exploit the combinatorial properties of the more difficult and larger problem instances, such as the unsolved colouring instances or matching instances for high-degree graph families.

Moreover, exploring alternative and more succinct characterisations of distributed algorithms for graph problems is a possible avenue for future studies.

Acknowledgements

I am grateful to my thesis advisers Jukka Suomela and Petteri Kaski for invaluable discussions and comments. In addition, I wish to thank Veli Mäkinen for feedback and supervising the work, Matti Järvisalo for discussions on SAT solving, and the following people for various ideas and comments: Patrik Floréen, Janne Korhonen, Topi Musto, Laura Pesola, Jan–Mikael Rybicki, and Jara Uitto.

References

Aar07 Aaronson, S., Experimental complexity theory, June 2007. http://www.scottaaronson.com/blog/?p=252. Accessed 1st April, 2010.

ABLP89 Alon, N., Bar-Noy, A., Linial, N. and Peleg, D., On the complexity of radio communication. *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC, Seattle, WA, USA, May 1989)*, New York, NY, USA, 1989, ACM Press, pages 274–285.

ÅFP+09 Åstrand, M., Floréen, P., Polishchuk, V., Rybicki, J., Suomela, J. and Uitto, J., A local 2-approximation algorithm for the vertex cover problem. *Proc. 23rd International Symposium on Distributed Computing (DISC, Elche, Spain, September 2009)*, volume 5805 of *Lecture Notes in Computer Science*, Berlin, Germany, 2009, Springer, pages 191–205.

AH89 Appel, K. and Haken, W., *Every Planar Map Is Four Colorable*. American Mathematical Society, 1989.

Ang80 Angluin, D., Local and global properties in networks of processors. *Proc. 12th Annual ACM Symposium on Theory of Computing (STOC, Los Angeles, CA, USA, April 1980)*, New York, NY, USA, 1980, ACM Press, pages 82–93.

ÅS10 Åstrand, M. and Suomela, J., Fast distributed approximation algorithms for vertex cover and set cover in anonymous networks. *Proc. 22nd Annual*

ACM Symposium on Parallelism in Algorithms and Architectures (SPAA, Santorini, Greece, June 2010), New York, NY, USA, 2010, ACM Press, pages 294–302.

Awe85 Awerbuch, B., Complexity of network synchronization. *Journal of the ACM*, 32,4(1985), pages 804–823.

BBH+98 Bar-Noy, A., Bellare, M., Halldórsson, M., Shachnai, H. and Tamir, T., On chromatic sums and distributed resource allocation. *Information and Computation*, 140,2(1998), pages 183–202.

BE09 Barenboim, L. and Elkin, M., Distributed (Δ+1)-coloring in linear (in Δ) time. *Proc. 41st Annual ACM Symposium on Theory of Computing (STOC, Bethesda, MD, USA, May–June 2009)*, New York, NY, USA, 2009, ACM Press, pages 111–120.

BE10a Barenboim, L. and Elkin, M., Deterministic distributed vertex coloring polylogarithmic time. *Proc. 29th Annual ACM Symposium on Principles of Distributed Computing (PODC, Zurich, Switzerland, July 2010)*, New York, NY, USA, 2010, ACM Press, pages 410–419.

BE10b Barenboim, M. and Elkin, M., Distributed deterministic edge coloring using bounded neighborhood independence, October 2010. Manuscript, arXiv:1010.2454v1 [cs.DC]. To appear in PODC 2011.

BGS98 Bellare, M., Goldreich, O. and Sudan, M., Free bits, PCPs, and non-approximability – towards tight results. *SIAM Journal on Computing*, 27,3(1998), pages 804–915.

BH06 Björklund, A. and Husfeldt, T., Inclusion-exclusion algorithms for counting set partitions. *Proc. 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS, Berkeley, CA, USA, October 2006)*, Los Alamitos, CA, USA, 2006, IEEE Computer Society Press, pages 575–582.

BHKK10 Björklund, A., Husfeldt, T., Kaski, P. and Koivisto, M., Covering and packing in linear space. *Proc. 37th International Colloquium on Automata, Languages and Programming (ICALP, Bordeaux, France, July 2010)*, volume 6198 of *Lecture Notes in Computer Science*, Berlin, Germany, 2010, Springer, pages 727–737.

Bie10 Biere, A., Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT race 2010. Technical Report 10/1, Institute for Formal Models and Verifcation, Johannes Kepler University, August 2010. FMV Report Series.

Bro41 Brooks, R. L., On colouring the nodes of a network. *Mathematical Proceedings of the Cambridge Philosophical Society*, 37,2(1941), pages 194–197.

BV01 Boldi, P. and Vigna, S., An effective characterization of computability in anonymous networks. *Proc. 15th International Symposium on Distributed Computing (DISC, Lisbon, Portugal, October 2001)*, volume 2180 of *Lecture Notes in Computer Science*, Berlin, Germany, 2001, Springer, pages 33–47.

CAC+81 Chaitin, G. J., Auslander, M. A., Chandra, A. K., Cocke, J., Hopkins, M. E. and Markstein, P. W., Register allocation via coloring. *Computer Languages*, 6,1(1981), pages 47–57.

Cay79 Cayley, A., On the colouring of maps. *Proceedings of the Royal Geographical Society and Monthly Record of Geography*, 1,4(1879), pages 259–261.

CHW08 Czygrinow, A., Hańćkowiak, M. and Wawrzyniak, W., Fast distributed approximations in planar graphs. *Proc. 22nd International Symposium on Distributed Computing (DISC, Arcachon, France, September 2008)*, volume 5218 of *Lecture Notes in Computer Science*, Berlin, Germany, 2008, Springer, pages 78–92.

CLRS01 Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C., *Introduction to Algorithms*. The MIT Press, Cambridge, MA, USA, second edition, 2001.

CNS81 Chiba, N., Nishizeki, T. and Saito, N., A linear 5-coloring algorithm of planar graphs. *Journal of Algorithms*, 2,4(1981), pages 317–327.

Coo71 Cook, S. A., The complexity of theorem-proving procedures. *Proc. 3rd Annual ACM Symposium on Theory of Computing (STOC, Shaker Heights, OH, USA, May 1971)*, New York, NY, USA, 1971, ACM Press, pages 151–158.

CV86 Cole, R. and Vishkin, U., Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70,1(1986), pages 32–53.

Die10 Diestel, R., *Graph Theory*. Springer, Berlin, Germany, fourth edition, 2010.

Epp05 Eppstein, D., Quasiconvex programming. In *Combinatorial and Computational Geometry*, Goodman, J. E., Pach, J. and Welzl, E., editors, Cambridge University Press, 2005, chapter 16, pages 287–331.

Epp06 Eppstein, D., Quasiconvex analysis of multivariate recurrence equations for backtracking algorithms. *ACM Transactions on Algorithms*, 2,4(2006), pages 492–509.

Erd59 Erdős, P., Graph theory and probability. *Canadian Journal of Mathematics*, 11,1(1959), pages 34–38.

ES03 Eén, N. and Sörensson, N., An extensible SAT-solver. *Proc. 6th International Conference on Theory and Applications of Satisfiability Testing (SAT, Santa Margherita Ligure, Italy, May 2003)*, volume 2919 of *Lecture Notes in Computer Science*, Berlin, Germany, 2003, Springer, pages 502–518.

Est03 Estrada, G. G., A note on designing logical circuits using SAT. *Proc. 5th International Conference on Evolvable Systems (ICES, Trondheim, Norway, March 2003)*, volume 2606 of *Lecture Notes in Computer Science*, Berlin, Germany, 2003, Springer.

FGIP07 Fraigniaud, P., Gavoille, C., Ilcinkas, D. and Pelc, A., Distributed computing with advice: Information sensitivity of graph coloring. *Proc. 34th International Colloquium on Automata, Languages and Programming (ICALP, Wrocław, Poland, July 2007)*, volume 4596 of *Lecture Notes in Computer Science*, Berlin, Germany, 2007, Springer, pages 231–242.

FGK05 Fomin, F. V., Grandoni, F. and Kratsch, D., Some new techniques in design and analysis of exact (exponential) algorithms. *Bulletin of the EATCS*, 87(2005), pages 47–77.

FGK09 Fomin, F. V., Grandoni, F. and Kratsch, D., A measure & conquer approach for the analysis of exact algorithms. *Journal of the ACM*, 56,5(2009), pages 25:1–25:32.

FK98 Feige, U. and Kilian, J., Zero knowledge and the chromatic number. *Journal of Computer and System Sciences*, 57,2(1998), pages 187–199.

FK06 Fedin, S. S. and Kulikov, A. S., Automated proofs of upper bounds on the running time of splitting algorithms. *Journal of Mathematical Sciences*, 134,5(2006), pages 2328–2391.

FK10 Fomin, F. V. and Kratsch, D., *Exact Exponential Algorithms*. Springer, Berlin, Germany, 2010.

FRRS09 Fellows, M. R., Rosamund, F. A., Rotics, U. and Szeider, S., Clique-width is NP-complete. *SIAM Journal on Discrete Mathematics*, 23,2(2009), pages 909–939.

GB65 Golomb, S. W. and Baumert, L. D., Backtrack programming. *Journal of the ACM*, 12,4(1965), pages 516–524.

GGHN04 Gramm, J., Guo, J., Hüffner, F. and Niedermeier, R., Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica*, 39,4(2004), pages 321–347.

GJ76 Garey, M. R. and Johnson, D. S., The complexity of near-optimal graph coloring. *Journal of the ACM*, 23,1(1976), pages 43–49.

GJ79 Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, NY, USA, 1979.

GJM06 Gent, I. P., Jefferson, C. and Miguel, I., MINION: A fast, scalable, constraint solver. *Proc. 17th European Conference on Artificial Intelligence (ECAI, Riva del Garda, Italy, August–September 2006)*, Amsterdam, Netherlands, 2006, IOS Press, pages 98–102.

GJS74 Garey, M. R., Johnson, D. S. and Stockmeyer, L. J., Some simplified NP-complete problems. *Proc. 6th Annual ACM Symposium on Theory of Computing (STOC, Seattle, WA, USA, May 1974)*, New York, NY, USA, 1974, ACM Press, pages 47–63.

GKNS07 Gebser, M., Kaufmann, B., Neumann, A. and Schaub, T., clasp: A conflict-driven answer set solver. *Proc. 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR, Tempe, AZ, USA, May 2007)*, volume 4483 of *Lecture Notes in Computer Science*, Berlin, Germany, 2007, Springer, pages 260–265.

GLS93 Grötschel, M., Lovász, L. and Schrijver, A., *Geometric Algorithms and Combinatorial Optimization*. Springer, Berlin, Germany, second edition, 1993.

Gon08 Gonthier, G., Formal proof – the four-color theorem. *Notices of the AMS*, 55,11(2008), pages 1382–1393.

GP87 Goldberg, A. V. and Plotkin, S. A., Parallel $(\Delta+1)$-coloring of constant-degree graphs. *Information Processing Letters*, 25,4(1987), pages 241–245.

GPS88 Goldberg, A. V., Plotkin, S. A. and Shannon, G. E., Parallel symmetry-breaking in sparse graphs. *SIAM Journal on Discrete Mathematics*, 1,4(1988), pages 434–446.

GRS80 Graham, R. L., Rothschild, B. L. and Spencer, J. H., *Ramsey Theory*. John Wiley & Sons, New York, NY, USA, 1980.

GS02 Gomes, C. P. and Shmoys, D., Completing quasigroups or Latin squares: A structured graph coloring problem. *Proc. of the Computational Symposium on Graph Coloring and its Generalization*, Ithaca, NY, USA, 2002, pages 22–39.

Hel63 Hellerman, L., A catalog of three-variable or-invert and and-invert logical circuits. *IEEE Transactions on Electronic Computers*, 12,3(1963), pages 198–223.

HJS09 Hamadi, Y., Jabbour, S. and Sais, L., ManySAT: A parallel SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 6,1(2009), pages 245–2262.

HK65 Hall, M. and Knuth, D. E., Combinatorial analysis and computers. *The American Mathematical Monthly*, 78,2(1965), pages 21–28.

HKP98 Hańćkowiak, M., Karoński, M. and Panconesi, A., On the distributed complexity of computing maximal matchings. *Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA, San Francisco, CA, USA, January 1998)*, Philadelphia, PA, USA, 1998, Society for Industrial and Applied Mathematics, pages 219–225.

Hol81 Holyer, I., The NP-completeness of edge-colouring. *SIAM Journal on Computing*, 10,4(1981), pages 718–720.

HSW56 Hall, M., Swift, J. D. and Walker, R. J., Uniqueness of the projective plane of order eight. *Mathematical Tables and Other aids to Computation*, 10,56(1956), pages 186–194.

JK10 Junttila, T. and Kaski, P., Exact cover via satisfiability: An empirical study. *Proc. 16th International Conference on Principles and Practice of Constraint Programming (CP, St. Andrews, Scotland, September 2010)*, volume 6308 of *Lecture Notes in Computer Science*, Berlin, Germany, 2010, Springer, pages 297–304.

Kar72 Karp, R. M., Reducibility among combinatorial problems. *Complexity of Computer Computations*, Miller, R. E. and Thatcher, J. W., editors, New York, NY, USA, 1972, Plenum Press, pages 85–103.

KKRR93 Kamath, A. P., Karmarkar, N. K., Ramakrishnan, K. G. and Resende, M. G. C., An interior point approach to Boolean vector function synthesis. *Proc. 36th Midwest Symposium on Circuits and Systems (MWSCAS, Detroit, MI, USA, August 1993)*, Piscataway, NJ, USA, 1993, IEEE, pages 185–189.

KKY09 Kojevnikov, A., Kulikov, A. S. and Yaroslavtsev, G., Finding efficient circuits using SAT-solvers. *Proc. 12th International Conference on Theory and Applications of Satisfiability Testing (SAT, Swansea, UK, June–July 2009)*, volume 5584 of *Lecture Notes in Computer Science*, Berlin, Germany, 2009, Springer.

KMW10 Kuhn, F., Moscibroda, T. and Wattenhofer, R., Local computation: Lower and upper bounds, 2010. Manuscript, arXiv:1011.5470 [cs.DC].

Knu11 Knuth, D. E., *The Art of Computer Programming*, volume 4A. Pearson Education, Upper Saddle River, NJ, USA, 2011.

KÖ06 Kaski, P. and Östergård, P. R. J., *Classification Algorithms for Codes and Designs*. Springer, Berlin, Germany, 2006.

Koi06 Koivisto, M., An $\mathcal{O}^*(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion–exclusion. *Proc. 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS, Berkeley, CA, USA, October 2006)*, Los Alamitos, CA, USA, 2006, pages 583–590.

KR03 Kobler, D. and Rotics, U., Edge dominating set and coloring on graphs with fixed clique-width. *Discrete Applied Mathematics*, 126,2–3(2003), pages 197–221.

Kuh09 Kuhn, F., Weak graph colorings: Distributed algorithms and applications. *Proc. 21st Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA, Calgary, Canada, August 2009)*, New York, NY, USA, 2009, ACM Press, pages 138–144.

KW06 Kuhn, F. and Wattenhofer, R., On the complexity of distributed graph coloring. *Proc. 25th Annual ACM Symposium on Principles of Distributed Computing (PODC, Denver, CO, USA, July 2006)*, New York, NY, USA, 2006, ACM Press, pages 7–15.

Lam91 Lam, C. W. H., The search for a finite projective plane of order 10. *American Mathematical Monthly*, 98,4(1991), pages 305–318.

Lin87 Linial, N., Distributive graph algorithms – global solutions from local data. *Proc. 28th Annual Symposium on Foundations of Computer Science (FOCS, Los Angeles, CA, USA, October 1987)*, Los Alamitos, CA, USA, 1987, IEEE Computer Society Press, pages 331–335.

Lin92 Linial, N., Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21,1(1992), pages 193–201.

LMS08 Lynce, I. and Marques-Silva, J., Haplotype inference with Boolean satisfiability. *International Journal on Artificial Intelligence Tools*, 17,2(2008), pages 355–387.

Lov68 Lovász, L., On chromatic numbers of finite set-systems. *Acta Mathematica Aeademiae Seientiarum Hungaricae*, 19,1–2(1968), pages 59–67.

LTS89 Lam, C. W. H., Thiel, L. and Swiercz, S., The non-existence of finite projective planes of order 10. *Canadian Journal of Mathematics*, 41,6(1989), pages 1117–1123.

LY94 Lund, C. and Yannakakis, M., On the hardness of approximating minimization problems. *Journal of the ACM*, 41,5(1994), pages 960–981.

MR08 Mulzer, W. and Rote, G., Minimum-weight triangulation is NP-hard. *Journal of the ACM*, 55,2(2008), pages 11:1–11:29.

Mus11 Musto, T., Knowledge of global bounds in local algorithms. Master's thesis, Department of Computer Science, University of Helsinki, 2011. To appear.

Myc57 Mycielski, J., Sur le coloriage des graphes. *Colloquim Mathematicum*, 3(1957), pages 161–162.

Nao91 Naor, M., A lower bound on probabilistic algorithms for distributive ring coloring. *SIAM Journal on Discrete Mathematics*, 4,3(1991), pages 409–412.

NS95 Naor, M. and Stockmeyer, L., What can be computed locally? *SIAM Journal on Computing*, 24,6(1995), pages 1259–1277.

Pel00 Peleg, D., *Distributed Computing – A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

PR01 Panconesi, A. and Rizzi, R., Some simple distributed algorithms for sparse networks. *Distributed Computing*, 14,2(2001), pages 97–100.

Pre04 Prestwich, S., Local search on SAT-encoded colouring problems. *Proc. 7th International Conference on Theory and Applications of Satisfiability Testing (SAT, Vancouver, BC, Canada, May 2004)*, volume 2919 of *Lecture Notes in Computer Science*. Springer, 2004, pages 105–119.

PS98 Papadimitriou, C. H. and Steiglitz, K., *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, Inc., Mineola, NY, USA, 1998.

PWZ96 Petkovšek, M., Wilf, H. and Zeillberger, D., *A=B*. A K Peters, Wellesley, MA, USA, 1996.

Rad09 Radziszowski, S. P., Small Ramsey numbers. *Electronic Journal of Combinatorics*, DS1(2009). 12th revision.

Ram30 Ramsey, F. P., On a problem of formal logic. *Proceedings of the London Mathematical Society*, 30(1930), pages 264–286.

Ram99 Ramanathan, S., A unified framework and algorithm for channel assignment in wireless networks. *Wireless Networks*, 5,2(1999), pages 81–94.

RSST96 Robertson, N., Sanders, D. P., Seymour, P. and Thomas, R., Efficiently four-colouring planar graphs. *Proc. 28th Annual ACM Symposium on Theory of Computing (STOC, Philadelphia, PA, USA, May 1996)*, New York, NY, USA, 1996, ACM Press, pages 571–575.

RSST97 Robertson, N., Sanders, D., Seymour, P. and Thomas, R., The four-colour theorem. *Journal of Combinatorial Theory, Series B*, 70,1(1997), pages 2–44.

Ryb11 Rybicki, J., Local colour reduction algorithms, http://cs.helsinki.fi/group/parac/colour-reduction/, April 2011. Online appendix.

Sch09 Schaafsma, B., MiniMerge: Symmetry-free learning in combinatorial problems. Master's thesis, Delft University of Technology, 2009.

SHvM09 Schaafsma, B., Heule, M. J. and van Maaren, H., Dynamic symmetry breaking by simulating Zykov contraction. *Proc. 12th International Conference on Theory and Applications of Satisfiability Testing (SAT, Swansea, UK, June–July 2009)*, volume 5584 of *Lecture Notes In Computer Science*, Berlin, Germany, 2009, Springer, pages 223–236.

Sip06 Sipser, M., *Introduction to the Theory of Computation*. Thomson Course Technology, Boston, MA, USA, second edition, 2006.

Suo11 Suomela, J., Survey of local algorithms, http://www.iki.fi/jukka.suomela/local-survey, 2011. Manuscript submitted for publication.

SV93 Szegedy, M. and Vishwanathan, S., Locality based graph coloring. *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC, San Diego, CA, USA, May 1993)*. ACM Press, 1993, pages 201–207.

TSSW96 Trevisan, L., Sorkin, G. B., Sudan, M. and Williamson, D. P., Gadgets, approximation, and linear programming. *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS, Burlington, VT, USA, October 1996)*, Los Alamitos, CA, USA, 1996, IEEE Computer Society Press, pages 617–626.

Van08 Van Gelder, A., Another look at graph coloring via propositional satisfiability. *Discrete Applied Mathematics*, 156,2(2008), pages 230–243.

Vaz01 Vazirani, V. V., *Approximation Algorithms*. Springer, Berlin, Germany, 2001.

Wal00 Walsh, T., SAT v CSP. *Proc. 6th International Conference on Principles and Practice of Constraint Programming (CP, Singapore, September 2000)*, volume 1894 of *Lecture Notes in Computer Science*, Berlin, Germany, 2000, Springer, pages 441–456.

Weg87 Wegener, I., *The Complexity of Boolean Functions*. John Wiley & Sons, Chichester, UK, 1987.

Wil07 Williams, R., *Algorithms and Resource Requirements for Fundamental Problems*. Ph.D. thesis, Carnegie Mellon University, 2007.

Wil08 Williams, R., Applying practice to theory. *ACM SIGACT News*, 39,4(2008), pages 37–52.

Wil10 Williams, R., Alternation-trading proofs, linear programming, and lower bounds. *Proc. 27th International Symposium on Theoretical Aspects of Computer Science (STACS, Nany, France, March 2010)*, volume 5 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Dagstuhl, Wadern, Germany, 2010, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pages 669–680.