Graphs, New Models, and Complexity

a dissertation presented
by
Jeremy J. Kun
to
The Department of Mathematics, Statistics, and Computer Science

in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of
Mathematics

University of Illinois at Chicago
Chicago, IL
May 2016

Thesis advisor: Professor Lev Reyzin | Jeremy J. Kun

Graphs, New Models, and Complexity

Abstract

Contents

Listing of figures

To my cherished wife Erin, who embraces me quirks and all; to my parents Judy and Mihály, who raised me with adventure and imagination; and to my grandmother Erzsébet, who made sure I ate my vegetables.

Acknowledgments

0

Introduction

This dissertation contains selected work of the author over the course of his graduate studies, focusing on the study of computational complexity in new models. The models studied concern graphs, networks, and the relation between the global and distributed computation computation of some quantity.

In the first chapter we discuss *anti-coordination* games played on graphs. These are games in which players (nodes) are connected by edges, and the players are incentivized to choose strategies that differ from their neighbors. We study the price of anarchy, which measures the tradeoff between players acting independently and greedily versus a central planning authority. We introduce a directed graph generalization which allows one to model both anti-coordination and coordination incentives. We further prove that the complexity of computing strategies with certain properties (akin to being a certain kind of Nash equilibrium) is NP-hard.

In the second chapter we introduce a new model for measuring the complexity of a combinatorial decision problem called *resilience*. Loosely speaking, an instance of a problem is resilient if it is satisfiable and remains satisfiable under small adversarial manipulations. For graph coloring, this corresponds to a graph which is, say, 3-colorable and remains so even after an adversary adds an arbitrary edge to the graph. In general, we ask how resilient a problem must be in order to make finding solutions tractable. Surprisingly, for the previous example of graph 3-coloring, it remains NP-hard to find a 3-coloring. We further study the gradient between hardness and tractability for resilient coloring. We also completely characterize the complexity of resilient boolean satisfiability: it is either vacuous or NP-hard.

In the third chapter we turn our attention to *MapReduce*, a popular model of distributed computation which has novel constraints on communication and space. We first refine an existing theoretical model of MapReduce. Then we prove a general result on the ability of a two-round MapReduce protocol to capture all of sublogarithmic space Turing machines. Finally, we prove a connection between MapReduce, the exponential time hypothesis, and long-standing open conjectures about complexity hierarchies within simultaneous time/space-bounded copmlexity classes (TISP). A simplification of this result is that the exponential time hypothesis implies a hierarchy within linear-space TISP, which in turn implies a hierarchy within MapReduce for each of the parameters of interest.

The reader is assumed to be familiar with the basic concepts of computational complexity—Turing machines, the classes P, NP, TIME, SPACE, the concept of a complexity hierarchy, basic reductions, etc.—as well as basic concepts in graph theory and game theory.

1

Anti-Coordination Games and Stable Graph Colorings

In this chapter we study *anti-coordination games* played on graphs. In brief, this game involves a set of n players V^*, each of whom can choose an action from a finite set $\{1, \ldots, k\}$. The players are connected by directed or undirected edges E. To play the game, the players choose actions simultaneously, and each player i is rewarded for each neighbor j that chooses a different action from player i. The natural game-theoretic question is to study the equilibria of this game. In this chapter we first show that it suffices to study pure equilibria, which are equivalent

[*] In this chapter we use the terms 'players,' 'agents,' 'nodes,' and 'vertices' interchangeably.

to a certain kind of graph coloring that we call *stable k-colorings*. The directed version of the game allows one to encode both coordination and anti-coordination.

The primary question tackled in this chapter is to determine the computational complexity of finding stable colorings. The main results presented in this chapter are that it is NP-hard to find strictly stable colorings in undirected graphs, and that it is NP-hard to find stable colorings in directed graphs. We also provide a tight bound of $(k-1)/k$ on the price of anarchy of this game. Stable colorings are equivalent to a handful of other combinatorial notions, and they some open problems in these areas, most notably the complexity of the strictly unfriendly partition problem.

1.1 Introduction and background

Anti-coordination games form some of the basic payoff structures in game theory. Such games are ubiquitous; miners deciding which land to drill for resources, company employees trying to learn diverse skills, and airplanes selecting flight paths all need to mutually anti-coordinate their strategies in order to maximize their profits or even avoid catastrophe.

Two-player anti-coordination is simple and well understood. In its barest form, the players have two actions, and payoffs are symmetric for the players, paying off 1 if the players choose different actions and 0 otherwise. This game has two strict pure-strategy equilibria, paying off 1 to each player, as well as a non-strict mixed-strategy equilibrium paying off an expected 1/2 to each player.

In the real world, however, coordination and anti-coordination games are more complex than the simple two-player game. People, companies, and even countries play such multi-party games simultaneously with one another. One straightforward way to model this is with a graph, whose vertices correspond to agents and whose edges capture their pairwise interactions. A vertex then chooses one of k strategies, trying to anti-coordinate with all its neighbors simultaneously. The payoff of a vertex is the sum of the payoffs of its games with its neighbors – namely the number of neighbors with which it has successfully anti-coordinated. It is easy to see that this model naturally captures many applications. For example countries may choose commodities to produce, and their value will depend on how many trading partners do not produce that commodity.

In this chapter we focus on finding pure strategies in equilibrium, as well as their associated social welfare and price of anarchy, concepts we shall presently define. We look at both strict and non-strict pure strategy equilibria, as well as games on directed and undirected graphs. Directed graphs characterize the case where only one of the vertices is trying to anti-coordinate with another. The directed case turns out to not only generalize the symmetric undirected case, but also captures coordination in addition to anti-coordination.

These problems also have nice interpretations as certain natural graph coloring and partition problems, variants of which have been extensively studied. For instance, a pure strategy equilibrium in an undirected graph corresponds to what we call a stable k-coloring of the graph, in which no vertex can have fewer neighbors of any color different than its own. For $k = 2$ colors this is equivalent to the well-studied *unfriendly partition* or *co-satisfactory partition* problem. The strict equilibrium version of this problem (which corresponds to what we call a strictly stable k-coloring) generalizes the *strictly unfriendly partition problem*. We establish both the NP-hardness of the decision problem for strictly unfriendly partitions and NP-hardness for higher k.

1.1.1 Previous work

In an early work on what can be seen as a coloring game, Naor and Stockmeyer[58] define a *weak k-coloring* of a graph to be one in which each vertex has a differently colored neighbor. They give a locally distributed algorithm that, under certain conditions, weakly 2-colors a graph in constant time.

Then, in an influential experimental study of anti-coordination in networks, Kearns et al.[45] propose a true graph coloring game, in which each participant controlled the color of a vertex, with the goal of coloring a graph in a distributed fashion. The players receive a reward only when a proper coloring of the graph is found. The theoretical properties of this game are further studied by Chaudhuri et al.[16] who prove that in a graph of maximum degree d, if players have $d+2$ colors available they will w.h.p. converge to a proper coloring rapidly using a greedy local algorithm. Our work is also largely motivated by the work of Kearns et al., but for a somewhat relaxed version of proper coloring.

Bramoullé et al.[11] also study a general anti-coordination game played on networks. In their formulation, vertices can choose to form links, and the payoffs of two anti-coordinated strategies may not be identical. They go on to characterize the strict equilibria of such games, as well as the effect of network structure on the behavior of individual agents. We, on the other hand, consider an arbitrary number of strategies but with a simpler payoff structure.

The game we study is related to the MAX-k-CUT game, in which each player (vertex) chooses its place in a partition so as to maximize the number of neighbors in other partitions. Hoefer[36], Monnot & Gourvès[31], research Nash equlibria and coalitions in this context. Our Propositions 1 and 2 generalize known facts proved there, and we include them for completeness.

This paper also has a strong relationship to *unfriendly partitions* in graph theory. An unfriendly partition of a graph is one in which each vertex has at least as many neighbors in other partitions as in its own. This topic has been extensively studied, especially in the combinatorics community[2,12,65]. While locally finite graphs admit 2-unfriendly partitions, uncountable graphs may not[65].

Friendly (the natural counterpart) and unfriendly partitions are also studied under the names *max satisfactory* and *min co-satisfactory partitions* by Bazgan et al.[6], who focus on partitions of size greater than 2. They characterize the complexity of determining whether a graph has a k-friendly partition and asked about characterizing k-unfriendly partitions for $k > 2$. Our notion of stable colorings captures unfriendly partitions, and we also solve the $k > 2$ case.

A natural strengthening of the notion above yields *strictly unfriendly partitions*, defined by Shafique and Dutton [64]. A strictly unfriendly partition requires each vertex to have strictly more neighbors outside its partition than inside it. Shafique and Dutton characterize a weaker notion, called *alliance-free partition*, but leave characterizing strictly unfriendly partitions open. Our notion of strictly stable coloring captures strictly unfriendly partitions, giving some of the first results on this problem. Cao and Yang [14] also study a related problem originating from sociology, called the *matching pennies game*, where some vertices try to coordinate and others try to anti-coordinate. They prove that deciding whether such a game has a pure strategy equilibrium is NP-Hard. Our work on the directed case generalizes their notion (which they suggested for future work). Among our results we give a simpler proof of their hardness result for $k = 2$ and also tackle $k > 2$, settling one of their open questions.

There are a few related games on graphs that involve coloring, but they instead focus on finding good proper colorings. In [60] Panagopoulou and Spirakis define a coloring game in which the payoff for a vertex is either zero if it shares a color with a neighbor, and otherwise the number of vertices in the graph with which it shares a color. They prove pure Nash equilibria always exist and can be efficiently computed, and provide nice bounds on the number of colors used. Chatzigiannakis, et al. [15] extend this line of work by analyzing distributed algorithms for this game, and Escoffier, et al. [23] improve their bounds.

1.1.2 Results

We provide proofs of the following, the last two being our main results.

1. *For all $k \geq 2$, every undirected graph has a stable k-coloring, and such a coloring can be*

found in polynomial time.

Our notion of stable k-colorings is a strengthening of the notion of k-unfriendly partitions of Bazgan et al.[6], solving their open problem number 15.

2. *For undirected graphs, the price of anarchy for stable k-colorings is bounded by* $\frac{k}{k-1}$*, and this bound is tight.*

3. *In undirected graphs, for all* $k \geq 2$*, determining whether a graph has a strictly stable k-coloring is NP-hard.*

   For $k = 2$, this notion is equivalent to the notion that is defined by Shafique and Dutton[64], but left unsolved.

4. *For all* $k \geq 2$*, determining whether a directed graph has even a non-strictly stable k-coloring is NP-hard.*

   Because directed graphs also capture coordination, this solves two open problems of Cao and Yang[14], namely generalizing the coin matching game to more than two strategies and considering the directed case.

1.2 Preliminaries and definitions

1.2.1 Stable Colorings

For an unweighted undirected graph $G = (V, E)$, let $C = \{f \mid f : V \to \{1, \ldots, k\}\}$. We call a function $c \in C$ a **coloring**. We study the following anti-coordination game played on a graph $G = (V, E)$. In the game, all vertices simultaneously choose a color, which induces a coloring $c \in C$ of the graph. In a given coloring c, an agent v's **payoff**, $\mu_c(v)$, is the number of neighbors choosing colors different from v's, namely

$$\mu_c(v) := \sum_{\{v,w\}\in E} \mathbf{1}_{\{c(v)\neq c(w)\}}.$$

Note that in this game higher degree vertices have higher potential payoffs.

We also have a natural generalization to directed graphs. That is, if $G = (V, E)$ is a directed graph and c is a coloring of V, we can define the payoff $\mu_c(v)$ of a vertex $v \in V$ analogously as the sum over outgoing edges:

$$\mu_c(v) := \sum_{(v,w)\in E} \mathbf{1}_{\{c(v)\neq c(w)\}}$$

Here a directed edge from v to w is interpreted as "v cares about w." We can then define the social welfare and price of anarchy for directed graphs identically using this payoff function.

Given a graph G, we define the **social welfare** of a coloring c to be

$$W(G, c) := \sum_{v\in V} \mu_c(v).$$

We say a coloring c is **stable**, or in equilibrium, if no vertex can improve its payoff by changing its color from $c(v)$ to another color. We define Q to be the set of stable colorings.

We call a coloring function c **strictly stable**, or in strict equilibrium, if every vertex would decrease its payoff by changing its color from $c(v)$ to another color. If a coloring function is stable and at least one vertex can change its color without decreasing its payoff, then the coloring is **non-strict**.

We define the **price of anarchy** for a graph G to be

$$\mathrm{PoA}(G) := \frac{\max_{c'\in C} W(G, c')}{\min_{c\in Q} W(G, c)}.$$

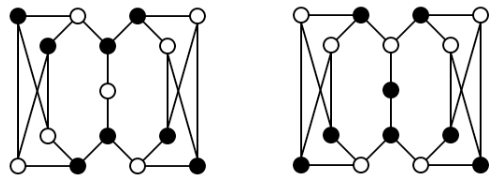


Figure 1.1: The strictly stable 2-coloring on the left attains a social welfare of 40 while the non-strictly stable coloring on the right attains 42, the maximum for this graph.

This concept was originally introduced by Koutsoupias and Papadimitriou in [49], where they consider the ratio of social payoffs in the best and worst-case Nash equilibria. Much work has since focused on the price of anarchy, e.g. [27,62].

1.2.2 Mixed and pure strategies

It is natural to consider both pure and mixed strategies for the players in our network anti-coordination game. A pure strategy solution does not in general exist for every 2 player game, while a mixed strategy solution will. However, in this coloring game not only will a pure strategy solution always exist, but for any mixed strategy solution there is a pure strategy equilibrium solution which achieves a social welfare at least as good, and where each player's payoff is identical with its expected payoff under the mixed strategy.

1.2.3 Strict and non-strict stability

It is worthwhile to note that a strictly stable coloring c need not provide the maximum social welfare. In fact, it is not difficult to construct a graph for which a strictly stable coloring exists yet the maximum social welfare is achieved by a non-strictly stable coloring, as shown in Figure 1.1.

1.3 Stable colorings

First we consider the problem of finding stable colorings in graphs. For the case $k = 2$, this is equivalent to the solved unfriendly partition problem. For this case our algorithm is equivalent to the well-studied local algorithm for MAX-CUT[21,57]. Our argument is a variant of a standard approximation algorithm for MAX-CUT, generalized to work with partitions of size $k \geq 2$.

Proposition 1. *For all $k \geq 2$, every finite graph $G = (V, E)$ admits a stable k-coloring. Moreover, a stable k-coloring can be found in polynomial time.*

Proof. Given a coloring c of a graph, define $\Phi(c)$ to be the number of properly-colored edges. It is clear that this function is bounded and that social welfare is $2\Phi(c)$. Moreover, the change in a vertex's utility by switching colors is exactly the change in Φ, realizing this as an exact potential game[56]. In a given coloring, we call a vertex v *unhappy* if v has more neighbors of its color than of some other color. We now run the following process: while any unhappy vertex exists, change its color to the color

$$c'(u) = \operatorname*{argmin}_{m \in \{1,\dots,k\}} \sum_{v \in N(u)} \mathbf{1}_{\{c(v)=m\}}. \tag{1.1}$$

As we only modify the colors of unhappy vertices, such an amendment to a coloring increases the value of Φ by at least 1. After at most $|E|$ such modifications, no vertex will be unhappy, which by definition means the coloring is stable. $\square$

We note that because, in the case of $k = 2$, maximizing the social welfare of a stable coloring is equivalent to finding the MAX-CUT of the same graph, which is known to be NP-hard[29], we cannot hope to find a global optimum for the potential function. However, we can ask about the price of anarchy, for which we obtain a tight bound. The following result also appears, using a different construction, in[36], but we include it herein for completeness.

Proposition 2. *The price of anarchy of the k-coloring anti-coordination game is at most* $\frac{k}{k-1}$*, and this bound is tight.*

Proof. By the pigeonhole principle, each vertex can always achieve a $\frac{k-1}{k}$ fraction of its maximum payoff by choosing its color according to Equation 1.1. Hence, if some vertex does not achieve this payoff then the coloring is not stable. This implies that the price of anarchy is at most $\frac{k}{k-1}$.

To see that this bound is tight take two copies of K_k on vertices $v_1, \ldots, v_k$ and $v_{k+1}, \ldots, v_{2k}$ respectively. Add an edge joining v_i with v_{i+k} for $i \in \{1, \ldots, k\}$. If each vertex v_i and v_{i+k} is given color i this gives a stable k-coloring of the graph, as each vertex has one neighbor of each of the k colors attaining the social welfare lower bound of $2(\frac{k-1}{k})|E|$. If, however, the vertices v_{i+k} take color $i+1$ for $i \in \{1, \ldots, k-1\}$ and v_{2k} takes color 1, the graph achieves the maximum social welfare of $2|E|$. This is illustrated for $k = 5$ in Figure 1.2. $\square$

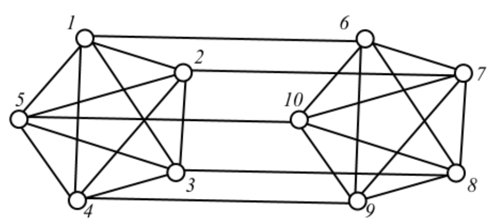


Figure 1.2: A graph achieving PoA of $\frac{5}{4}$, for k=5

1.4 Strictly Stable Colorings

In this section we show that the problem of finding a strictly stable equilibrium with any fixed number $k \geq 2$ of colors is NP-complete. We give NP-hardness reductions first for $k \geq 3$ and then for $k = 2$. The $k = 2$ case is equivalent to the strictly unfriendly 2-partition problem[64], whose complexity we settle.

Theorem 1. *For all $k \geq 2$, determining whether a graph has a strictly stable k-coloring is NP-complete.*

Proof. This problem is clearly in NP. We now analyze the hardness in two cases.

1) $k \geq 3$: For this case we reduce from classical k-coloring. Given a graph G, we produce a graph G' as follows.

Start with $G' = G$, and then for each edge $e = \{u, v\}$ in G add a copy H_e of K_{k-2} to G' and enough edges s.t. the induced subgraph of G' on $V(H_e) \cup \{u, v\}$ is the complete graph on k vertices. Figure 1.3 illustrates this construction.

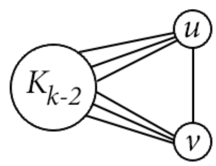


Figure 1.3: The gadget added for each edge in G.

Now supposing that G is k-colorable, we construct a strictly stable equilibrium in G' as follows. Fix any proper k-coloring ϕ of G. Color each vertex in G' which came from G (which is not in any H_e) using ϕ. For each edge $e = (u, v)$ we can trivially assign the remaining $k - 2$ colors among the vertices of H_e to put the corresponding copy of K_k in a strict equilibrium. Doing this for every such edge results in a strictly stable coloring. Indeed, this is a proper k-coloring of G' in which every vertex is adjacent to vertices of all other $k - 1$ colors.

Conversely, suppose G' has a strictly stable equilibrium with k colors. Then no edge e originally coming from G can be monochromatic. If it were, then there would be $k - 1$ remaining colors to assign among the remaining $k - 2$ vertices of H_e. No matter the choice, some color is unused and any vertex of H_e could change its color without penalty, contradicting that G' is in a strict equilibrium.

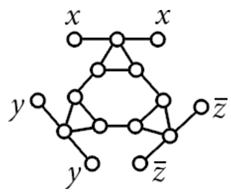


Figure 1.4: The clause gadget for $(x \vee y \vee \bar{z})$. Each literal corresponds to a pair of vertices, and a literal being satisfied corresponds to both vertices having the same color.

The only issue is if G originally has an isolated vertex. In this case, G' would have an isolated vertex, and hence will not have a strict equilibrium because the isolated vertex may switch colors arbitrarily without decreasing its payoff. In this case, augment the reduction to attach a copy of K_{k-1} to the isolated vertex, and the proof remains the same.

2) $k = 2$: We reduce from 3-SAT. Let $\phi = C_1 \wedge \cdots \wedge C_k$ be a boolean formula in 3-CNF form. We construct a graph G by piecing together gadgets as follows.

For each clause C_i construct an isomorphic copy of the graph shown in Figure 1.4. We call this the *clause gadget* for C_i. In Figure 1.4, we label certain vertices to show how the construction corresponds to a clause. We call the two vertices labeled by the same literal in a clause gadget a *literal gadget.* In particular, Figure 1.4 would correspond to the clause $(x \vee y \vee \bar{z})$, and a literal assumes a value of true when the literal gadget is monochromatic. Later in the proof we will force literals to be consistent across all clause gadgets, but presently we focus on the following key property of a clause gadget.

Lemma 1. *Any strictly stable 2-coloring of a clause gadget has a monochromatic literal gadget. Moreover, any coloring of the literal gadgets which includes a monochromatic literal extends to a strictly stable coloring of the clause gadget (excluding the literal gadgets).*

Proof. The parenthetical note will be resolved later by the high-degree of the vertices in the literal gadgets. Up to symmetries of the clause gadget (as a graph) and up to swapping colors, the proof of Lemma 1 is illustrated in Figure 1.5. The first five graphs show the cases where one or more literal gadgets are monochromatic, and the sixth shows how no strict equilibrium can exist otherwise. Using the labels in Figure 1.5, whatever the choice of color for the vertex v_1, its two uncolored neighbors must have the same color (or else v_1 is not in strict equilibrium). Call this color a. For v_2, v_3, use the same argument and call the corresponding colors b, c, respectively. Since there are only two colors, one pair of a, b, c must agree. WLOG suppose $a = b$. But then the two vertices labeled by a and b which are adjacent are not in strict equilibrium. □

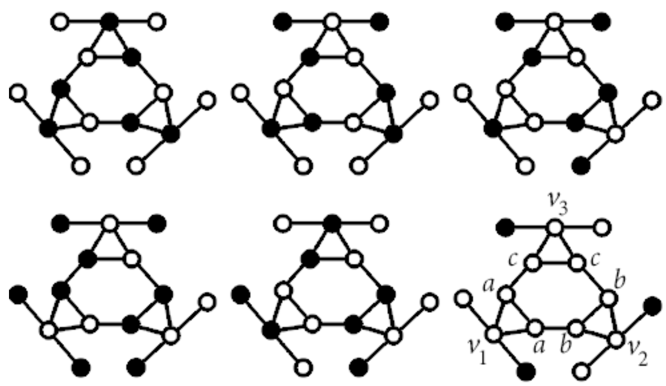


Figure 1.5: The first five figures show that a coloring with a monochromatic literal gadget can be extended to a strict equilibrium. The sixth (bottom right) shows that no strict equilibrium can exist if all the literals are not monochromatic.

Using Lemma 1, we complete the proof of the theorem. We must enforce that any two identical literal gadgets in different clause gadgets agree (they are both monochromatic or both not monochromatic), and that any negated literals disagree. We introduce two more simple gadgets for each purpose.

The first is for literals which must agree across two clause gadgets, and we call this the *literal persistence gadget*. It is shown in Figure 1.6. The choice of colors for the literals on one side determines the choice of colors on the other, provided the coloring is strictly stable. In particular, this follows from the central connecting vertex having degree 2. A nearly identical argument applies to the second gadget, which forces negated literals to assume opposite truth values. We call this the *literal negation gadget*, and it is shown in Figure 1.6. We do not connect all matching literals pairwise by such gadgets but rather choose one reference literal x' per variable and connect all literals for $x, \overline{x}$ to x' by the needed gadget.

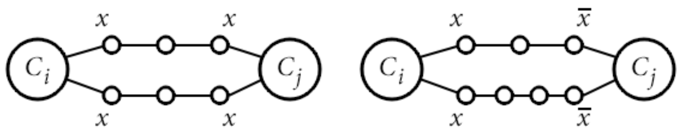


Figure 1.6: The literal persistence gadget (left) and literal negation gadget (right) connecting two clause gadgets C_i and C_j. The vertices labeled x on the left are part of the clause gadget for C_i, and the vertices labeled x on the right are in the gadget for C_j.

The reduction is proved in a straightforward way. If ϕ is satisfiable, then monochromatically color all satisfied literal gadgets in G. We can extend this to a stable 2-coloring: all connection gadgets and unsatisfied literal gadgets are forced, and by Lemma 1 each clause gadget can be extended to an equilibrium. By attaching two additional single-degree vertices to each vertex in a literal gadget, we can ensure that the literal gadgets themselves are in strict equilibrium and this does not affect any of the forcing arguments in the rest of the construction.

Conversely, if G has a strictly stable 2-coloring, then each clause gadget has a monochromatic literal gadget which gives a satisfying assignment of ϕ. All of the gadgets have a constant number of vertices so the construction is polynomial in the size of ϕ. This completes the reduction and proves the theorem. □

1.5 Stable colorings in directed graphs

In this section we turn to directed graphs. The directed case clearly generalizes the undirected as each undirected edge can be replaced by two directed edges. Moreover, directed graphs can capture coordination. For two colors, if vertex u wants to coordinate with vertex v, then instead of adding an edge (u, v) we can add a proxy vertex u' and edges (u, u') and (u', v). To be in equilibrium, the proxy has no choice but to disagree with v, and so u will be more inclined to agree with v. For k colors we can achieve the same effect by adding an undirected copy of K_{k-1}, appropriately orienting the edges, and adding edges $(u, x), (x, v)$ for each $x \in K_{k-1}$. Hence, this model is quite general.

Unlike in the undirected graph case, a vertex updating its color according to Equation 1.1 does not necessarily improve the overall social welfare. In fact, we cannot guarantee that a pure strategy equilibrium even exists – e.g. a directed 3-cycle has no stable 2-coloring, a fact that we will use in this section.

We now turn to the problem of determining if a directed graph has an equilibrium with k colors and prove it is NP-hard. Indeed, for strictly stable colorings the answer is immediate by reduction from the undirected case. Interestingly enough, it is also NP-hard for non-strict k-colorings for any $k \geq 2$.

Theorem 2. *For all $k \geq 2$, determining whether a directed graph has a stable k-coloring is NP-complete.*

Proof. This problem is clearly in NP. We again separate the hardness analysis into two parts: $k = 2$ and $k \geq 3$.

1) $k = 2$: We reduce from the balanced unfriendly partition problem. A balanced 2-partition of an undirected graph is called unfriendly if each vertex has at least as many neighbors outside its part as within. Bazgan et al. proved that the decision problem for balanced unfriendly partitions is NP-complete[6]. Given an undirected graph G as an instance of balanced unfriendly partition, we construct a directed graph G' as follows.

Start by giving G' the same vertex set as G, and replace each undirected edge of G with a pair of directed edges in G'. Add two vertices u, v to G', each with edges to the other and to all other vertices in G'. Add an additional vertex w with an edge (w, v), and connect one vertex of a directed 3-cycle to u and to w, as shown in Figure 1.7.

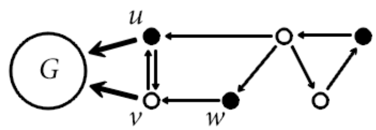


Figure 1.7: The construction from balanced unfriendly partition to directed stable 2-coloring. Here u and v "stabilize" the 3-cycle. A bold arrow denotes a complete incidence from the source to the target.

An unbalanced unfriendly partition of G corresponds to a two-coloring of G in which the colors occur equally often. Partially coloring G' in this way, we can achieve stability by coloring u, v opposite colors, coloring w the same color as u, and using this to stabilize the 3-cycle, as shown in Figure 1.7. Conversely, suppose G does not have a balanced unfriendly partition and fix a stable 2-coloring of G'. WLOG suppose G has an even number of vertices and suppose color 1 occurs more often among the vertices coming from G. Then u, v must both have color 2, and hence w has color 1. Since u, w have different colors, the 3-cycle will not be stable. This completes the reduction.

2) $k \geq 3$: We reduce from the case of $k = 2$. The idea is to augment the construction G' above by disallowing all but two colors to be used in the G' part. We call the larger construction G''.

We start with $G'' = G'$ add two new vertices x, y to G'' which are adjacent to each other. In a stable coloring, x and y will necessarily have different colors (in our construction they will not be the tail of any other edges). We call these colors 1 and 2, and will force them to be used in coloring G'. Specifically, let n be the number of vertices of G', and construct n^3 copies of K_{k-2}. For each vertex v in any copy of K_{k-2}, add the edges $(v, x), (v, y)$. Finally, add all edges (a, b) where $a \in G'$ and b comes from a copy of K_{k-2}. Figure 1.8 shows this construction.

Now in a stable coloring any vertex from a copy of K_{k-2} must use a different color than both x, y, and the vertex set of a copy of K_{k-2} must use all possible remaining $k - 2$ colors. By being

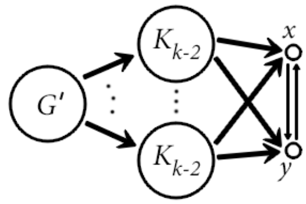


Figure 1.8: Reducing k colors to two colors. A bold arrow indicates complete incidence from the source subgraph to the target subgraph.

connected to n^3 copies of K_{k-2}, each $a \in G'$ will have exactly n^3 neighbors of each of the $k-2$ colors. Even if a were connected to all other vertices in G' and they all use color 1, it is still better to use color 1 than to use any of the colors in $\{3, \ldots, k\}$. The same holds for color 2, and hence we force the vertices of G' to use only colors 1 and 2. □

1.6 Discussion and open problems

In this chapter we defined new notions of graph coloring. Our results elucidated anti-coordination behavior, and solved some open problems in related areas.

Many interesting questions remain. For instance, one can consider alternative payoff functions. For players choosing colors i and j, the payoff $|i-j|$ is related to the *channel assignment problem*[70]. In the cases when the coloring problem is hard, as in our problem and the example above, we can find classes of graphs in which it is feasible, or study random graphs in which we conjecture colorings should be possible to find. Another variant is to study weighted graphs, perhaps with weights, as distances, satisfying a Euclidian metric. Finally, one could find an appropriate generalization of this game to hypergraphs and study equilibria in that setting.

2

Resilience and Resiliently Colorable Graphs

An important goal in studying NP-complete combinatorial problems is to find precise boundaries between tractability and NP-hardness. This is often done by adding constraints to the instances being considered until a polynomial time algorithm is found. For instance, while SAT is NP-hard, the restricted 2-SAT and XOR-SAT versions are decidable in polynomial time.

In this chapter we present a new angle for studying the boundary between NP-hardness and tractability. We informally define the resilience of a constraint-based combinatorial problem and we focus on the case of resilient graph colorability. Roughly speaking, a positive instance is resilient if it remains a positive instance up to the addition of a constraint. For example, an instance G of Hamiltonian circuit would be "r-resilient" if G has a Hamiltonian circuit, and G minus any r edges *still* has a Hamiltonian circuit. In the case of coloring, we say a graph G is r-resiliently k-colorable if G is k-colorable and will remain so even if any r edges are added. One would imagine that *finding* a k-coloring in a very resilient graph would be easy, as that instance is very "far" from being not colorable. And in general, one can pose the question: how resilient can instances be and have the search problem still remain hard?[*]

Most NP-hard problems have natural definitions of resilience. For instance, resilient positive instances for optimization problems over graphs can be defined as those that remain positive instances even up to the addition or removal of any edge. For satisfiability, we say a resilient instance is one where variables can be "fixed" and the formula remains satisfiable. In problems like set-cover, we could allow for the removal of a given number of sets. Indeed, this can be seen as a general notion of resilience for adding constraints in constraint satisfaction problems (CSPs), which have an extensive literature[52].[†]

Therefore we focus on a specific combinatorial problem, graph coloring. Resilience is defined up to the addition of edges, and we first show that this is an interesting notion: many famous, well studied graphs exhibit strong resilience properties. Then, perhaps surprisingly, we prove that 3-coloring a 1-resiliently 3-colorable graph is NP-hard – that is, it is hard to color a graph even when it is guaranteed to remain 3-colorable under the addition of any edge. Briefly, our reduction works by mapping positive instances of 3-SAT to 1-resiliently 3-colorable graphs and negative instances to graphs of chromatic number at least 4. An algorithm which can color 1-resiliently 3-colorable graphs can hence distinguish between the two. On the other hand, we

[*]We focus on the search versions of the problems because the decision version on resilient instances induces the trivial "yes" answer.

[†]However, a resilience definition for general CSPs is not immediate because the ability to add any constraint (e.g., the negation of an existing constraint) is too strong.

observe that 3-resiliently 3-colorable graphs have polynomial-time coloring algorithms (leaving the case of 3-coloring 2-resiliently 3-colorable graphs tantalizingly open). We also show that efficient algorithms exist for k-coloring $\binom{k}{2}$-resiliently k-colorable graphs for all k, and discuss the implications of our lower bounds.

This chapter is organized as follows. In the next two subsections we review the literature on other notions of resilience and on graph coloring. In Section 2.1 we characterize the resilience of boolean satisfiability, which is used in our main theorem on 1-resilient 3-coloring. In Section 2.2 we formally define the resilient graph coloring problem and present preliminary upper and lower bounds. In Section 2.3 we prove our main theorem, and in Section 2.4 we discuss open problems.

2.0.1 Related work on resilience

There are related concepts of resilience in the literature. Perhaps the closest in spirit is Bilu and Linial's notion of stability[9]. Their notion is restricted to problems over metric spaces; they argue that practical instances often exhibit some degree of stability, which can make the problem easier. Their results on clustering stable instances have seen considerable interest and have been substantially extended and improved[5,9,61]. Moreover, one can study TSP and other optimization problems over metrics under the Bilu-Linial assumption[55]. A related notion of stability by Ackerman and Ben-David[1] for clustering yields efficient algorithms when the data lies in Euclidian space.

Our notion of resilience, on the other hand, is most natural in the case when the optimization problem has natural constraints, which can be fixed or modified. Our primary goal is also different – we seek to more finely delineate the boundary between tractability and hardness in a systematic way across problems.

Property testing can also be viewed as involving resilience. Roughly speaking property testing algorithms distinguish between combinatorial structures that satisfy a property or are very far from satisfying it. These algorithms are typically given access to a small sample depending on a parameter ε alone. For graph property testing, as with resilience, the concept of being ε-far from having a property involves the addition or removal of an arbitrary set of at most $\varepsilon\binom{n}{2}$ edges from G. Our notion of resilience is different in that we consider adding or removing a constant number of constraints. More importantly, property testing is more concerned with query complexity than with computational hardness.

2.0.2 Previous work on coloring

As our main results are on graph colorability, we review the relevant past work. A graph G is k-colorable if there is an assignment of k distinct colors to the vertices of G so that no edge is monochromatic. Determining whether G is k-colorable is a classic an NP-hard problem [43]. Many attempts to simplify the problem, such as assuming planarity or bounded degree, still result in NP-hardness [18]. A large body of work surrounds positive and negative results for explicit families of graphs. The list of families that are polynomial-time colorable includes triangle-free planar graphs, perfect graphs and almost-perfect graphs, bounded tree- and clique-width graphs, quadtrees, and various families of graphs defined by the lack of an induced subgraph [13,22,35,48,50].

With little progress on coloring general graphs, research has naturally turned to approximation. In approximating the chromatic number of a general graph, the first results were of Garey and Johnson, giving a performance guarantee of $O(n/\log n)$ colors [40] and proving that it is NP-hard to approximate chromatic number to within a constant factor less than two [28]. Further work improved this bound by logarithmic factors [8,33]. In terms of lower bounds, Zuckerman [74] derandomized the PCP-based results of Håstad [34] to prove the best known approximability lower-bound to date, $O(n^{1-\varepsilon})$.

There has been much recent interest in coloring graphs which are already known to be colorable while minimizing the number of colors used. For a 3-colorable graph, Wigderson gave an algorithm using at most $O(n^{1/2})$ colors[72], which Blum improved to $\tilde{O}(n^{3/8})$[10]. A line of research improved this bound still further to $o(n^{1/5})$[44]. Despite the difficulties in improving the constant in the exponent, and as suggested by Arora[4], there is no evidence that coloring a 3-colorable graph with as few as $O(\log n)$ colors is hard.

On the other hand there are asymptotic and concrete lower bounds. Khot[47] proved that for sufficiently large k it is NP-hard to color a k-colorable graph with fewer than $k^{O(\log k)}$ colors; this was improved by Huang to $2^{\sqrt[3]{k}}$[37]. It is also known that for every constant h there exists a sufficiently large k such that coloring a k-colorable graph with hk colors is NP-hard[20]. In the non-asymptotic case, Khanna, Linial, and Safra[46] used the PCP theorem to prove it is NP-hard to 4-color a 3-colorable graph, and more generally to color a k colorable graph with at most $k + 2\lfloor k/3 \rfloor - 1$ colors. Guruswami and Khanna give an explicit reduction for $k = 3$[32]. Assuming a variant of Khot's 2-to-1 conjecture, Dinur et al. prove that distinguishing between chromatic number K and K' is hard for constants $3 \leq K < K'$[20]. This is the best conditional lower bound we give in Section 2.2.3, but it does not to our knowledge imply Theorem 4.

Without large strides in approximate graph coloring, we need a new avenue to approach the NP-hardness boundary. In this chapter we consider the coloring problem for a general family of graphs which we call *resiliently colorable*, in the sense that adding edges does not violate the given colorability assumption.

2.1 Resilient SAT

We begin by describing a resilient version of k-satisfiability, which is used in proving our main result for resilient coloring in Section 2.3.

Problem 1 (resilient k-SAT). *A boolean formula ϕ is* r-resilient *if it is satisfiable and remains satisfiable if any set of r variables are fixed. We call* r-resilient k-SAT *the problem of finding a satisfying assignment for an r-resiliently satisfiable k-CNF formula. Likewise, r-resilient CNF-SAT is for r-resilient formulas in general CNF form.*

The following lemma allows us to take problems that involve low (even zero) resilience and blow them up to have large resilience and large clause size.

Lemma 2 (blowing up). *For all $r \geq 0$, $s \geq 1$, and $k \geq 3$, r-resilient k-SAT reduces to* $[(r+1)s-1]$-*resilient* (sk)-*SAT in polynomial time.*

Proof. Let ϕ be an r-resilient k-SAT formula. For each i, let ϕ^i denote a copy of ϕ with a fresh set of variables. Construct $\psi = \bigvee_{i=1}^{s} \phi^i$. The formula ψ is clearly equivalent to ϕ, and by distributing the terms we can transform ψ into (sk)-CNF form in time $O(n^s)$. We claim that ψ is $[(r+1)s-1]$-resilient. If fewer than $(r+1)s$ variables are fixed, then by the pigeonhole principle one of the s sets of variables has at most r fixed variables. Suppose this is the set for ϕ^1. As ϕ is r-resilient, ϕ^1 is satisfiable and hence so is ψ. $\square$

As a consequence of the blowing up lemma for $r = 0, s = 2, k = 3$, 1-resilient 6-SAT is NP-hard (we reduce from this in our main coloring lower bound). Moreover, a slight modification of the proof shows that r-resilient CNF-SAT is NP-hard for all $r \geq 0$. The next lemma allows us to reduce in the other direction, shrinking down the resilience and clause sizes.

Lemma 3 (shrinking down). *Let $r \geq 1$, $k \geq 2$, and $q = \min(r, \lfloor k/2 \rfloor)$. Then r-resilient k-SAT reduces to q-resilient* $(\lceil \frac{k}{2} \rceil + 1)$-*SAT in polynomial time.*

Proof. For ease of notation, we prove the case where k is even. For a clause $C = \bigvee_{i=1}^{k} x_i$, denote by $C[: k/2]$ the sub-clause consisting of the first half of the literals of C, specifically $\bigvee_{i=1}^{k/2} x_i$. Similarly denote by $C[k/2 :]$ the second half of C. Now given a k-SAT formula $\phi = \bigwedge_{j=1}^{k} C_j$, we construct a $(\frac{k}{2}+1)$-SAT formula ψ by the following. For each j introduce a new variable z_j, and define

$$\psi = \bigwedge_{j=1}^{k} (C_j[: k/2] \vee z_j) \wedge (C_j[k/2 :] \vee \overline{z_j})$$

The formulas ϕ and ψ are logically equivalent, and we claim ψ is q-resilient. Indeed, if some of the original set of variables are fixed there is no problem, and each z_i which is fixed corresponds to a choice of whether the literal which will satisfy C_j comes from the first or the second half. Even stronger, we can arbitrarily *pick* another literal in the correct half and fix its variable so as to satisfy the clause. The r-resilience of ϕ guarantees the ability to do this for up to r of the z_i. But with the observation that there are no l-resilient l-SAT formulas, we cannot get $k/2+1$ resilience when $r > k/2$, giving the definition of q. □

Combining the blowing up and shrinking down lemmas, we get a tidy characterization: r-resilient k-SAT is either NP-hard or vacuously trivial.

Theorem 3. *For all $k \geq 3$, $0 \leq r < k$, r-resilient k-SAT is NP-hard.*

Proof. We note that increasing k or decreasing r (while leaving the other parameter fixed) cannot make r-resilient k-SAT easier, so it suffices to reduce from 3-SAT to $(k-1)$-resilient k-SAT for all $k \geq 3$. For any r we can blow up from 3-SAT to r-resilient $3(r+1)$-SAT by setting $s = r+1$ in the blowing up lemma. We want to iteratively apply the shrinking down lemma until the clause size is s. If we write $s_0 = 3s$ and $s_i = \lceil s_i/2 \rceil + 1$, we would need that for some m, $s_m = s$ and that for each $1 \leq j < m$, the inequality $\lfloor s_j/2 \rfloor \geq r = s-1$ holds.

Unfortunately this is not always true. For example, if $s = 10$ then $s_1 = 16$ and $16/2 < 9$, so we cannot continue. However, we can avoid this for sufficiently large r by artificially increasing k after blowing up. Indeed, we just need to find some $x \geq 0$ for which $a_1 = \left\lceil \frac{3s+x}{2} \right\rceil + 1 =$

$2(s-1)$. And we can pick $x = s-6 = r-5$, which works for all $r \geq 5$. For $r = 2,3,4$, we can check by hand that one can find an x that works.[‡] For $r = 2$ we can start from 2-resilient 9-SAT; for $r = 3$ we can start from 16-SAT; and for $r = 4$ we can start from 24-SAT. □

2.2 Resilient graph coloring and preliminary bounds

In contrast to satisfiability, resilient graph coloring has a more interesting hardness boundary, and it is not uncommon for graphs to have relatively high resilience. In this section we present some preliminary bounds.

2.2.1 Problem definition and remarks

Problem 2 (resilient coloring). *A graph G is called* r-resiliently k-colorable *if G remains k-colorable under the addition of any set of r new edges.*

This notion is not vacuously trivial. Indeed, Figure 2.1 provides the resilience properties of some classic graphs. Moreover, Table 2.1 provides a count of the resilience properties of all graphs on 6-8 vertices for a small number of colors. These were determined by exhaustive computer search.

There are a few interesting constructions to build intuition about resilient graphs. First, it is clear that every k-colorable graph is 1-resiliently $(k+1)$-colorable (just add one new color for the additional edge), but for all $k > 2$ there exist k-colorable graphs which are not 2-resiliently $(k+1)$-colorable. Simply remove two disjoint edges from the complete graph on $k+2$ vertices. A slight generalization of this argument provides examples of graphs which are $\lfloor (k+1)/2 \rfloor$-colorable but not $\lfloor (k+1)/2 \rfloor$-resiliently k-colorable for $k \geq 3$. On the other hand, every

[‡]The difference is that for $r \geq 5$ we can get what we need with only two iterations, but for smaller r we require three steps.

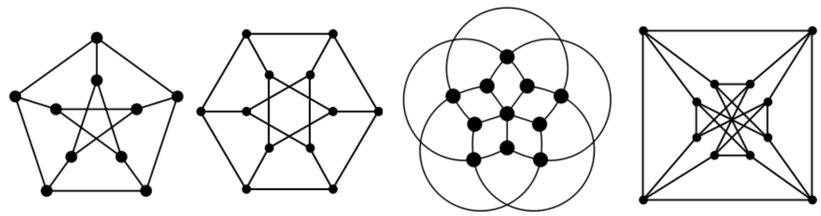


Figure 2.1: From left to right: the Petersen graph, 2-resiliently 3-colorable; the Dürer graph, 4-resiliently 4-colorable; the Grötzsch graph, 4-resiliently 4-colorably; and the Chvátal graph, 3-resiliently 4-colorable. These are all maximally resilient (no graph is more resilient than stated) and chromatic (no graph is colorably with fewer colors).

$\lfloor (k+1)/2 \rfloor$-colorable graph is $(\lfloor (k+1)/2 \rfloor - 1)$-resiliently k-colorable, since r-resiliently k-colorable graphs are $(r+m)$-resiliently $(k+m)$-colorable for all $m \geq 0$ (add one new color for each added edge).

One expects high resilience in a k-colorable graph to reduce the number of colors required to color it. While this may be true for super-linear resilience, there are easy examples of $(k-1)$-resiliently k-colorable graphs which are k-chromatic. For instance, add an isolated vertex to the complete graph on k vertices.

2.2.2 Observations

We are primarily interested in the complexity of coloring resilient graphs, and so we pose the question: for which values of k, r does the task of k-coloring an r-resiliently k-colorable graph admit an efficient algorithm? The following observations aid us in the classification of such pairs, which is displayed in Figure 2.2.

Observation 1. *An r-resiliently k-colorable graph is r'-resiliently k-colorable for any $r' \leq r$. Hence, if k-coloring is in P for r-resiliently k-colorable graphs, then it is for s-resiliently k-colorable graphs for all $s \geq r$. Conversely, if k-coloring is NP-hard for r-resiliently k-colorable graphs, then it is for s-resiliently k-colorable graphs for all $s \leq r$.*

r	1	2	3	4
k				
3	58.0	22.7	5.9	1.7
4	93.3	79.3	58.0	35.3
5	99.4	98.1	94.8	89.0
6	100.0	100.0	100.0	100.0

(a) $n = 6$ nodes

r	1	2	3	4
k				
3	38.1	8.2	1.2	0.3
4	86.7	62.6	35.0	14.9
5	98.7	95.6	88.5	76.2
6	99.9	99.7	99.2	98.3

(b) $n = 7$ nodes

r	1	2	3	4
k				
3	21.3	2.1	0.2	0.0
4	77.6	44.2	17.0	4.5

(c) $n = 8$ nodes

Table 2.1: The percentage of k-colorable graphs on $n = (6, 7, 8)$ nodes which are r-resilient. All values are rounded to the nearest tenth of a percent.

Hence, in Figure 2.2 if a cell is in P, so are all of the cells to its right; and if a cell is NP-hard, so are all of the cells to its left.

Observation 2. *If k-coloring is in P for r-resiliently k-colorable graphs, then k'-coloring r-resiliently k'-colorable graphs is in P for all $k' \le k$. Similarly, if k-coloring is in NP-hard for r-resiliently k-colorable graphs, then k'-coloring is NP-hard for r-resiliently k'-colorable graphs for all $k' \ge k$.*

Proof. If G is r-resiliently k-colorable, then we construct G' by adding a new vertex v with complete incidence to G. Then G' is r-resiliently $(k+1)$-colorable, and an algorithm to color G' can be used to color G. □

Observation 2 yields the rule that if a cell is in P, so are all of the cells above it; if a cell is NP-hard, so are the cells below it. More generally, we have the following observation which allows us to apply known bounds.

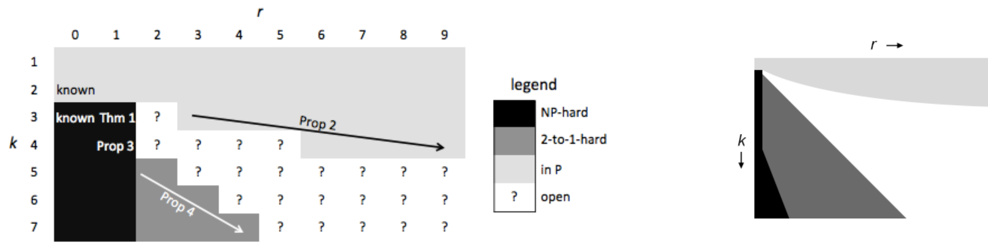


Figure 2.2: The classification of the complexity of k-coloring r-resiliently k-colorable graphs. Left: the explicit classification for small k, r. Right: a zoomed-out view of the same table, with the NP-hard (black) region added by Proposition 6.

Observation 3. *If it is NP-hard to $f(k)$-color a k-colorable graph, then it is NP-hard to $f(k)$-color an $(f(k) - k)$-resiliently $f(k)$-colorable graph.*

This observation is used in Propositions 4 and 5, and follows from the fact that an r-resiliently k-colorable graph is $(r + m)$-resiliently $(k + m)$-colorable for all $m \geq 0$ (here $r = 0, m = f(k) - k$).

2.2.3 Upper and lower bounds

In this section we provide a simple upper bound on the complexity of coloring resilient graphs, we apply known results to show that 4-coloring a 1-resiliently 4-colorable graph is NP-hard, and we give the conditional hardness of k-coloring $(k - 3)$-resiliently k-colorable graphs for all $k \geq 3$. This last result follows from the work of Dinur et al., and depends a variant of Khot's 2-to-1 conjecture[20]; a problem is called *2-to-1-hard* if it is NP-hard assuming this conjecture holds. Finally, applying the result of Huang[37], we give an asymptotic lower bound.

All our results on coloring are displayed in Figure 2.2. To explain Figure 2.2 more explicitly, Proposition 3 gives an upper bound for $r = \binom{k}{2}$, and Proposition 4 gives hardness of the cell $(4, 1)$ and its consequences. Proposition 5 provides the conditional lower bound, and Theorem 4 gives the hardness of the cell $(3, 1)$. Proposition 6 provides an NP-hardness result.

Proposition 3. *There is an efficient algorithm for k-coloring $\binom{k}{2}$-resiliently k-colorable graphs.*

Proof. If G is $\binom{k}{2}$-resiliently k-colorable, then no vertex may have degree $\geq k$. For if v is such a vertex, one may add complete incidence to any choice of k vertices in the neighborhood of v to get K_{k+1}. Finally, graphs with bounded degree $k-1$ are greedily k-colorable. □

Proposition 4. *4-coloring a 1-resiliently 4-colorable graph is NP-hard.*

Proof. It is known that 4-coloring a 3-colorable graph is NP-hard, so we may apply Observation 3. Every 3-colorable graph G is 1-resiliently 4-colorable, since if we are given a proper 3-coloring of G we may use the fourth color to properly color any new edge that is added. So an algorithm A which efficiently 4-colors 1-resiliently 4-colorable graphs can be used to 4-color a 3-colorable graph. □

Proposition 5. *For all $k \geq 3$, it is 2-to-1-hard to k-color a $(k-3)$-resiliently k-colorable graph.*

Proof. As with Proposition 4, we apply Observation 3 to the conditional fact that it is NP-hard to k-color a 3-colorable graph for $k > 3$. Such graphs are $(k-3)$-resiliently k-colorable. □

Proposition 6. *For sufficiently large k it is NP-hard to $2^{\sqrt[3]{k}}$-color an r-resiliently $2^{\sqrt[3]{k}}$-colorable graph for $r < 2^{\sqrt[3]{k}} - k$.*

Proposition 6 comes from applying Observation 3 to the lower bound of Huang[37]. The only unexplained cell of Figure 2.2 is (3,1), which we prove is NP-hard as our main theorem in the next section.

2.3 NP-hardness of 1-resilient 3-colorability

Theorem 4. *It is NP-hard to 3-color a 1-resiliently 3-colorable graph.*

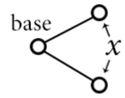


Figure 2.3: The gadget for a literal. The two single-degree vertices represent a single literal, and are interpreted as true if they have the same color. The base vertex is always colored gray. Note this gadget comes from Kun et al.[53].

Proof. We reduce 1-resilient 3-coloring from 1-resilient 6-SAT. This reduction comes in the form of a graph which is 3-colorable if and only if the 6-SAT instance is satisfiable, and 1-resiliently 3-colorable when the 6-SAT instance is 1-resiliently satisfiable. We use the colors white, black, and gray.

We first describe the gadgets involved and prove their consistency (that the 6-SAT instance is satisfiable if and only if the graph is 3-colorable), and then prove the construction is 1-resilient. Given a 6-CNF formula $\phi = C_1 \wedge \cdots \wedge C_m$ we construct a graph G as follows. Start with a base vertex b which we may assume w.l.o.g. is always colored gray. For each literal we construct a *literal gadget* consisting of two vertices both adjacent to b, as in Figure 2.3. As such, the vertices in a literal gadget may only assume the colors white and black. A variable is interpreted as true iff both vertices in the literal gadget have the same color. We will abbreviate this by saying a literal is *colored true* or *colored false*.

We connect two literal gadgets for $x, \overline{x}$ by a *negation gadget* in such a way that the gadget for x is colored true if and only if the gadget for $\overline{x}$ is colored false. The negation gadget is given in Figure 2.4. In the diagram, the vertices labeled 1 and 3 correspond to x, and those labeled 10 and 12 correspond to $\overline{x}$. We start by showing that no proper coloring can exist if both literal gadgets are colored true. If all four of these vertices are colored white or all four are black, then vertices 6 and 7 must also have this color, and so the coloring is not proper. If one pair is colored both white and the other both black, then vertices 13 and 14 must be gray, and the coloring is again not proper. Next, we show that no proper coloring can exist if both literal

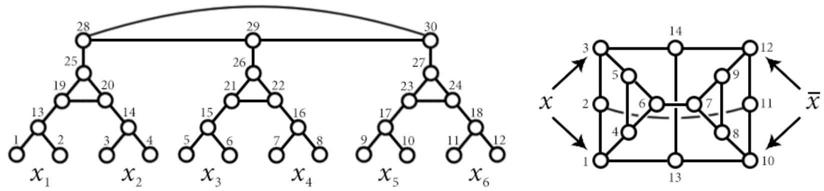
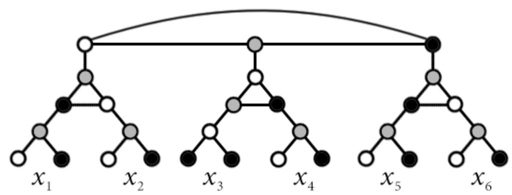


Figure 2.4: Left: the gadget for a clause. Right: the negation gadget ensuring two literals assume opposite truth values.

Figure 2.5: A valid coloring of the clause gadget when one variable (in this case x_3) is true.



gadgets are colored false. First, if vertices 1 and 10 are white and vertices 3 and 12 are black, then vertices 2 and 11 must be gray and the coloring is not proper. If instead vertices 1 and 12 are white and vertices 3 and 10 black, then again vertices 13 and 14 must be gray. This covers all possibilities up to symmetry. Moreover, whenever one literal is colored true and the other false, one can extend it to a proper 3-coloring of the whole gadget.

Now suppose we have a clause involving literals, w.l.o.g., $x_1, \ldots, x_6$. We construct the *clause gadget* shown in Figure 2.4, and claim that this gadget is 3-colorable iff at least one literal is colored true. Indeed, if the literals are all colored false, then the vertices 13 through 18 in the diagram must be colored gray, and then the vertices 25, 26, 27 must be gray. This causes the central triangle to use only white and black, and so it cannot be a proper coloring. On the other hand, if some literal is colored true, we claim we can extend to a proper coloring of the whole gadget. Suppose w.l.o.g. that the literal in question is x_1, and that vertices 1 and 2 both are black. Then Figure 2.5 shows how this extends to a proper coloring of the entire gadget regardless of the truth assignments of the other literals (we can always color their branches as if the literals were false).

It remains to show that G is 1-resiliently 3-colorable when ϕ is 1-resiliently satisfiable. This is because a new edge can, at worst, fix the truth assignment (perhaps indirectly) of at most one literal. Since the original formula ϕ is 1-resiliently satisfiable, G maintains 3-colorability. Additionally, the gadgets and the representation of truth were chosen so as to provide flexibility w.r.t. the chosen colors for each vertex, so many edges will have no effect on G's colorability.

First, one can verify that the gadgets themselves are 1-resiliently 3-colorable.[§] We break down the analysis into eight cases based on the endpoints of the added edge: within a single clause/negation/literal gadget, between two distinct clause/negation/literal gadgets, between clause and negation gadgets, and between negation and literal gadgets. We denote the added edge by $e = (v, w)$ and call it *good* if G is still 3-colorable after adding e.

Literal Gadgets. First, we argue that e is good if it lies within or across literal gadgets. Indeed, there is only one way to add an edge within a literal gadget, and this has the effect of setting the literal to false. If e lies across two gadgets then it has no effect: if c is a proper coloring of G without e, then after adding e either c is still a proper coloring or we can switch to a different representation of the truth value of v or w to make e properly colored (i.e. swap "white white" with "black black," or "white black" with "black white" and recolor appropriately).

Negation Gadgets. Next we argue that e is good if it involves a negation gadget. Let N be a negation gadget for the variable x. Indeed, by 1-resilience an edge within N is good; e only has a local effect within negation gadgets, and it may result in fixing the truth value of x. Now suppose e has only one vertex v in N. Figure 2.6 shows two ways to color N, which together with reflections along the horizontal axis of symmetry have the property that we may choose from at least two colors for any vertex we wish. That is, if we are willing to fix the truth value of x, then we may choose between one of two colors for v so that e is properly colored regardless of which color is adjacent to it.

[§]These graphs are small enough to admit verification by computer search.

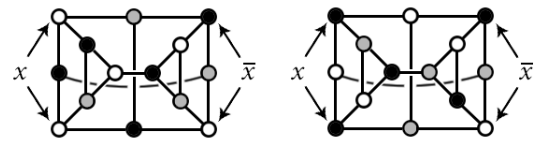


Figure 2.6: Two distinct ways to color a negation gadget without changing the truth values of the literals. Only the rightmost center vertex cannot be given a different color by a suitable switch between the two representations or a reflection of the graph across the horizontal axis of symmetry. If the new edge involves this vertex, we must fix the truth value appropriately.

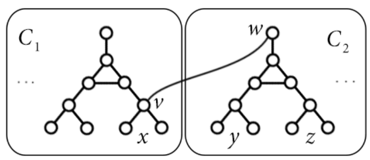


Figure 2.7: An example of an edge added between two clauses C_1, C_2.

Clause Gadgets. Suppose e lies within a clause gadget or between two clause gadgets. As with the negation gadget, it suffices to fix the truth value of one variable suitably so that one may choose either of two colors for one end of the new edge. Figure 2.7 provides a detailed illustration of one case. Here, we focus on two branches of two separate clause gadgets, and add the new edge $e = (v, w)$. The added edge has the following effect: if x is false, then neither y nor z may be used to satisfy C_2 (as w cannot be gray). This is no stronger than requiring that either x be true or y and z both be false, i.e., we add the clause $x \vee (\overline{y} \wedge \overline{z})$ to ϕ. This clause can be satisfied by fixing a single variable (x to true), and ϕ is 1-resilient, so we can still satisfy ϕ and 3-color G. The other cases are analogous.

This proves that G is 1-resilient when ϕ is, and finishes the proof. □

2.4 Discussion and open problems

The notion of resilience introduced in this chapter leaves many questions unanswered, both specific problems about graph coloring and more general exploration of resilience in other combinatorial problems and CSPs.

Regarding graph coloring, our chapter established the fact that 1-resilience doesn't affect the difficulty of graph coloring. However, the question of 2-resilience is open, as is establishing linear lower bounds without dependence on the 2-to-1 conjecture. There is also room for improvement in finding efficient algorithms for highly-resilient instances, closing the gap between NP-hardness and tractability.

On the general side, our framework applies to many NP-complete problems, including Hamiltonian circuit, set cover, 3D-matching, integer LP, and many others. Each presents its own boundary between NP-hardness and tractability, and there are undoubtedly interesting relationships across problems.

3

Computational Complexity and MapReduce

In this chapter we study the MapReduce[19] complexity class (MRC) defined by Karloff et al.[42], which is a formal complexity-theoretic model of MapReduce. We show that constant-round MRC computations can decide regular languages and simulate sublogarithmic space-bounded Turing machines. In addition, we prove hierarchy theorems for MRC under certain complexity-theoretic assumptions. These theorems show that sufficiently increasing the number of rounds or the amount of time per processor strictly increases the computational power of MRC. This work lays the foundation for further analysis relating MapReduce to established complexity classes. These results also hold for Valiant's BSP model[69] of parallel computation and the MPC model of Beame et al[7].

3.1 Introduction

MapReduce is a programming model originally developed to separate algorithm design from the engineering challenges of massively distributed computing. A programmer can separately implement a "map" function and a "reduce" function that satisfy certain constraints, and the underlying MapReduce technology handles all the communication, load balancing, fault tolerance, and scaling. MapReduce frameworks and their variants have been successfully deployed in industry by Google[19], Yahoo![67], and many others.

MapReduce offers a unique and novel model of parallel computation because it alternates parallel and sequential steps, and imposes sharp constraints on communication and random access to the data. This distinguishes MapReduce from classical theoretical models of parallel computation and this, along with its popoularity in industry, is a strong motivation to study the theoretical power of MapReduce. From a theoretical standpoint we ask how MapReduce relates to established complexity classes. From a practical standpoint we ask which problems can be efficiently modeled using MapReduce and which cannot.

In 2010 Karloff et al.[42] initiated a principled theoretical study of MapReduce, providing the definition of the complexity class MRC and comparing it with the classical PRAM models of parallel computing. But since this initial paper, almost all of the work on MapReduce has focused on algorithmic issues.

Complexity theory studies the classes of problems defined by resource bounds on different models of computation in which they are solved. A central goal of complexity theory is to understand the relationships between different models, i.e. to see if the problems solvable with bounded resources on one computational model can be solved with a related resource bound on a different model. In this chapter we prove a result that establishes a connection between MapReduce and space-bounded computation on classical Turing machines. Another traditional question asked by complexity theory is whether increasing the resource bound on a certain computational resource strictly increases the set of solvable problems. Such so-called hierarchy theorems exist for time and space on deterministic and non-deterministic Turing machines, among other settings. In this chapter we prove conditional hierarchy theorems for MapReduce rounds and time.

First we lay a more precise theoretical foundation for studying MapReduce computations (Section 3.3). In particular, we observe that Karloff et al.'s definitions are non-uniform, allowing the complexity class to contain undecidable languages. We reformulate the definition of[42] to make a uniform model and to more finely track the parameters involved (Section 3.3.2). In addition, we point out that the results in this chapter hold for other important models of parallel computations, including Valiant's Bulk-Synchronous Processing (BSP) model[69] and the Massively Parallel Communication (MPC) model of Beame et al[7]. (Section 3.3.3). We then prove two main theorems: SPACE($o(\log n)$) has constant-round MapReduce computations (Section 3.6) and, conditioned on a version of the Exponential Time Hypothesis, there are strict hierarchies within MRC. In particular, sufficiently increasing time or number of rounds increases the power of MRC (Section 3.7).

The sub-logarithmic space result is achieved by a direct simulation, using a two-round protocol that localizes state-to-state transitions to the section of the input being simulated, combining the sections in the second round. It is a major open problem whether undirected graph connectivity (a canonical logarithmic-space problem) has a constant-round MapReduce algorithm, and this result is the most general that can be proven without a breakthrough on graph connectivity. The hierarchy theorem involves proving a conditional time hierarchy within linear space achieved by a padding argument, along with proving a time-and-space upper and lower bounds on simulating MRC machines within P. This hierarchy theorem is the first of its kind. We conclude with a discussion and open questions raised by our work (Section 3.8).

3.2 Background and Previous Work

3.2.1 MapReduce

The MapReduce protocol can be roughly described as follows. The input data is given as a list of key-value pairs, and over a series of rounds two things happen per round: a "mapper" is applied to each key-value pair independently (in parallel), and then for each distinct key a "reducer" is applied to all corresponding values for a group of keys. The canonical example is counting word frequencies with a two-round MapReduce protocol. The inputs are (index, word) pairs, the first mapper maps $(k, v) \mapsto (v, k)$, and the first reducer computes the sum of the word frequencies for the given key. In the second round the mapper sends all data to a single processor via $(k, n_k) \mapsto (1, (k, n_k))$, and the second processor formats the output appropriately.

One of the primary challenges in MapReduce is data locality. MapReduce was designed for processing massive data sets, so MapReduce programs require that every reducer only has access to a substantially sublinear portion of the input, and the strict modularization prohibits reducers from communicating within a round. All communication happens indirectly through mappers, which are limited in power by the independence requirement. Finally, it's understood in practice that a critical quantity to optimize for is the number of rounds[42], so algorithms which cannot avoid a large number of rounds are considered inefficient and unsuitable for MapReduce.

There are a number of MapReduce-like models in the literature, including the MRC model of Karloff et al.[42], the "mud" algorithms of Feldman et al.[25], Valiant's BSP model[69], the MPC model of Beame et al.[7], and extensions or generalizations of these, e.g.[30]. The MRC class of Karloff et al. is the closest to existing MapReduce computations, and is also among the most restrictive in terms of how it handles communication and tracks the computational power of individual processors. In their influential paper[42], Karloff et al. display the algorithmic power of MRC, and prove that MapReduce algorithms can simulate CREW PRAMs which use subquadratic total memory and processors. It is worth noting that the work of Karloff et al. did not include comparisons to the standard (non-parallel) complexity classes, which is the aim of the present work.

Since[42], there has been extensive work in developing efficient algorithms in MapReduce-like frameworks. For example, Kumar et al.[51] analyze a sampling technique allowing them to translate sequential greedy algorithms into log-round MapReduce algorithms with a small loss of quality. Farahat et al.[24] investigate the potential for sparsifying distributed data using random projections. Kamara and Raykova[41] develop a homomorphic encryption scheme for MapReduce. And much work has been done on graph problems such as connectivity, matchings, sorting, and searching[30]. Chu et al.[17] demonstrate the potential to express any statistical-query learning algorithm in MapReduce. Finally, Sarma et al.[63] explore the relationship between communication costs and the degree to which a computation is parallel in one-round MapReduce problems. Many of these papers pose general upper and lower bounds on MapReduce computations as an open problem, and the results in this chapter are the first to do so with classical complexity classes.

The study of MapReduce has resulted in a wealth of new and novel algorithms, many of which run faster than their counterparts in classical PRAM models. As such, a more detailed study of the theoretical power of MapReduce is warranted. These results contribute to this by establishing a more precise definition of the MapReduce complexity class, proving that it contains sublogarithmic deterministic space, and showing the existence of certain kinds of hierarchies.

3.2.2 Complexity

From a complexity-theory viewpoint, MapReduce is unique in that it combines bounds on time, space and communication. Each of these bounds would be very weak on its own: the total time available to processors is polynomial; the total space and communication are slightly less than quadratic. In particular, even though arranging the communication between processors is one of the most difficult parts of designing MapReduce algorithms, classical results from communication complexity do not apply since the total communication available is more than linear. These innocent-looking bounds lead to serious restrictions when combined, as demonstrated by the fact that it is unknown whether constant-round MRC machines can decide graph connectivity (the best known result achieves a logarithmic number of rounds with high probability[42]), although it is solvable using only logarithmic space on a deterministic Turing machine.

We relate the MRC model to more classical complexity classes by studying simultaneous time-space bounds. $\mathrm{TISP}(T(n), S(n))$ are the problems that can be decided by a Turing machine which on inputs of length n takes at most $O(T(n))$ time and uses at most $O(S(n))$ space. Note that in general it is believed that $\mathrm{TISP}(T(n), S(n)) \neq \mathrm{TIME}(T(n)) \cap \mathrm{SPACE}(S(n))$. The complexity class TISP is studied in the context of time-space tradeoffs (see, for example,[26,73]). Unfortunately much less is known about TISP than about TIME or SPACE; for example there is no known time hierarchy theorem for fixed space. The existence of such a hierarchy is mentioned as an open problem in the monograph of Wagner and Wechsung[71].

To prove the results about TISP that imply the existence of a hierarchy in MRC, we use the Exponential Time Hypothesis (ETH) introduced by Impagliazzo, Paturi, and Zane[38,39], versions of which conjecture that 3-SAT does not have subexponential time algorithms. ETH and its siblings have been used to prove conditional lower bounds for specific hard problems like vertex cover, and for algorithms in the context of fixed parameter tractability (see, e.g., the survey of Lokshtanov, Marx and Saurabh [54]). The first open problem mentioned in [54] is to relate ETH to some other known complexity theoretic hypotheses, which we do with our TISP hierarchy.

We show in Lemma 5 that a weaker version of ETH directly implies a time-space trade-off, eg. that there are problems solvable in, say, n^6 time that cannot be solved in simultaneous quadratic time and linear space*. This 'weaker' ETH is not a well-known complexity theoretic hypothesis, but relative strengths of ETH, this weaker hypothesis, and the statement of the lemma seem to be unknown.

3.3 Models

In this section we introduce the model we will use in this chapter, a uniform version of Karloff's MapReduce Class (MRC), and contrast MRC to other models of parallel computation, such as Valiant's Bulk-Synchronous Parallel (BSP) model, for which these results also hold.

3.3.1 MapReduce and MRC

The central piece of data in MRC is the key-value pair, which we denote by a pair of strings $\langle k, v\rangle$, where k is the key and v is the value. An input to an MRC machine is a list of key-value pairs $\langle k_i, v_i\rangle_{i=1}^{N}$ with a total size of $n = \sum_{i=1}^{N} |k_i| + |v_i|$. The definitions in this subsection are adapted from [42].

Definition 1. *A* mapper *μ is a Turing machine*† *which accepts as input a single key-value pair*

*The actual constants depend on the ETH constant

†The definitions of [42] were for RAMs. However, because we wish to relate MapReduce to classical complexity classes, we reformulate the definitions here in terms of Turing machines.

$\langle k, v\rangle$ *and produces a list of key-value pairs* $\langle k_1', v_1'\rangle, \ldots, \langle k_s', v_s'\rangle$.

Definition 2. *A* reducer ρ *is a Turing machine which accepts as input a key* k *and a list of values* $\langle v_1, \ldots, v_m\rangle$*, and produces as output the same key and a new list of values* $\langle v_1', \ldots, v_M'\rangle$.

Definition 3. *For a decision problem, an input string* $x \in \{0,1\}^*$ *to an MRC machine is the list of pairs* $\langle i, x_i\rangle_{i=1}^n$ *describing the index and value of each bit. We will denote by* $\langle x\rangle$ *the list* $\langle i, x_i\rangle$.

An MRC machine operates in rounds. In each round, a set of mappers running in parallel first process all the key-value pairs. Then the pairs are partitioned (by a mechanism called "shuffle and sort" that is not considered part of the runtime of an MRC machine) so that each reducer only receives key-value pairs for a single key. Then the reducers process their data in parallel, and the results are merged to form the list of key-value pairs for the next round. More formally:

Definition 4. *An* R-round MRC machine *is an alternating list of mappers and reducers* $M = (\mu_1, \rho_1, \ldots, \mu_R, \rho_R)$. *The execution of the machine is as follows. For each* $r = 1, \ldots, R$:

1. *Let* U_{r-1} *be the list of key-value pairs generated by round* $r-1$ *(or the input pairs when* $r = 1$*). Apply* μ_r *to each key-value pair of* U_{r-1} *to get the multiset* $V_r = \bigcup_{\langle k,v\rangle \in U_{r-1}} \mu_r(k, v)$.
2. *Shuffle-and-sort groups the values by key. Call each of the pieces* $V_{k,r} = \{k, (v_{k,1}, \ldots, v_{k,s_k})\}$.
3. *Assign a different copy of reducer* ρ_r *to each* $V_{k,r}$ *(run in parallel) and set* $U_r = \bigcup_k \rho_r(V_{k,r})$.

The output is the final set of key-value pairs. For decision problems, we define M to accept $\langle x\rangle$ if in the final round $U_R = \emptyset$. Equivalently we may give each reducer a special accept state and say the machine accepts if at any time any reducer enters the accept state. We say M *decides* a language L if it accepts $\langle x\rangle$ if and only if $x \in L$.

The central caveat that makes MRC an interesting class is that the reducers have space constraints that are sublinear in the size of the input string. In other words, no sequential computation may happen that has random access to the entire input. Thinking of the reducers as processors, cooperation between reducers is obtained not by message passing or shared memory, but rather across rounds in which there is a global communication step.

In the MRC model we use in this chapter, we require that every mapper and reducer arise as separate runs of the same Turing machine M. The Turing machine $M(m, r, n, y)$ will accept as input the current round number r, a bit m denoting whether to run the r-th map or reduce function, the total number of rounds n, and the corresponding input y. Equivalently, we can imagine a list of mappers and reducers in each round $\mu_1, \rho_1, \mu_2, \rho_2, \dots$, where the descriptions of the μ_i, ρ_i are computable in polynomial time in $|i|$.

Definition 5 (Uniform Deterministic MRC). *A language L is said to be in* MRC$[f(n), g(n)]$ *if there is a constant $0 < c < 1$, an $O(n^c)$-space and $O(g(n))$-time Turing machine $M(m, r, n, y)$, and an $R = O(f(n))$, such that for all $x \in \{0, 1\}^n$, the following holds.*

1. *Letting $\mu_r = M(1, r, n, -)$, $\rho_r = M(0, r, n, -)$, the MRC machine $M_R = (\mu_1, \rho_1, \dots, \mu_R, \rho_R)$ accepts x if and only if $x \in L$.*
2. *Each μ_r outputs $O(n^c)$ distinct keys.*

This definition closely mirrors practical MapReduce computations: $f(n)$ represents the number of times global communication has to be performed, $g(n)$ represents the time each processor gets, and sublinear space bounds in terms of $n = |x|$ ensure that the size of the data on each processor is smaller than the full input.

Remark 1. *By $M(1, r, n, -)$, we mean that the tape of M is initialized by the string $\langle 1, r, n\rangle$. In particular, this prohibits an MRC algorithm from having $2^{\Omega(n)}$ rounds; the space constraints would prohibit it from storing the round number.*

Remark 2. *Note that a polynomial time Turing machine with sufficient time can trivially simulate a uniform MRC machine. All that is required is for the machine to perform the key grouping manually, and run the MRC machine as a subroutine. As such,* $\mathrm{MRC}[poly(n), poly(n)] \subseteq P$. *We give a more precise computation of the amount of overhead required in the proof of Lemma 6.*

Definition 6. *Define by* MRC^i *the union of uniform MRC classes*

$$\mathrm{MRC}^i = \bigcup_{k \in \mathbb{N}} \mathrm{MRC}[\log^i(n), n^k].$$

So in particular $\mathrm{MRC}^0 = \bigcup_{k \in \mathbb{N}} \mathrm{MRC}[1, n^k]$.

3.3.2 Nonuniformity

A complexity class is informally called uniform if the descriptions of the machines solving problems in it do not depend on the length of an input instance. Classical complexity classes defined by Turing machines with resource bounds, such as P, NP, and $\mathrm{SPACE}(\log(n))$, are uniform. On the other hand, circuit complexity classes are naturally nonuniform; a fixed Boolean circuit can only accept inputs of a single length. There is ambiguity about the uniformity of MRC as defined in [42]. Since we wish to relate the MRC model to classical complexity classes such as P and $\mathrm{SPACE}(\log(n))$, making sure that the model is uniform is crucial. Indeed, innocuous-seeming changes to the definitions above introduce nonuniformity (and in particular this is true of the original MRC definition in [42]). In Section 3.4 we show that the nonuniform MRC model defined in [42] allows MRC machines to solve undecidable problems in a logarithmic number of rounds, including the halting problem. We introduce the uniform version of MRC above to rule out such pathological behavior.

3.3.3 Other Models of Parallel Computation

Several other models of parallel computation have been introduced, including the BSP model of Valiant[69] and the MPC model of Beame et. al.[7]. The main difference between BSP and MapReduce is that in the BSP models the key-value pairs and the shuffling steps needed to redistribute them are replaced with point-to-point messages. Similarly to[42], in Valiant's paper[69] there is also ambiguity about the uniformity of the model. In this chapter, when we refer to BSP we mean a uniform deterministic version of the model. We give the exact definition in Setion 3.4.

Goodrich et al.[30] and Pace[59] showed that MapReduce computations can be simulated in the BSP model and vice versa, with only a constant blow-up in the computational resources needed. This implies that our theorems about MapReduce automatically apply to BSP.

Similarly, the MPC model uses point-to-point messages and Beame et. al.'s paper[7] does not discuss the uniformity of the model. The main distinguishing charateristic of the MPC model is that it introduces the number of processors p as an explicit paramter. Setting $p = O(n^c)$, our results will also hold in this model.

There are other variants of these models, including the model that Andoni et. al.[3] uses, which follows the MPC model but also introduces the additional constraint that total space used across each round must be no more than $O(n)$. It is straightforward to check that the proofs of our results never use more than $O(n)$ space, implying that our results hold even under this more restrictive model.

3.4 Nonuniform MRC

In this section we show that the original definition of MRC[42] allows MRC machines to decide undecidable languages. This definition required a polylogarithmic number of rounds, and also allowed completely different MapReduce machines for different input sizes. For simplicity's sake, we will allow a linear number of rounds, and use our notation $\mathrm{MRC}[f(n), g(n)]$ to denote an MRC machine that operates in $O(f(n))$ rounds and each processor gets $O(g(n))$ time per round. In particular, we show that nonuniform $\mathrm{MRC}[n, \sqrt{n}]$ accepts all unary languages, i.e. languages of the form $L \subseteq \{1^n \mid n \in \mathbb{N}\}$.

Lemma 4. *Let L be a unary language. Then L is in nonuniform* $\mathrm{MRC}[n, \sqrt{n}]$.

Proof. We define the mappers and reducers as follows. Let μ_1 distribute the input as contiguous blocks of $\sqrt{n}$ bits, ρ_1 compute the length of its input, μ_2 send the counts to a single processor, and ρ_2 add up the counts, i.e. find $n = |x|$ where x is the input. Now the input data is reduced to one key-value pair $\langle \star, n \rangle$. Then let ρ_i for $i \geq 3$ be the reducer that on input $\langle \star, i-3 \rangle$ accepts if and only if $1^{i-3} \in L$ and otherwise outputs the input. Let μ_i for $i \geq 3$ send the input to a single processor. Then ρ_{n+3} will accept iff x is in L. Note that ρ_1, ρ_2 take $O(\sqrt{n})$ time, and all other mappers and reducers take $O(1)$ time. All mappers and reducers are also in $\mathrm{SPACE}(\sqrt{n})$.

□

In particular, Lemma 4 implies that nonuniform $\mathrm{MRC}[n, \sqrt{n}]$ contains the unary version of the halting problem. A more careful analysis shows all unary languages are even in $\mathrm{MRC}[\log n, \sqrt{n}]$, by having ρ_{i+3} check 2^i strings for membership in L.

3.5 Uniform BSP

We define the BSP model of Valiant[69] similarly to MRC, where essentially key-value pairs are replaced with point-to-point messages.

A BSP machine with p processors is a list $(M_1, \ldots, M_p)$ of p Turing machines which on any input, output a list $((j_1, y_1), (j_2, y_2), \ldots, (j_m, y_m))$ of messages to be sent to other processors in the next round. Specifically, message y_k is sent to prcessor j_k. A BSP machine operates in rounds as follows. In the first round the input is partitioned into equal-sized pieces $x_{1,0}, \ldots, x_{p,0}$ and distributed arbitrarily to the processors. Then for rounds $r = 1, \ldots, R$,

1. Each processor i takes $x_{i,r}$ as input and computes some number s_i of messages $M_i(x_{i,r}) = \{(j_{i,k}, y_{i,k}) : k = 1, \ldots, s_i\}$.
2. Set $x_{i,r+1}$ to be the set of all messages sent to i (as with MRC's shuffle-and-sort, this is not considered part of processor i's runtime).

We say the machine *accepts* a string x if any machine accepts at any point before round R finishes. We now define uniform deterministic BSP analogously to MRC.

Definition 7 (Uniform Deterministic BSP). *A language L is said to be in* $\mathrm{BSP}[f(n), g(n)]$ *if there is a constant $0 < c < 1$, an $O(n^c)$-space and $O(g(n))$-time Turing machine $M(p, y)$, and an $R = O(f(n))$, such that for all $x \in \{0,1\}^n$, the following holds: letting $M_i = M(i, -)$, the BSP machine $M = (M_1, M_2, \ldots, M_{n^c})$ accepts x in R rounds if and only if $x \in L$.*

Remark 3. *As with MRC, we count the size and number of each message as part of the space bound of the machine generating/receiving the messages. Differing slightly from Valiant, we do not provide persistent memory for each processor. Instead we assume that on processor i, any memory cell not containing a message will form a message whose destination is i. This is without loss of generality since we are not concerned with the cost of sending individual messages.*

3.6 Space Complexity Classes in MRC^0

In this section we prove that small space classes are contained in constant-round MRC. Again, the results in this section also hold for other similar models of parallel computation, including the BSP model and the MPC model. First, we prove that the class REGULAR of regular languages is in MRC^0. It is well known that $\mathrm{SPACE}(O(1)) = \mathrm{REGULAR}$[66], and so this result can be viewed as a warm-up to the theorem that $\mathrm{SPACE}(o(\log n)) \subseteq \mathrm{MRC}^0$. Indeed, both proofs share the same flavor, which we sketch before proceeding to the details.

We wish to show that any given DFA can be simulated by an MRC^0 machine. The simulation works as follows: in the first round each parallel processor receives a contiguous portion of the input string and constructs a state transition function using the data of the globally known DFA. Though only the processor with the beginning of the string knows the true state of the machine during its portion of the input, all processors can still compute the *entire* table of state-to-state transitions for the given portion of input. In the second round, one processor collects the transition tables and chains together the computations, and this step requires only the first bit of input and the list of tables.

We can count up the space and time requirements to prove the following theorem.

Theorem 5. $\mathrm{REGULAR} \subsetneq \mathrm{MRC}^0$

Proof. Let L be a regular language and D a deterministic finite automaton recognizing L. Define the first mapper so that the j^{th} processor has the bits from $j\sqrt{n}$ to $(j+1)\sqrt{n}$. This means we have $K = O(\sqrt{n})$ processors in the first round. Because the description of D is independent of the size of the input string, we also assume each processor has access to the relevant set of states S and the transition function $t : S \times \{0,1\} \to S$.

We now define ρ_1. Fix a processor j and call its portion of the input y. The processor constructs a table T_j of size at most $|S|^2 = O(1)$ by simulating D on y starting from all possible states and recording the state at the end of the simulation. It then passes T_j and the first bit of y to the single processor in the second round.

In the second round the sole processor has K tables T_j and the first bit x_1 of the input string x (among others but these are ignored). Treating T_j as a function, this processor computes $q = T_K(\dots T_2(T_1(x_1)))$ and accepts if and only if q is an accepting state. This requires $O(\sqrt{n})$ space and time and proves containment. To show this is strict, inspect the prototypical problem of deciding whether the majority of bits in the input are 1's. □

Remark 4. *While the definition of* MRC^0 *inclues languages with time complexity* $O(n^k)$ *for all* $k \geq 0$*, our Theorem 5 is more efficient than the definition implies: we show that regular languages can be computed in* MRC^0 *in time and space* $O(\sqrt{n})$*, with the option of a tradeoff between time* n^ε *and space* $n^{1-\varepsilon}$.

One specific application of this result is that for any given regular expression, a two-round MapReduce computation can decide if a string matches that regular expression, even if the string is so long that any one machine can only store n^ε bits of it.

We now move on to prove $\mathrm{SPACE}(o(\log n)) \subseteq \mathrm{MRC}^0$. It is worth noting that this is a strictly stronger statement than Theorem 5. That is, $\mathrm{REGULAR} = \mathrm{SPACE}(O(1)) \subsetneq \mathrm{SPACE}(o(\log n))$. Several non-trivial examples of languages that witness the strictness of this containment are given in[68].

The proof is very similar to the proof of Theorem 5: Instead of the processors computing the entire table of state-to-state transitions of a DFA, the processors now compute the entire table of all transitions possible among the configurations of the work tape of a Turing machine that uses $o(\log n)$ space.

Theorem 6. $\mathrm{SPACE}(o(\log n)) \subseteq \mathrm{MRC}^0$.

Proof. Let L be a language in $\mathrm{SPACE}(o(\log n))$ and T a Turing machine recognizing L in polynomial time and $o(\log(n))$ space, with a read/write work tape W. Define the first mapper so that the j^{th} processor has the bits from $j\sqrt{n}$ to $(j+1)\sqrt{n}$. Let $\mathcal{C}$ be the set of all possible configurations of W and let S be the states of T. Since the size of S is independent of the input, we can assume that each processor has the transition function of T stored on it.

Now we define ρ_1 as follows: Each processor j constructs the graph of a function $T_j : \mathcal{C} \times \{L, R\} \times S \to \mathcal{C} \times \{L, R\} \times S$, which simulates T when the read head starts on either the left or right side of the jth $\sqrt{n}$ bits of the input and W is in some configuration from $\mathcal{C}$. It outputs whether the read head leaves the y portion of the read tape on the left side, the right side, or else accepts or rejects. To compute the graph of T_j, processor j simulates T using its transition function, which takes polynomial time.

Next we show that the graph of T_j can be stored on processor j by showing it can be stored in $O(\sqrt{n})$ space. Since W is by assumption size $o(\log n)$, each entry of the table is $o(\log n)$, so there are $2^{o(\log n)}$ possible configurations for the tape symbols. There are also $o(\log n)$ possible positions for the read/write head, and a constant number of states T could be in. Hence $|\mathcal{C}| = 2^{o(\log n)} o(\log n) = o(n^{1/3})$. Then processor j can store the graph of T_j as a table of size $O(n^{1/3})$.

The second map function μ_2 sends each T_j (there are $\sqrt{n}$ of them) to a single processor. Each is size $O(n^{1/3})$, and there are $\sqrt{n}$ of them, so a single processor can store all the tables. Using these tables, the final reduce function can now simulate T from starting state to either the accept or reject state by computing $q = T_k^*(\ldots T_2^*(T_1^*(\emptyset, L, \mathit{initial})))$ for some k, where $\emptyset$ denotes the initial configuration of T, *initial* is the initial state of T, and q is either in the accept or reject state. Note T_j^* is the modification of T_j such that if $T_j(x)$ outputs L, then $T_j^*(x)$ outputs R and vice versa. This is necessary because if the read head leaves the j^{th} $\sqrt{n}$ bits to the right, it enters the $j + 1^{\text{th}}$ $\sqrt{n}$ bits from the left, and vice versa. Finally, the reducer accepts if and only if q is in an accept state.

This algorithm successfully simulates T, which decides L, and only takes a constant number of rounds, proving containment. $\square$

3.7 Hierarchy Theorems

In this section we prove two main results (Theorems 7 and 8) about hierarchies within MRC relating to increases in time and rounds. They imply that allowing MRC machines sufficiently more time or rounds strictly increases the computing power of the machines. The first theorem states that for all α, β there are problems $L \notin \mathrm{MRC}[n^\alpha, n^\beta]$ which can be decided by *constant time* MRC machines when given enough extra rounds.

Theorem 7. *Suppose the ETH holds with constant c. Then for every* $\alpha, \beta \in \mathbb{N}$ *there exists a* $\gamma = O(\alpha + \beta)$ *such that*

$$\mathrm{MRC}[n^\gamma, 1] \not\subseteq \mathrm{MRC}[n^\alpha, n^\beta].$$

The second theorem is analogous for time, and says that there are problems $L \notin \mathrm{MRC}[n^\alpha, n^\beta]$ that can be decided by a *one round* MRC machine given enough extra time.

Theorem 8. *Suppose the ETH holds with constant c. Then for every* $\alpha, \beta \in \mathbb{N}$ *there exists a* $\gamma = O(\alpha + \beta)$ *such that*

$$\mathrm{MRC}[1, n^\gamma] \not\subseteq \mathrm{MRC}[n^\alpha, n^\beta].$$

As both of these theorems depend on the ETH, we first prove a complexity-theoretic lemma that uses the ETH to give a time-hierarchy within linear space TISP. Recall that TISP is the complexity class defined by simultaneous time and space bounds. The lemma can also be described as a time-space tradeoff. For some $b > a$ we prove the existence of a language that can be decided by a Turing machine with simultaneous $O(n^b)$ time and linear space, but cannot be decided by a Turing machine in time $O(n^a)$ even without any space restrictions. It is widely believed such languages exist for *exponential* time classes (for example, TQBF, the language of true quantified Boolean formulas, is a linear space language which is PSPACE-complete). We ask whether such tradeoffs can be extended to polynomial time classes, and this lemma shows that indeed this is the case.

Lemma 5. *Suppose that the ETH holds with constant c. Then for any positive integer a there exists a positive integer $b > a$ such that*

$$\mathrm{TIME}(n^a) \nsubseteq \mathrm{TISP}(n^b, n).$$

Proof. By the ETH, 3-SAT $\in \mathrm{TISP}(2^n, n) \setminus \mathrm{TIME}(2^{cn})$. Let $b := \lceil \frac{a}{c} \rceil + 2$, $\delta := \frac{1}{2}(\frac{1}{b} + \frac{c}{a})$. Pad 3-SAT with $2^{\delta n}$ zeros and call this language L, i.e. let $L := \{x0^{2^{\delta|x|}} \mid x \in \text{3-SAT}\}$. Let $N := n+2^{\delta n}$. Then $L \in \mathrm{TISP}(N^b, N)$ since $N^b > 2^n$. On the other hand, assume for contradiction that $L \in \mathrm{TIME}(N^a)$. Then, since $N^a < 2^{cn}$, it follows that 3-SAT $\in \mathrm{TIME}(2^{cn})$, contradicting the ETH. □

There are a few interesting complexity-theoretic remarks about the above proof. First, the starting language does not need to be 3-SAT, as the only assumption we needed was its hypothesized time lower bound. We could relax the assumption to the hypothesis that there exists a $c > 0$ such that TQBF, the PSPACE-complete language of true quantified Boolean formulas, requires 2^{cn} time, or further still to the following complexity hypothesis.

Conjecture 1. *There exist c', c satisfying $0 < c' < c < 1$ such that* $\mathrm{TISP}(2^n, 2^{c'n}) \setminus \mathrm{TIME}(2^{cn}) \neq \emptyset$.

Second, since $\mathrm{TISP}(n^a, n) \subseteq \mathrm{TIME}(n^a)$, this conditionally proves the existence of a hierarchy within $\mathrm{TISP}(\mathrm{poly}(n), n)$. We note that finding time hierarchies in fixed-space complexity classes was posed as an open question by[71], and so removing the hypothesis or replacing it with a weaker one is an interesting open problem.

Using this lemma we can prove Theorems 7 and 8. The proof of Theorem 7 depends on the following lemma.

Lemma 6. *For every $\alpha, \beta \in \mathbb{N}$ the following holds:*

$$\mathrm{TISP}(n^{\alpha}, n) \subseteq \mathrm{MRC}[n^{\alpha}, 1] \subseteq \mathrm{MRC}[n^{\alpha}, n^{\beta}] \subseteq \mathrm{TISP}(n^{\alpha+\beta+2}, n^2).$$

Proof. The first inequality follows from a simulation argument similar to the proof of Theorem 6. The MRC machine will simulate the $\mathrm{TISP}(n^{\alpha}, n)$ machine by making one step per round, with the tape (including the possible extra space needed on the work tape) distributed among the processors. The position of the tape is passed between the processors from round to round. It takes constant time to simulate one step of the $\mathrm{TISP}(n^{\alpha}, n)$ machine, thus in n^{α} rounds we can simulate all steps. Also, since the machine uses only linear space, the simulation can be done with $O(\sqrt{n})$ processors using $O(\sqrt{n})$ space each. The second inequality is trivial.

The third inequality is proven as follows. Let $T(n) = n^{\alpha+\beta+2}$. We first show that any language in $\mathrm{MRC}[n^{\alpha}, n^{\beta}]$ can be simulated in time $O(T(n))$, i.e. $\mathrm{MRC}[n^{\alpha}, n^{\beta}] \subseteq \mathrm{TIME}(T(n))$. The r-th round is simulated by applying μ_r to each key-value pair in sequence, shuffle-and-sorting the new key-value pairs, and then applying ρ_r to each appropriate group of key-value pairs sequentially. Indeed, $M(m, r, n, -)$ can be simulated naturally by keeping track of m and r, and adding n to the tape at the beginning of the simulation. Each application of μ_r takes $O(n^{\beta})$ time, for a total of $O(n^{\beta+1})$ time. Since each mapper outputs no more than $O(n^c)$ keys, and each mapper and reducer is in $\mathrm{SPACE}(O(n^c))$, there are no more than $O(n^2)$ keys to sort. Then shuffle-and-sorting takes $O(n^2 \log n)$ time, and the applications of ρ_r also take $O(n^{\beta+1})$ time. So a round takes $O(n^{\beta+1} + n^2 \log n)$ time. Note that keeping track of m,r, and n takes no more than the above time. So over $O(n^{\alpha})$ rounds, the simulation takes $O(n^{\alpha+\beta+1} + n^{\alpha+2} \log(n)) = O(T(n))$ time. □

Now we prove Theorem 7.

Proof. By Lemma 5, there is a language L in $\mathrm{TISP}(n^{\gamma}, n) \setminus \mathrm{TIME}(n^{\alpha+\beta+2})$ for some γ. By Lemma 6, $L \in \mathrm{MRC}[n^{\gamma}, 1]$. On the other hand, because $L \notin \mathrm{TIME}(n^{\alpha+\beta+2})$ and $\mathrm{MRC}[n^{\alpha}, n^{\beta}] \subseteq \mathrm{TIME}(n^{\alpha+\beta+2})$, we can conclude that $L \notin \mathrm{MRC}[n^{\alpha}, n^{\beta}]$. $\square$

Next, we prove Theorem 8 using a padding argument.

Proof. Let $T(n) = n^{\alpha+\beta+2}$ as in Lemma 6. By Lemma 5, there is a γ such that $\mathrm{TISP}(n^{\gamma}, n) \setminus \mathrm{TIME}(T(n^2))$ is nonempty. Let L be a language from this set. Pad L with n^2 zeros, and call this new language L', i.e. let $L' = \{x0^{|x|^2} \mid x \in L\}$. Let $N = n + n^2$. There is an $\mathrm{MRC}[1, N^{\gamma}]$ algorithm to decide L': the first mapper discards all the key-value pairs except those in the first n, and sends all remaining pairs to a single reducer. The space consumed by all pairs is $O(n) = O(\sqrt{N})$. This reducer decides L, which is possible since $L \in \mathrm{TISP}(n^{\gamma}, n)$. We now claim L' is not in $\mathrm{MRC}[N^{\alpha}, N^{\beta}]$. If it were, then L' would be in $\mathrm{TIME}(T(N))$. A Turing machine that decides L' in $T(N)$ time can be modified to decide L in $T(N)$ time: pad the input string with n^2 ones and use the decider for L'. This shows L is in $\mathrm{TIME}(T(n^2))$, a contradiction. $\square$

We conclude by noting explicitly that Theorems 7, 8 give proper hierarchies within MRC, and that proving certain stronger hierarchies imply the separation of L and P.

Corollary 1. *Suppose the ETH. For every α, β there exist $\mu > \alpha$ and $\nu > \beta$ such that*

$$\mathrm{MRC}[n^{\alpha}, n^{\beta}] \subsetneq \mathrm{MRC}[n^{\mu}, n^{\beta}]$$

and

$$\mathrm{MRC}[n^{\alpha}, n^{\beta}] \subsetneq \mathrm{MRC}[n^{\alpha}, n^{\nu}].$$

Proof. By Theorem 8, there is some $\mu > \alpha$ such that $\mathrm{MRC}[n^\mu, 1] \not\subseteq \mathrm{MRC}[n^\alpha, n^\beta]$. It is immediate that $\mathrm{MRC}[n^\alpha, n^\beta] \subseteq \mathrm{MRC}[n^\mu, n^\beta]$ and also that $\mathrm{MRC}[n^\mu, 1] \subseteq \mathrm{MRC}[n^\mu, n^\beta]$. So $\mathrm{MRC}[n^\alpha, n^\beta] \neq \mathrm{MRC}[n^\mu, n^\beta]$. The proof of the second claim is similar. □

Corollary 2. *If* $\mathrm{MRC}[\mathrm{poly}(n), 1] \subsetneq \mathrm{MRC}[\mathrm{poly}(n), \mathrm{poly}(n)]$, *then it follows that* $\mathrm{SPACE}(\log(n)) \neq \mathrm{P}$.

Proof.

$$\begin{aligned}\mathrm{SPACE}(\log(n)) &\subseteq \mathrm{TISP}(\mathrm{poly}(n), \log n) \subseteq \mathrm{TISP}(\mathrm{poly}(n), n) \subseteq \mathrm{MRC}[\mathrm{poly}(n), 1] \\ &\subseteq \mathrm{MRC}[\mathrm{poly}(n), \mathrm{poly}(n)] \subseteq \mathrm{P}.\end{aligned}$$

The first containment is well known, the third follows from Lemma 6, and the rest are trivial. □

Corollary 2 is interesting because if any of the containments in the proof are shown to be proper, then $\mathrm{SPACE}(\log(n)) \neq \mathrm{P}$. Moreover, if we provide MRC with a polynomial number of rounds, Corollary 2 says that determining whether time provides substantially more power is at least as hard as separating $\mathrm{SPACE}(\log(n))$ from P. On the other hand, it does not rule out the possibility that $\mathrm{MRC}[\mathrm{poly}(n), \mathrm{poly}(n)] = \mathrm{P}$, or even that $\mathrm{MRC}[\mathrm{poly}(n), 1] = \mathrm{P}$.

3.8 Discussion and Open Problems

In this chapter we established the first general connections between MapReduce and classical complexity classes, and showed the conditional existence of a hierarchy within MapReduce. The results in this chapter also apply to variants of MapReduce, most notably Valiant's BSP model.

This work suggests some natural open problems. How does MapReduce relate to other complexity classes, such as the circuit class uniform AC^0? Can one improve the bounds from Corollary 1 or remove the dependence on Hypothesis 1? Does Lemma 5 imply Hypothesis 1? Can one give explicit hierarchies for space or time alone, e.g. $\mathrm{MRC}[n^\alpha, \mathrm{poly}(n)] \subsetneq \mathrm{MRC}[n^\mu, \mathrm{poly}(n)]$?

We also ask whether $\mathrm{MRC}[\mathrm{poly}(n), \mathrm{poly}(n)] = \mathrm{P}$. In other words, if a problem has an efficient solution, does it have one with using data locality? A negative answer implies $\mathrm{SPACE}(\log(n)) \neq \mathrm{P}$ which is a major open problem in complexity theory, and a positive answer would likely provide new and valuable algorithmic insights. Finally, while we have focused on the relationship between rounds and time, there are also implicit parameters for the amount of (sublinear) space per processor, and the (sublinear) number of processors per round. A natural complexity question is to ask what the relationship between all four parameters are.

4

Conclusion

A

Source code listings

For completeness we provide the source code used in Chapter 2. There were three aspects for which a computer search helped:

1. To determine the resilience of famous graphs.
2. To provide the tables of resilience properties for small graphs, Table 2.1.
3. To double-check the resilience properties of the gadgets in Theorem 4.

Note that the computational search for the third item is not necessary, as the sketch given in the proof of Theorem 4 can be turned into a small set of cases to check by hand. However, by providing it here we hope to reinforce the certainty of the result.

The full source code and datasets used is also available on Github at

`https://github.com/j2kun/resilient-coloring-code`

Listing A.1: The sourcecode for computing resilience by brute force and heuristically-guided search.

```
import itertools
import random
import parameters
from gradient import localMaximum

def memoize(f):
   cache = {}

   def memoizedFunction(*args):
      if args not in cache:
         cache[args] = f(*args)
      return cache[args]

   memoizedFunction.cache = cache
   return memoizedFunction


def getGraphs(filename):
   with open(filename, 'r') as graphFile:
      lines = graphFile.readlines()

   pairs = lambda L: tuple(L[i:i+2] for i in range(0, len(L)-1, 2))
   intEdges = lambda L: tuple((int(i), int(j)) for i,j in L)

   graphStrings = [(info.strip().split(), pairs(edges.strip().split()))
      for (info, edges) in pairs(lines)]

   graphInfo = tuple((int(x[0]), int(x[1])) for x,_ in graphStrings)
   graphEdges = tuple(intEdges(edgeList) for _,edgeList in graphStrings)

   return tuple(zip(graphInfo, graphEdges))


def properColoring(edges, colors):
   for e in edges:
      if colors[e[0]] == colors[e[1]]:
         return False
   return True


def numBadEdges(edges, colors):
   count = 0
   for e in edges:
      if colors[e[0]] == colors[e[1]]:
         count += 1
   return count


def allColorings(n, k):
   return itertools.product(range(k),repeat=n)

def anyColoring(n,k):
   return tuple(random.choice(range(k)) for _ in range(n))

def vertexNeighbors(c,i,k):
   newColors = set(range(k)) - set([c[i]])
```

```
    return (tuple(newColor if index == i else color
        for (index,color) in enumerate(c))
        for newColor in newColors)

def neighboringColorings(c, k):
    return (x for i in range(len(c)) for x in vertexNeighbors(c, i, k))


@memoize
def hasProperColoring(g, k):
    # extend this to return the coloring
    ((n, _), edgeList) = g
    for coloring in allColorings(n, k):
        if properColoring(edgeList, coloring):
            return True
    return False


def allEdges(n):
    return ((i,j) for i in range(n) for j in range(n) if i < j)


def sortEdges(edgeList):
    return tuple(tuple(sorted(e)) for e in edgeList)


def isResilient(g, resilience, k):
    (n,m), edgeList = g
    count = 0
    edges = sortEdges(edgeList)
    edgesToCheck = itertools.combinations(set(allEdges(n))
          - set(edges), resilience)

    for newEdges in edgesToCheck:
        newGraph = ((n, m + len(newEdges)), edges + newEdges)
        if not hasProperColoring(newGraph, k):
            return False
        count += 1
        if count % 1000 == 0:
            print(count)

    return True


def tryProveResilience(g, resilience, k, leftVertices=[], rightVertices=[]):
    (n,m), edgeList = g
    count = 0
    edges = sortEdges(edgeList)
    print(edges)
    neighbors = lambda c: neighboringColorings(c,k)
    numSteps = 10000

    if leftVertices == []:
        edgeSet = itertools.combinations(set(allEdges(n)) - set(edges), resilience)
    else: # "bipartite" edges
```

```
      edgeSet = itertools.combinations([(i,j) for i in leftVertices
         for j in rightVertices if i != j], resilience)

   for newEdges in edgeSet:
      newGraph = ((n, m + len(newEdges)), tuple(edges) + newEdges)
      fitness = lambda c: -numBadEdges(newGraph[1], c)

      numAttempts = 0
      while fitness(localMaximum(anyColoring(n, k), fitness, neighbors, numSteps)) != 0:
         numAttempts += 1
         if numAttempts > 20000:
            print(newEdges)
            return False

      count += 1
      if count % 10000 == 0:
         print(count)

   return True


def resilienceProfile(graphs, k, resilienceCap=4):
   # continue computing with only those graphs who passed 1-resilience
   counts = []
   goodGraphs = graphs

   for i in range(1, 1 + resilienceCap):
      goodGraphs = [g for g in goodGraphs if isResilient(g, i, k)]
      counts.append(len(goodGraphs))

   return counts


def analyze(filename, maxk=6):
   graphs = getGraphs(filename)

   print("Percentage␣of␣k-colorable␣graphs␣which␣are␣n-resilient".center(40))
   print(filename)
   print("")

   print('k\\n' + ''.join([("%d" % i).rjust(8) for i in range(1,5)]))
   print('␣␣' + "-"*40)

   for k in range(3, maxk+1):
      kColorableGraphs = [g for g in graphs if hasProperColoring(g, k)]
      # print('%d %d-colorable graphs' % (len(kColorableGraphs), k))
      counts = resilienceProfile(kColorableGraphs, k)

      row = [c * 100.0 / len(kColorableGraphs) for c in counts]
      print(str(k) + '␣␣␣' + ''.join([("%.1f" % x).rjust(8) for x in row]))

   print("")


def combineGraphs(G, H):
```

```
    gDim, gEdges = G
    hDim, hEdges = H
    offset = gDim[0]

    leftVertices = range(gDim[0])
    rightVertices = range(offset + 1, offset + hDim[0])

    combinedGraph = ((gDim[0] + hDim[0], gDim[1] + hDim[1]),
                    gEdges + tuple((i + offset, j + offset) for (i,j) in
     hEdges))
    return combinedGraph, leftVertices, rightVertices


def checkInterEdgeResilience(G, H, resilience, k):
    unionGraph, gVertices, hVertices = combineGraphs(G, H)
    return tryProveResilience(unionGraph, resilience,
       k, leftVertices=gVertices, rightVertices=hVertices)


if __name__ == "__main__":
    pass
    analyze(parameters.filename, maxk=parameters.tableMaxK)

    for filename in ['graph6.txt', 'graph7.txt', 'graph8.txt']:
       analyze('data/' + filename, maxk=3)


    # graphs have the form ((n, m), (e1, e2, ...))
    petersen = ((10,15), ((0,1), (1,2), (2,3), (3,4), (4,5), (5,6), (6,7),
                (7,8), (8,0), (0,9), (3,9), (6,9), (2,7), (4,8), (5,1)))

    durer = ((12,18), ((1,2), (2,3), (3,4), (4,5), (5,6), (6,1),
             (1,7), (2,8), (3,9), (4,10), (5,11), (6,0), (8,10),
             (8,0), (9,7), (9,11), (10,0), (11,7)))

    grotzsch = ((11,20), ((1,3), (1,5), (1,7), (1,9), (1,0), (2,3),
                (3,4), (4,5), (5,6), (6,7), (7,8), (8,9), (9,10),
                (10,0), (2,0), (2,6), (2,8), (4,8), (4,10), (6,10)))

    chvatal = ((12,24), ((1,2), (2,3), (3,4), (1,4), (1,5), (1,6),
               (2,7), (2,8), (3,9), (3,10), (4,11), (4,0), (5,0),
               (5,9), (5,10), (6,7), (6,9), (6,10), (7,0), (7,11),
               (8,0), (8,11), (8,9), (10,11)))

    k33 = ((15,18), ((0,1), (0,2), (0,3), (1,4), (2,5), (3,6), (4,7),
           (4,11), (5,8), (5,12), (6,9), (6,13), (7,10), (8,10),
           (9,10), (11, 14), (12,14), (13,14)))

    print(isResilient(k33, 2, 3))
    print(tryProveResilience(petersen, 2, 3))
    print(tryProveResilience(durer, 4, 4))
    print(tryProveResilience(grotzsch, 4, 4))
    print(tryProveResilience(chvatal, 3, 4))

    negationGadget = ((15,25),
((0,1), (0,3), (0,12), (1,2), (1,10), (2,4), (2,13), (3,4), (3,5),
(4,5), (5,6), (6,7), (6,8), (7,8), (7,9), (8,11), (9,10), (9,12),
```

```
(10,11), (11,13), (12,13), (0,14), (2,14), (9,14), (11,14)))


   clauseGadget = ((31,44),
((1,13), (2,13), (3,14), (4,14), (5,15), (6,15), (7,16), (8,16),
(9,17), (10,17), (11,18), (12,18), (13,19), (14,20), (15,21),
(16,22), (17,23), (18,24), (19,20), (19,25), (20,25), (21,26),
(21,22), (22,26), (23,24), (23,27), (24,27), (25,28), (27,0), (29,0),
(28,29), (28,0), (1,30), (2,30), (3,30), (4,30), (5,30), (6,30),
(7,30), (8,30), (9,30), (10,30), (11,30), (12,30)))


   negationClauseDisconnected = ((46,69),
((1,13), (2,13), (3,14), (4,14), (5,15), (6,15), (7,16), (8,16),
(9,17), (10,17), (11,18), (12,18), (13,19), (14,20), (15,21),
(16,22), (17,23), (18,24), (19,20), (19,25), (20,25), (21,26),
(21,22), (22,26), (23,24), (23,27), (24,27), (25,28), (27,0), (29,0),
(28,29), (28,0), (1,30), (2,30), (3,30), (4,30), (5,30), (6,30),
(7,30), (8,30), (9,30), (10,30), (11,30), (12,30), (31,32), (31,34),
(31,43), (32,33), (32,41), (33,35), (33,44), (34,35), (34,36),
(35,36), (36,37), (37,38), (37,39), (38,39), (38,40), (39,42),
(40,41), (40,43), (41,42), (42,44), (43,44), (31,45), (33,45),
(40,45), (42,45)))

   negationClauseConnected = ((45,69),
((1,13), (2,13), (3,14), (4,14), (5,15), (6,15), (7,16), (8,16),
(9,17), (10,17), (11,18), (12,18), (13,19), (14,20), (15,21),
(16,22), (17,23), (18,24), (19,20), (19,25), (20,25), (21,26),
(21,22), (22,26), (23,24), (23,27), (24,27), (25,28), (27,0), (29,0),
(28,29), (28,0), (1,30), (2,30), (3,30), (4,30), (5,30), (6,30),
(7,30), (8,30), (9,30), (10,30), (11,30), (12,30), (1,31), (1,33),
(1,42), (31,2), (31,40), (2,34), (2,43), (33,34), (33,35), (34,35),
(35,36), (36,37), (36,38), (37,38), (37,39), (38,41), (39,40),
(39,42), (40,41), (41,43), (42,43), (1,44), (2,44), (39,44),
(41,44)))

   clauseClauseDisconnected = ((61, 88),
# first half
((1, 13), (2, 13), (3, 14), (4, 14), (5, 15), (6, 15), (7, 16), (8,
16), (9, 17), (10, 17), (11, 18), (12, 18), (13, 19), (14, 20), (15,
21), (16, 22), (17, 23), (18, 24), (19, 20), (19, 25), (20, 25), (21,
26), (21, 22), (22, 26), (23, 24), (23, 27), (24, 27), (25, 28), (27,
0), (29, 0), (28, 29), (28, 0), (1, 30), (2, 30), (3, 30), (4, 30),
(5, 30), (6, 30), (7, 30), (8, 30), (9, 30), (10, 30), (11, 30), (12,
30),
# second half
(32, 44), (33, 44), (34, 45), (35, 45), (36, 46), (37, 46), (38, 47),
(39, 47), (40, 48), (41, 48), (42, 49), (43, 49), (44, 50), (45, 51),
(46, 52), (47, 53), (48, 54), (49, 55), (50, 51), (50, 56), (51, 56),
(52, 57), (52, 53), (53, 57), (54, 55), (54, 58), (55, 58), (56, 59),
(58, 31), (60, 31), (59, 60), (59, 31), (32, 30), (33, 30), (34, 30),
(35, 30), (36, 30), (37, 30), (38, 30), (39, 30), (40, 30), (41, 30),
(42, 30), (43, 30)))

   clauseClauseConnected = ((61, 88),
# first half
((1, 13), (2, 13), (3, 14), (4, 14), (5, 15), (6, 15), (7, 16), (8,
```

```
16), (9, 17), (10, 17), (11, 18), (12, 18), (13, 19), (14, 20), (15,
21), (16, 22), (17, 23), (18, 24), (19, 20), (19, 25), (20, 25), (21,
26), (21, 22), (22, 26), (23, 24), (23, 27), (24, 27), (25, 28), (27,
0), (29, 0), (28, 29), (28, 0), (1, 30), (2, 30), (3, 30), (4, 30),
(5, 30), (6, 30), (7, 30), (8, 30), (9, 30), (10, 30), (11, 30), (12,
30),

# second half
(1, 44), (2, 44), (34, 45), (35, 45), (36, 46), (37, 46), (38, 47),
(39, 47), (40, 48), (41, 48), (42, 49), (43, 49), (44, 50), (45, 51),
(46, 52), (47, 53), (48, 54), (49, 55), (50, 51), (50, 56), (51, 56),
(52, 57), (52, 53), (53, 57), (54, 55), (54, 58), (55, 58), (56, 59),
(58, 31), (60, 31), (59, 60), (59, 31), (1, 30), (2, 30), (34, 30),
(35, 30), (36, 30), (37, 30), (38, 30), (39, 30), (40, 30), (41, 30),
(42, 30), (43, 30)))

   print(tryProveResilience(negationGadget, 1, 3))
   print(tryProveResilience(clauseGadget, 1, 3))
   print(tryProveResilience(negationClauseDisconnected, 1, 3,
         leftVertices=range(31), rightVertices=range(31,46)))
   print(tryProveResilience(negationClauseConnected, 1, 3,
         leftVertices=range(31), rightVertices=[1,2] + range(31,45)))
   print(tryProveResilience(clauseClauseConnected, 1, 3,
         leftVertices=range(31), rightVertices=[1,2] + range(31,61)))
   print(tryProveResilience(clauseClauseDisconnected, 1, 3,
         leftVertices=range(31), rightVertices=range(31,61)))
```

Listing A.2: A simple gradient ascent algorithm

```
# localMax: 'a, ('a -> number), 'a -> ['a], int -> 'a
def localMaximum(posn, fitness, neighbors, numSteps):
   value = fitness(posn)
   nbrs = iter(neighbors(posn))

   for step in range(numSteps):
      try:
         nextPosn = nbrs.next()
      except:
         break

      nextValue = fitness(nextPosn)

      if nextValue > value:
         posn, value = nextPosn, nextValue
         nbrs = iter(neighbors(posn))

   return posn
```

References

[1] Ackerman, M. & Ben-David, S. (2009). Clusterability: A theoretical study. *Journal of Machine Learning Research - Proceedings Track*, 5, 1–8.

[2] Aharoni, R., Milner, E. C., & Prikry, K. (1990). Unfriendly partitions of a graph. *J. Comb. Theory, Ser. B*, 50(1), 1–10.

[3] Andoni, A., Nikolov, A., Onak, K., & Yaroslavtsev, G. (2014). Parallel algorithms for geometric graph problems. In *STOC* (pp. 574–583).

[4] Arora, S. & Ge, R. (2011). New tools for graph coloring. In *APPROX-RANDOM* (pp. 1–12).

[5] Awasthi, P., Blum, A., & Sheffet, O. (2012). Center-based clustering under perturbation stability. *Inf. Process. Lett.*, 112(1-2), 49–54.

[6] Bazgan, C., Tuza, Z., & Vanderpooten, D. (2010). Satisfactory graph partition, variants, and generalizations. *Eur. J. Oper. Res.*, 206(2), 271–280.

[7] Beame, P., Koutris, P., & Suciu, D. (2013). Communication steps for parallel query processing. In *PODS* (pp. 273–284).

[8] Berger, B. & Rompel, J. (1990). A better performance guarantee for approximate graph coloring. *Algorithmica*, 5(3), 459–466.

[9] Bilu, Y. & Linial, N. (2012). Are stable instances easy? *Combinatorics, Probability & Computing*, 21(5), 643–660.

[10] Blum, A. (1994). New approximation algorithms for graph coloring. *J. ACM*, 41(3), 470–516.

[11] Bramoullé, Y., López-Pintado, D., Goyal, S., & Vega-Redondo, F. (2004). Network formation and anti-coordination games. *Int. J. Game Theory*, 33(1), 1–19.

[12] Bruhn, H., Diestel, R., Georgakopoulos, A., & Sprüssel, P. (2010). Every rayless graph has an unfriendly partition. *Combinatorica*, 30(5), 521–532.

[13] Cai, L. (2003). Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127(3), 415–429.

[14] Cao, Z. & Yang, X. (2012). The fashion game: Matching pennies on social networks. *SSRN*.

[15] Chatzigiannakis, I., Koninis, C., Panagopoulou, P. N., & Spirakis, P. G. (2010). Distributed game-theoretic vertex coloring. In *OPODIS'10* (pp. 103–118).

[16] Chaudhuri, K., Graham, F. C., & Jamall, M. S. (2008). A network coloring game. In *WINE* (pp. 522–530).

[17] Chu, C.-T., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G. R., Ng, A. Y., & Olukotun, K. (2006). Map-reduce for machine learning on multicore. In *NIPS* (pp. 281–288).

[18] Dailey, D. P. (1980). Uniqueness of colorability and colorability of planar 4-regular graphs are np-complete. *Discrete Mathematics*, 30(3), 289 – 293.

[19] Dean, J. & Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1), 107–113.

[20] Dinur, I., Mossel, E., & Regev, O. (2009). Conditional hardness for approximate coloring. *SIAM J. Comput.*, 39(3), 843–873.

[21] Elsässer, R. & Tscheuschner, T. (2011). Settling the complexity of local max-cut (almost) completely. In *ICALP (1)* (pp. 171–182).

[22] Eppstein, D., Bern, M. W., & Hutchings, B. L. (2002). Algorithms for coloring quadtrees. *Algorithmica*, 32(1), 87–94.

[23] Escoffier, B., Gourvès, L., & Monnot, J. (2010). Strategic coloring of a graph. In *CIAC'10* (pp. 155–166). Berlin, Heidelberg: Springer-Verlag.

[24] Farahat, A. K., Elgohary, A., Ghodsi, A., & Kamel, M. S. (2013). Distributed column subset selection on mapreduce. In *ICDM* (pp. 171–180).

[25] Feldman, J., Muthukrishnan, S., Sidiropoulos, A., Stein, C., & Svitkina, Z. (2010). On distributing symmetric streaming computations. *ACM Transactions on Algorithms*, 6(4).

[26] Fortnow, L. (2000). Time-space tradeoffs for satisfiability. *J. Comput. Syst. Sci.*, 60(2), 337–353.

[27] Fotakis, D., Kontogiannis, S., Koutsoupias, E., Mavronicolas, M., & Spirakis, P. (2002). The structure and complexity of nash equilibria for a selfish routing game. In *ICALP* (pp. 123–134). Malaga, Spain.

[28] Garey, M. R. & Johnson, D. S. (1976). The complexity of near-optimal graph coloring. *J. ACM*, 23(1), 43–49.

[29] Garey, M. R. & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

[30] Goodrich, M. T., Sitchinava, N., & Zhang, Q. (2011). Sorting, searching, and simulation in the mapreduce framework. In *ISAAC* (pp. 374–383).

[31] Gourvès, L. & Monnot, J. (2009). On strong equilibria in the max cut game. In *In: Proc. of WINE 2009, Springer LNCS* (pp. 608–615).

[32] Guruswami, V. & Khanna, S. (2004). On the hardness of 4-coloring a 3-colorable graph. *SIAM J. Discrete Math.*, 18(1), 30–40.

[33] Halldórsson, M. M. (1993). A still better performance guarantee for approximate graph coloring. *Inf. Process. Lett.*, 45(1), 19–23.

[34] Håstad, J. (1999). Clique is hard to approximate within $n^{1-\varepsilon}$. *Acta Mathematica*, 182, 105–142.

[35] Hoàng, C. T., Maffray, F., & Mechebbek, M. (2012). A characterization of b-perfect graphs. *Journal of Graph Theory*, 71(1), 95–122.

[36] Hoefer, M. (2007). *Cost sharing and clustering under distributed competition*. PhD thesis, Universität Konstanz, Germany.

[37] Huang, S. (2013). Improved hardness of approximating chromatic number. *CoRR*, abs/1301.5216.

[38] Impagliazzo, R. & Paturi, R. (1999). The complexity of k-sat. *2012 IEEE 27th Conference on Computational Complexity*, 0, 237.

[39] Impagliazzo, R., Paturi, R., & Zane, F. (2001). Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4), 512–530.

[40] Johnson, D. S. (1974). Worst case behavior of graph coloring algorithms. In *Proc. 5th Southeastern Conf. on Comb., Graph Theory and Comput.* (pp. 513–527).

[41] Kamara, S. & Raykova, M. (2013). Parallel homomorphic encryption. In *Financial Cryptography Workshops* (pp. 213–225).

[42] Karloff, H., Suri, S., & Vassilvitskii, S. (2010). A model of computation for mapreduce. In *SODA '10* (pp. 938–948). Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

[43] Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of Computer Computations* (pp. 85–103).

[44] Kawarabayashi, K. & Thorup, M. (2014). Coloring 3-colorable graphs with $o(n^{1/5})$ colors. In *STACS*, volume 25 (pp. 458–469).

[45] Kearns, M., Suri, S., & Montfort, N. (2006). A behavioral study of the coloring problem on human subject networks. *Science*, 313, 2006.

[46] Khanna, S., Linial, N., & Safra, S. (2000). On the hardness of approximating the chromatic number. *Combinatorica*, 20(3), 393–415.

[47] Khot, S. (2001). Improved inaproximability results for maxclique, chromatic number and approximate graph coloring. In *FOCS* (pp. 600–609).

[48] Kobler, D. & Rotics, U. (2003). Edge dominating set and colorings on graphs with fixed clique-width. *Discrete Applied Mathematics*, 126(2-3), 197–221.

[49] Koutsoupias, E. & Papadimitriou, C. (1999). Worst-case equilibria. In *STACS* (pp. 404–413). Trier, Germany.

[50] Král, D., Kratochvíl, J., Tuza, Z., & Woeginger, G. J. (2001). Complexity of coloring graphs without forbidden induced subgraphs. In *WG* (pp. 254–262).

[51] Kumar, R., Moseley, B., Vassilvitskii, S., & Vattani, A. (2013). Fast greedy algorithms in mapreduce and streaming. In *SPAA '13* (pp. 1–10). New York, NY, USA: ACM.

[52] Kumar, V. (1992). Algorithms for constraint-satisfaction problems: A survey. *AI magazine*, 13(1), 32.

[53] Kun, J., Powers, B., & Reyzin, L. (2013). Anti-coordination games and stable graph colorings. In *SAGT* (pp. 122–133).

[54] Lokshtanov, D., Marx, D., & Saurabh, S. (2011). Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105, 41–72.

[55] Mihalák, M., Schöngens, M., Srámek, R., & Widmayer, P. (2011). On the complexity of the metric tsp under stability considerations. In *SOFSEM* (pp. 382–393).

[56] Monderer, D. & Shapley, L. S. (1996). Potential games. *Games and Economic Behavior*, 14(1), 124 – 143.

[57] Monien, B. & Tscheuschner, T. (2010). On the power of nodes of degree four in the local max-cut problem. In *CIAC* (pp. 264–275).

[58] Naor, M. & Stockmeyer, L. (1993). What can be computed locally? In *STOC '93* (pp. 184–193).: ACM.

[59] Pace, M. F. (2012). BSP vs mapreduce. In *Proceedings of the International Conference on Computational Science, ICCS 2012, Omaha, Nebraska, USA, 4-6 June, 2012* (pp. 246–255).

[60] Panagopoulou, P. N. & Spirakis, P. G. (2008). A game theoretic approach for efficient graph coloring. In *ISAAC '08* (pp. 183–195).

[61] Reyzin, L. (2012). Data stability in clustering: A closer look. In *ALT* (pp. 184–198).

[62] Roughgarden, T. & Tardos, E. (2002). How bad is selfish routing? *J. ACM*, 49(2), 236–259.

[63] Sarma, A. D., Afrati, F. N., Salihoglu, S., & Ullman, J. D. (2013). Upper and lower bounds on the cost of a map-reduce computation. In *PVLDB'13* (pp. 277–288).: VLDB Endowment.

[64] Shafique, K. & Dutton, R. D. (2009). Partitioning a graph into alliance free sets. *Discrete Mathematics*, 309(10), 3102–3105.

[65] Shelah, S. & Milner, E. C. (1990). Graphs with no unfriendly partitions. *A tribute to Paul Erdös*, (pp. 373–384).

[66] Shepherdson, J. C. (1959). The reduction of two-way automata to one-way automata. *IBM J. Res. Dev.*, 3(2), 198–200.

[67] Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The hadoop distributed file system. In M. G. Khatib, X. He, & M. Factor (Eds.), *MSST* (pp. 1–10).: IEEE Computer Society.

[68] Szepietowski, A. (1994). *Turing Machines with Sublogarithmic Space*. Ernst Schering Research Foundation Workshops. Springer.

[69] Valiant, L. G. (1990). A bridging model for parallel computation. *Commun. ACM*, 33(8), 103–111.

[70] van den Heuvel, J., Leese, R. A., & Shepherd, M. A. (1998). Graph labeling and radio channel assignment. *J. Graph Theory*, 29(4), 263–283.

[71] Wagner, K. & Wechsung, G. (1986). *Computational Complexity*. Mathematics and its Applications. Springer.

[72] Wigderson, A. (1983). Improving the performance guarantee for approximate graph coloring. *J. ACM*, 30(4), 729–735.

[73] Williams, R. (2008). Time-space tradeoffs for counting NP solutions modulo integers. *Computational Complexity*, 17(2), 179–219.

[74] Zuckerman, D. (2007). Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1), 103–128.