University of Alberta

Library Release Form

Name of Author: Adam Joseph Beacham

Title of Thesis: The Complexity of Problems Without Backbones

Degree: Master of Science

Year this Degree Granted: 2000

Adam Joseph Beacham
10335-118 Street N.W. Apt 203
Edmonton, AB
Canada, T5K 1Y6

Date: ____________________

This calls for a subtle combination
of mathematics and extreme violence.
– Ross Williams on rec.humor.funny

University of Alberta

The Complexity of Problems Without Backbones

by

Adam Joseph Beacham

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 2000

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **The Complexity of Problems Without Backbones** submitted by Adam Joseph Beacham in partial fulfillment of the requirements for the degree of **Master of Science**.

Joseph Culberson
Supervisor

Jim Hoover

Lorna Stewart

Bruce Cockburn
External Examiner

Date:

To my parents
Alan and Jane
and also
my best friend and partner,
Candace

Abstract

The backbone of a boolean satisfiability problem (SAT) instance is the set of literals forced to be true under all satisfying variable assignments. The sudden appearance of a backbone has been correlated with the exceptionally hard problems found near the SAT phase transition. It has been conjectured that the backbone is the cause of poor algorithm performance in this region.

We restrict the notion of a backbone to monotone graph properties, and introduce a parameterized class of backbone-free properties. It is shown that SAT, graph k-coloring, and many other NP-complete problems remain hard even if restricted to the subset of "unfrozen" (backbone-free) instances. We show that as the level of unfrozeness is increased for k-coloring, we move from NP-complete to P. We also investigate the complexity of determining whether or not an instance is maximal with respect to a monotone NP-complete property X. This is typically easy, but hard for certain X.

Acknowledgements

First, I must thank my supervisor, Joe Culberson. If he had not shared his interest in the frozen development process with me, this thesis could never have been written. He was always available to give me advice, and our multi-hour research discussions were a highlight of my degree. Thanks, Joe.

I must always thank Neil Burch, who was willing to take the time to go over early versions of many of the proofs in this thesis with me. The insight that lead to the proof that **U**(IS) is NP-complete is his.

My parents have been an unwavering source of support and love. They were always there to give me a push when I needed motivation, and were there to pull me up when I was blue. You two mean the world to me; thank you for all you've done.

Finally, I thank my partner Candace for her love and friendship. Thanks for being there when I needed you.

Contents

List of Figures

List of Tables

List of Symbols

Graph Theory

$\cong$	Isomorphism relation
$E(G)$	The set of edges in the graph G
E^*	The set of all edges and non-edges in G; $E^* = E \cup \overline{E}$
$G = (V, E)$	A graph with vertex set V and edge set E
G^c	The complement graph $(V, \overline{E})$
$G + e$	The graph $(V, E \cup \{e\})$
$G - e$	The graph $(V, E \setminus \{e\})$
$G[V']$	The subgraph induced on G by the vertex set V'
$H \times G$	The join of graphs H and G
K_n	An n-clique; the complete graph on n vertices
$N(v)$	The neighborhood of a vertex v
$V(G)$	The set of vertices in the graph G

Important Sets

$[a, b]$	The closed interval. $x \in [a, b] \iff a \leq x \leq b$.
$\mathbb{N}$	The set of natural numbers: $\{1, 2, 3, \ldots\}$
$\mathbb{R}$	The set of real numbers

Complexity Theory

NP	The class of decision problems whose proofs are checkable in polynomial time
P	The class of decision problems solvable in polynomial time
$O(f(n))$	Big O of $f(n)$ (sec 2.1)
$\Omega(f(n))$	Big Omega of $f(n)$ (sec 2.1)
$\Theta(f(n))$	$O(f(n)) \cap \Omega(f(n))$ (sec 2.1)
$\propto_m$	Many-to-one polynomial reduction
$\propto_T$	Polynomial reduction in the sense of Turing

Problems

k-COL	k-colorability of a graph
k-SAT	Boolean satisfiability in conjunctive normal form; k literals per clause
HP	Hamiltonian path
HC	Hamiltonian cycle
H3K2	3-arity hypergraph 2-coloring
NAE-3SAT	Not all equal 3-SAT
U(X)	Unfrozen version of monotone property X

Chapter 1

Introduction

The backbone of a boolean satisfiability problem (SAT) is the set of literals forced to be true under all satisfying variable assignments. The sudden appearance of a backbone has been correlated with the exceptionally hard problems found near the SAT phase transition. It has been conjectured that the backbone is the cause of poor algorithm performance in this region. We investigate the validity of this claim, and answer the question "Is the existence of a backbone necessary for an instance to be difficult?" with a firm *no*.

In fact, we introduce classes of problem instances that are arbitarily "far away" from having a backbone, and show that for some NP-complete problems even instances such as these can be hard to solve. On the other hand, in the case of k-coloring, we find that as we move from coloring all graphs to coloring colorable graphs without a backbone to coloring graphs $\frac{k(k+1)}{2}$ steps removed from having a backbone, the problem undergoes a transition from being NP-complete to being trivially solvable. Considerable progress is made in locating the transition where the problem moves from intractable to polynomial time solvable.

Chapter 2 presents the concepts and definitions needed to understand the rest of the thesis. The first half introduces basic complexity theory, including the difference between a problem and an instance, asymptotic notation, and the definitions of important complexity classes such as P and NP. We also define basic graph theory notation.

The second part of the chapter surveys the literature on phase transitions. We define what the SAT phase transition is, and explain why this phenomenon has attracted many researchers. We describe the easy-hard-easy problem instance distribution associated with the phase transition, and cite literature describing both empirical and theoretical attempts to pinpoint the location of the phase transition. Other literature involves the difference between sharp and coarse thresholds, and its possible relevance to the difficulty of problems.

The chapter concludes with a review of literature dealing with the backbone of a problem instance. Several possible definitions for the backbone found in the literature are described, and we discuss the merits of each. Literature is cited that supports the importance of the backbone and its implications for the difficulty of search.

New results begin in chapter 3, where we first introduce the notion of an instance being *unfrozen* with respect to a monotone property X; that is, an instance of X with no backbone. This leads to the definition of $\mathbf{U}(X)$, the problem class consisting of unfrozen instances of X. Since $\mathbf{U}(X)$ is itself a monotone graph property, we can recursively define $\mathbf{U}^i(X) \stackrel{\text{def}}{=} \mathbf{U}(\mathbf{U}^{i-1}(X))$. Given that the literature states that hard instances at the phase transition are associated with a large backbone, one might expect that if an instance is in $\mathbf{U}(X)$ (having no backbone at all), then it should be quite simple to determine that it has property X. In fact, we show that in general this is not the case. For NP-complete problems such as SAT, Hamilton cycle, and independent set, finding a solution to an unfrozen instance is proved to be NP-hard.

Our most fascinating and difficult results are proved in chapter 4. In this chapter, we prove that on the graph coloring experiences a transition from being NP-complete to being polynomial time solvable. To be precise, we show that for $i \leq 1$ $\mathbf{U}^i(k\mathrm{-COL}) \in$ NP-complete, while for $i \geq \binom{k}{2}$, $\mathbf{U}^i(k\mathrm{-COL}) \in$ P. Why does this transition exist for coloring, yet provably not appear for Hamilton cycle or independent set? What can be learned about the boundary between P and NP by investigating the details of this transition?

In chapter 5 we take a side trip to investigate the complexity of solving maximally constrained problems. We prove that many of the standard NP-complete problems are easily solved if we restrict our attention to maximally constrained instances. However, not all NP-complete problems become easy when so constrained. We prove that solving a maximally constrained version of subgraph isomorphism is isomorphism complete, and present a montone graph property that remains NP-hard to solve even when restricted to the subclass of instances which no longer have the property if any single edge is added to the graph.

Finally, chapter 6 summarizes some of the main results and suggests directions for future research.

Chapter 2

Background

2.1 Introduction to Complexity

We need to differentiate between a *problem* and an *instance* of a problem. Informally, a problem is a question such as "What is the result of multiplying two numbers together?" or "Is a particular item in a sorted list?" An instance of a problem is a specific question such as "What is 12×71?" or "Is 7 in the list -5, 2, 6, 7, 8, 9, 10?" If an algorithm solves a problem, then it must return a correct answer for every possible instance of that problem.

The efficiency of an algorithm is usually defined by the number of steps it requires to solve an instance of a problem according to the size n of the input. The size of an instance is a positive integer that describes the number of components in that instance. For example, in the problem of finding a particular value in a sorted list of integers, one would normally take the size of an instance to be the number of integers in the list. This notion allows us to theoretically compare the efficiency of algorithms: if algorithm A takes $A(n) = n$ steps to find a value in a list containing n elements while algorithm B takes $B(n) = \log_2(n)$ steps, then B is clearly more efficient than A.

Typically, an algorithm's efficiency is specified by its *asymptotic behavior*; that is, how it behaves as the input size n becomes sufficiently large. This is formalized by "Big O," "Omega," and "Theta" notation.

Definition: Let $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$. Then $O(g(n))$ ("big O of $g(n)$") is the set of all $f : \mathbb{N} \to \mathbb{R}^{\geq 0}$ such that there exists $c \in \mathbb{R}^{+}$, $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, we have $f(n) \leq cg(n)$. ■

Definition: Let $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$. Then $\Omega(g(n))$ ("Omega of $g(n)$") is the set of all $f : \mathbb{N} \to \mathbb{R}^{\geq 0}$ such that there exists $c \in \mathbb{R}^{+}$, $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, we have $f(n) \geq cg(n)$. ■

Definition: For $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$, $\Theta(g(n))$ ("Theta of $g(n)$") is equal to $O(g(n)) \bigcap \Omega(g(n))$. ■

We observe that the notations $\in O$, $\in \Omega$, and $\in \Theta$ are all reflexive ($f(n) \in O(f(n))$) and transitive ($f(n) \in O(g(n)), g(n) \in O(h(n)) \implies f(n) \in O(h(n))$). Furthermore, $\in \Theta$ is symmetric ($f(n) \in \Theta(g(n)) \implies g(n) \in \Theta(f(n))$). Intuitively, $f(n) \in O(g(n))$ means $f(n)$ is bounded from above (within a constant factor) by $g(n)$, $f(n) \in \Omega(g(n))$ means $f(n)$ is bounded from below (within a constant factor) by $g(n)$, and $f(n) \in \Theta(g(n))$ means f and g are of the exact same order.

We note the following additional properties for big O; similar properties hold for Ω and Θ.

- $O(f(n) + g(n)) = O(\max\{f(n), g(n)\})$
- $O(cf(n)) = O(f(n))$ for any $c \in \mathbb{R}^+$.

These properties make asymptotic notation useful for analyzing the efficiency of algorithms. It can greatly simplify the mathematics involved, as (for instance), we see that $O(n^3/17 + n^2 + n\log(n) + 176) = O(n^3)$. Furthermore, the fact that $O(cf(n)) = O(f(n))$ means that we can ignore the speed of the machine on which we implement an algorithm when we analyze the algorithm's efficiency. If an algorithm takes $100n$ microseconds on one machine and n microseconds on another, we can say that the algorithm itself takes $O(n)$ time. We also see that an algorithm that runs in $O(n^2)$ will eventually outperform an algorithm that takes $O(n^3)$ time as n increases, even if the $O(n^2)$ algorithm is implemented on a personal computer from the 1980s while the $O(n^3)$ algorithm runs on a modern-day supercomputer.

We will say that an algorithm takes *polynomial time* to solve a problem if it takes $O(n^k)$ time for some fixed $k \in \mathbb{N}$.

We will also make use of the following two definitions, which are used to relate functions that grow at very different rates.

Definition: If $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$ are such that $\lim_{n\to\infty} f(n)/g(n) = 0$, we write $f(n) \in o(g(n))$. ■

Definition: If $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$ are such that $\lim_{n\to\infty} f(n)/g(n) = \infty$, we write $f(n) \in \omega(g(n))$ ■

2.1.1 P and NP

The definitions for the classes P and NP in this section follow those given in Brassard and Bratley [7]. The canonical source for information on the theory of NP-completeness is Garey and Johnson [19].

We now introduce two important classes of problems: P and NP. These classes consist of *decision problems*; that is, problems for which the answer is either *yes* or *no*, or (equivalently) *true* or *false*. Thus, problems such as "Is the number p a prime?" is a decision problem, while "Find the factors of p" is not. A decision problem X can be viewed as a set of instances with *yes* answers; anything not in X is an instance whose answer is *no*.

Definition: P is the class of decision problems that can be solved by a polynomial time algorithm. ■

We say that problems in P have *efficient* solutions. Of course, this class potentially contains problems for which the best algorithm requires $\Theta(n^{100})$ time—not what one would normally consider efficient. On the other hand, any problem that is *not* in P is clearly intractable. Furthermore, all the standard deterministic models of computation can emulate each other within a polynomial factor in time; hence, P is a robust class that does not change if we alter our model of computation.

The class NP requires an auxiliary definition: that of a proof system. [7, 19]

Definition: Let X be a decision problem, and Q an arbitrary set called the *proof space* of X. A *proof system* for X is a subset $F \subseteq X \times Q$ such that for all $x \in X$, there exists $q \in Q$ with $< x, q > \in F$; moreover, for all $x \notin X$, there is no such $q \in Q$ with $< x, q > \in F$. Any q such that $< x, q > \in F$ is called a *proof* or *certificate* for $x \in X$. ■

For example, if X is the set of all graphs containing at least one triangle, then we may take Q as the set of all 3-tuples of vertices. We then define $< G, q > \in F$ if and only if the three vertices specified by q induce a triangle in G.

Also, note that F is itself a decision problem corresponding to the question, "Is the pair $< x, q >$ in F?" The class NP (*nondeterministic polynomial-time*) consists of all problems with an efficiently checkable proof system.

Definition: NP is the class of decision problems X that have a proof system $F \subseteq X \times Q$ such that there exists a polynomial $p(n)$ and a polynomial-time algorithm A such that

- For all $x \in X$ there exists a $q \in Q$ such that $< x, q > \in F$ and, furthermore, the size of q is at most $p(n)$, where n is the size of x.
- For all pairs $< x, q > \in X \times Q$, the algorithm A can verify whether or not $< x, q > \in F$. In other words, the decision problem $F \in \mathrm{P}$.

■

Without proof, we state the following theorem. [7, 19]

THEOREM **2.1.1** $\mathrm{P} \subseteq \mathrm{NP}$

2.1.2 NP-complete Problems

We now develop the idea of polynomial reductions between problems. This will allow us to define the NP-complete problems, which intuitively are the "hardest" problems in NP.

Definition: Let A and B be two problems. We say that A is *polynomially Turing reducible to* B if there exists an algorithm for solving A in a time that would be polynomial if we could solve arbitrary instances of B at unit cost. This is denoted $A \propto_T B$. ■

If we restrict our attention to decision problems, we can define a simpler notion of polynomial reducibility.

Definition: Let X and Y be two decision problems defined on sets of instances I and J, respectively. Problem X is *polynomially many-one reducible* to problem Y if there exists a function $f : I \to J$ computable in polynomial time such that $x \in X$ if and only if $f(x) \in Y$ for any instance $x \in I$ of problem X. This is denoted $X \propto_m Y$ and the function f is called the *reduction function.* ■

These two notions of reducibility are related.

THEOREM **2.1.2** *Consider two decision problems X and Y. If $X \propto_m Y$, then $X \propto_T Y$.*

Note that both $\propto_m$ and $\propto_T$ are transitive relations. We also have the following interesting result.

THEOREM **2.1.3** *Consider two problems A and B. If $A \propto_T B$ and $B \in \mathrm{P}$, then $A \in \mathrm{P}$.*

We are now in position to define the notion of NP-completeness.

Definition: A decision problem X is NP-complete if

1. $X \in$ NP

2. For every $Y \in$ NP, $Y \propto_T X$.

If (2) holds, then we say X is NP-*hard.* ■

As an easy corollary to Theorems 2.1.1 and 2.1.3, we have:

Corollary 2.1.1 *If X is* NP-complete *and $X \in$ P, then* NP = P.

Thus, if any NP-complete problem can be solved in polynomial time, then every problem in NP can be solved in polynomial time. On the other hand, most researchers conjecture that P $\neq$ NP. Under this assumption, no NP-complete problem can be solved in polynomial time. In practice, this means that if P $\neq$ NP, then for every possible algorithm which solves an NP-complete problem, there are infinitely many instances which require exponential time to solve. A natural question to ask is, "Do NP-complete problems exist?" The answer is yes, as shown by Cook [10].

THEOREM **2.1.4 (Cook, 1971)** *The problem of deciding whether or not a Boolean formula in conjunctive normal form is satisfiable (SAT-CNF) is* NP-complete.

The proof is quite difficult, since it must show that *every* problem in NP can be polynomially reduced to SAT-CNF. Proving that a given decision problem X is NP-complete would be quite an exercise if this had to be done every time. Fortunately, once we have a known NP-complete problem, the following theorem can be used to prove that a given $X \in$ NP-complete.

THEOREM **2.1.5** *Let X be a decision problem, and Y an* NP-complete *problem. If X satisfies the following two requirements:*

- $X \in$ NP.

- $Y \propto_T X$

then X is NP-complete.

Some well-known NP-complete problems are outlined in figure 2.1.

It should be noted that although it is conjectured that P $\neq$ NP, no one has managed to prove the existence of any problem $X \in$ NP with $X \notin$ P. On the other hand, no polynomial time algorithm has been found for any NP-complete problem.

Graph k-colorability (k-COL)
Input: A graph $G = (V, E)$ and an integer k.
Problem: Does there exist a k-coloring for G? That is, does there exist a function $C : V \to \{1, 2, \ldots, k\}$ such that for all $uv \in E$, $C(u) \neq C(v)$? NP-complete for $k \geq 3$.

k-Satisfiability (k-SAT)
Input: A boolean expression in conjunctive normal form with each clause containing exactly k literals.
Problem: Does there exist a truth assignment to the variables that satisfies every clause? NP-completefor $k \geq 3$.

Hamiltonian Path/Cycle (HP/HC)
Input: A graph $G = (V, E)$.
Problem: Does G contain a Hamiltonian path (cycle)? That is, does there exist an ordering of G's vertices $v_1, v_2, \ldots, v_n$ such that $\forall i,\ 1 \leq i \leq n-1,\ v_i v_{i+1} \in E$. For a Hamiltonian cycle, $v_n v_1 \in E$ must also hold.

Independent Set (k-set)
Input: A graph $G = (V, E)$ and an integer k.
Problem: Does G contain an independent set of size k? That is, does there exists a subset $V' \subseteq V$, $|V'| = k$ such that the induced graph $G[V']$ contains no edges?

Subgraph Isomorphism (SUBISO)
Input: Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$.
Problem: Is G_1 a subgraph of G_2? That is, does there exist a one-to-one function $f : V_1 \to V_2$ such that if $uv \in E_1$ then $f(u)f(v) \in E_2$?

Figure 2.1: Some well-known NP-complete problems

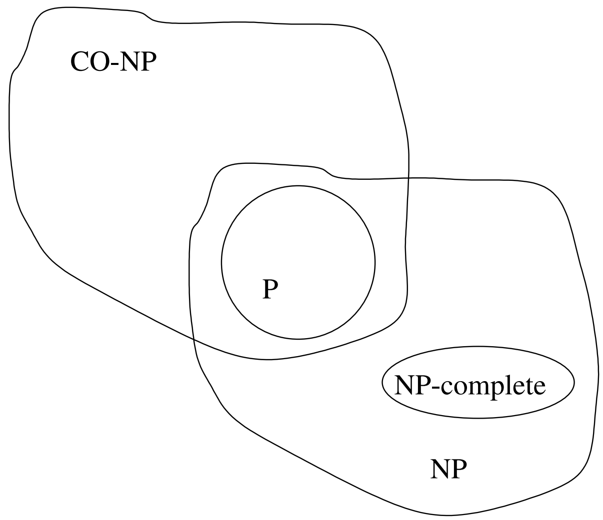

Figure 2.2: The classes P, NP, and CO-NP(assuming P $\neq$ NP and CO-NP $\neq$ NP)

Also, even if P $\neq$ NP, it does not follow that every instance of an NP-complete problem is intractable. Discovering which subclasses of NP-complete instances are in P is an active area of research.

2.1.3 Other Complexity Classes

Definition: The class of decision problems whose complements are in NP is denoted CO-NP. That is, if I is a set of instances and $X \subseteq I$ is a decision problem that belongs to NP, then $I \setminus X \in$ CO-NP. ■

It is conjectured that NP $\neq$ CO-NP. One reason is that many problems in CO-NP do not seem to have certificates that can be checked in polynomial time. For example, there seems to be no way to prove that a graph *cannot* be three-colored without listing all possible colorings and showing that they are not valid. Furthermore, if NP $\neq$ CO-NP then P $\neq$ NP [19]. Thus, this is further support for the conjecture that NP $\neq$ CO-NP. See figure 2.2 for a representation of the relationship between P, NP, and CO-NP assuming P $\neq$ NP and CO-NP $\neq NP$.

By a result of Ladner [28], if P $\neq$ NP then there are infinitely many problems of intermediate difficulty between P and NP-complete; that is, NP $\setminus$ (P $\cup$ NP-complete) $\neq \emptyset$. Although no problem has been proved to be in NP $\setminus$ (P $\cup$ NP-complete), there are some problems for which no polynomial-time algorithm has been found, yet no polynomial reduction from an NP-complete problem has been discovered either. One such problem is *Graph Isomorphism*;

Input: Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$.
Problem: Are G_1 and G_2 isomorphic? That is, does there exist a bijection $f : V_1 \to V_2$ such that $(u, v) \in E_1$ if and only if $(f(u), f(v)) \in E_2$?

Figure 2.3: The Graph Isomorphism Problem

see figure 2.3.

Isomorphism is considered to be one of the best candidates for an intermediate problem. Since many problems can be shown to be as hard as isomorphism, the term *isomorphism complete* has been defined in much the same way as NP-completeness.

Definition: A decision problem X is *isomorphism complete* if and only if $X \propto_T$ GraphISO and GraphISO $\propto_T X$. ■

2.2 Graph Theory

The graph notation and terminology in this section primarily follows that given in West [34]. We will be concerned only with simple, undirected graphs.

Definition: A graph G with n *vertices* and m *edges* consists of a vertex set $V(G) = \{v_1, v_2, \ldots, v_n\}$ and an edge set $E(G) = \{e_1, e_2, \ldots, e_m\}$. Each edge consists of two distinct vertices $u, v \in V(G)$ called its *endpoints*. We write uv for an edge $e = (u, v)$. If $uv \in E(G)$, then u and v are said to be *adjacent*. We will write $G = (V, E)$ to mean G is a graph with vertex set V and edge set E. ■

Definition: The *neighborhood* of a vertex $v \in V$ is the set $N(v) = \{u \in V \mid vu \in E\}$. ■

Definition: If $G = (V, E)$, then the *complement* of G is $G^c = (V, \overline{E})$ where $uv \in \overline{E} \iff uv \notin E$. ■

Definition: A *subgraph* of a graph $G = (V, E)$ is itself a graph $H = (V_H, E_H)$ such that $V_H \subseteq V$ and $E_H \subseteq E$. H is an *induced subgraph* if and only if $uv \in E$ and $u, v \in V_H$ implies $uv \in E_H$. For a vertex set $V' \subseteq V$, the *subgraph induced by* V', denoted $G[V']$ is the subgraph of G such that for $u, v \in V'$, $G[V']$ contains the edge uv if and only if $uv \in E$. ■

Definition: If $G = (V, E)$ and e is an edge uv with $u, v \in V$, we define $G + e$ by $G + e = (V, E \cup \{e\})$. Similarly, $G - e = (V, E \setminus \{e\})$. ■

Definition: The *clique* or *complete graph* on n vertices (denoted K_n) is the graph with vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set $E = \{v_i v_j \mid 1 \leq i \leq n-1, i+1 \leq j \leq n\}$. That is, the graph on n vertices containing all possible edges. ■

Definition: The *join* of two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ is the graph $G \times H = (V_G \cup V_H, E_G \cup E_H \cup \{uv \mid u \in V_G, v \in V_H\})$ ■

2.3 Phase Transitions and Thresholds

Although NP-complete problems are often considered to require exponential time in the worst case to solve, there still may be many instances where a well-designed algorithm can perform well. Thus a great number of papers have been written that investigate the performance of an algorithm on a test bed of problem instances; see [2, 9, 13, 24, 30, 31, 32, 33] for a (non-exhaustive) sample. One concern about such experiments is the need for a large number of problem instances — where do they come from? Although it can be argued that instances from "real world" domains would provide more meaningful results, there are two main problems: the pool of available problems is usually relatively small, and researchers run the risk of tuning their algorithms to work well *only* on instances with internal structures induced by the specific application. Hence a great deal of research has been done on generating random problem instances. Three such random models are presented below.

Definition: $\mathcal{G}(n, p)$ is the probability space of n-vertex graphs where the probability of each edge appearing in the graph is p. ■

Definition: An instance *random clause width SAT model* consists of m clauses on n variables. Each clause c is generated by including each literal l in c with probability p independently of other literals and clauses. ■

Definition: An instance of *fixed clause width k-SAT* consists of m clauses on n literals. Each clause is generated by independently choosing k out of the n variables, and negating them independently with probability 1/2. ■

The choice of the random model to use is very important. For instance, at one time it was believed that SAT was "easy on average," as Goldberg [21] demonstrated that for any p, random clause width instances could be solved in $O(mn^2)$ time on average. Franco and

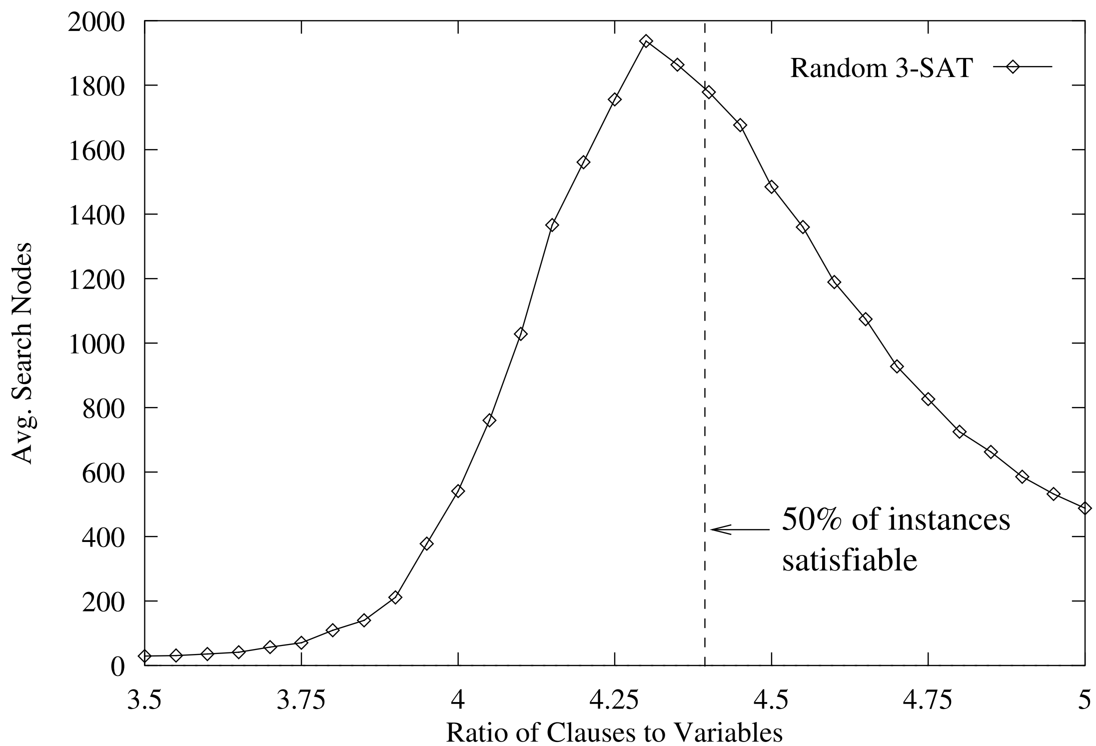

Figure 2.4: Number of backtracking nodes for 200-variable random 3-SAT instances

Swaminathan [17] give a good overview of why this random model is flawed and unsuitable as a source of problem instances.

Cheeseman, Kanefsky, and Taylor [9] were the first to draw attention to the fact that randomly generated instances of NP-complete problems displayed an easy-hard-easy distribution pattern with respect to some parameter. This phenomenon is seen around the *phase transition*, that is the point at which instances pass from being almost always satisfiable to almost always unsatisfiable. On instances far from this transition, algorithms tend to perform well; it is at the critical value where instances have a 50% chance of being satisfiable that very hard instances are found. In most graph problems generated with the $\mathcal{G}(n,p)$ model, the most commonly used parameter is p. The parameter used in the fixed-width clause model is the ratio $\alpha = m/n$ of clauses to variables.

Mitchell, Selman, and Levesque [30] showed empirical results that demonstrated that a hard region of 3-SAT problems occurred near $\alpha \approx 4.3$. My own experiments confirm this; see figure 2.4 for the number of search nodes required by a backtracking algorithm to solve 200-variable 3-SAT instances. Each point in the graph is the average over 1000 instances.

It is not known if there exists an α_0 such that for $\alpha < \alpha_0$ formulas are almost always satisfiable, and for $\alpha > \alpha_0$ almost always unsatisfiable. If such an α_0 does exist, it is known that $3.145 < \alpha_0 < 4.598$. The best lower bound is $\alpha > 3.145$ due to Achlioptas [1]. The best upper bound, determined by Kirousis, Kranakas, and Krizanc [27], is $\alpha < 4.598$.

A similar empirical result for graph 3-coloring was observed by Hogg and Williams [24]. An easy-hard-easy distribution was observed, with the instances 50% probable of being satisfiable occurring when the number of edges in the graph was $\approx 2.3n$. The best known analytically derived value r_0 such that for $r > r_0$ graphs with rn edges are asymptotically non-colorable is $r_0 = 2.495$ due to Kaporis, Kirousis, and Stamatiou [26].

Definition: A *monotone graph property (with respect to edges)* X is a property of graphs such that

- X is invariant under graph automorphisms.
- If G has property X then so does any graph G' on the same number of vertices containing G as a subgraph. The restriction on the number of vertices means that we are concerned only with edges.

A *monotone down property (with respect to edges)* Y is analogous: if a graph G has property Y, then so does any subgraph G' of G, where G' has the same number of vertices as G. ■

We will use the term monotone down to mean "monotone down (with respect to edges)."

Definition: A *threshold function* for a graph theoretic property X is a function $r(n)$ such that

1. When $p(n) \in o(r(n))$, $\lim_{n\to\infty} \Pr[G \in \mathcal{G}(n, p(n)) \implies G \in X] = 0$
2. When $p(n) \in \omega(r(n))$, $\lim_{n\to\infty} \Pr[G \in \mathcal{G}(n, p(n)) \implies G \in X] = 1$

or vice versa. Analogous notions of a threshold function can be defined for random k-SAT. ■

It turns out that *every* monotone graph property has a threshold function [4]. It is conjectured that for NP-complete monotone graph properties, the threshold must be "sharp". To be precise:

Definition: Given a monotone graph property X, let $\mu_p(X)$ be the probability that a random graph with edge probability p will have property X. Fix $\epsilon > 0$, and let p_0 be such that $\mu_{p_0}(X) = \epsilon$, and p_1 such that $\mu_{p_1}(X) = 1 - \epsilon$; both p_0 and p_1 depend on n. Define the threshold length $\delta(n) = p_1 - p_0$. There exists $p_c \in [p_0, p_1]$ such that $\mu_{p_c}(X) = 1/2$. We say that the threshold is *sharp* if $lim_{n\to\infty}\delta(n)/p_c(n) = 0$ and *coarse* otherwise. ■

By a result by Friedgut [18], if X has a coarse threshold, then there is a list of graphs of fixed size such that it can be approximated by the property of having one of those graphs as a subgraph. This would lead to a polynomial algorithm that is highly likely to correctly determine whether or not a given instance is in X. Thus, it is conjectured that all NP-complete problems have sharp thresholds. In the same paper, Friedgut proves that k-SAT does have a sharp threshold.

However, the conjecture that all NP-complete problems have sharp thresholds is false, since Istrate [25] shows that there exist monotone NP-complete decision problems A and B such that A has a coarse threshold while B has a sharp threshold. Similar results hold for problems in P.

On the other hand, just because a problem has a sharp threshold does not seem to imply that instances at the threshold will be difficult. In [13], Culberson and Vandegriend present empirical evidence that randomly generated graphs at the Hamiltonian cycle phase transition are not hard to solve.

Others have conjectured that the continuity of the phase transition is the difference between P and NP. Using techniques from statistical physics, Monasson, Zecchina, Kirkpatrick, and Selman [31] have argued that continuous phase transitions are associated with polynomial complexity problems, while discontinuous phase transitions lead to exponential search times. As support for this notion, they introduce $2+p$-SAT.

Definition: Let $0 \leq p \leq 1$. Then an instance of $2+p$-SAT on n variables with m clauses contains pm 3-clauses and $(1-p)n$ 2-clauses. The clauses are generated by independently choosing the appropriate number of variables and negating each with probability 0.5. ■

$2+p$-SAT provides a continuous transition from 2-SAT (which has an $O(n+m)$ solution) to 3-SAT (which is NP-complete and presumably requires exponential work to solve). The authors show via experimental means that the phase transition is continuous for $p < p_0$ where $0.4 \leq p_0 \leq 0.416$. For $p > p_0$, the phase transition is discontinuous and exponential search times are observed. Analytical studies by Achlioptas *et al.* [3] showing that $2+p$-SAT acts much like 2-SAT for $p < 2/5$ support these empirical results.

Those interested in more details about the generation of hard SAT instances should refer to the survey by Cook and Mitchell [11].

2.4 Backbone of a SAT Instance

Much of the current research on the SAT phase transition involves the development of a *backbone* as one moves across the phase transition [2, 6, 31, 32, 33]. Intuitively, the backbone is the set of literals that are forced to be set to true in any satisfying truth assignment; rigorous definitions may be found in section 2.4.1. If an instance has no backbone, then for every variable x in the instance, there exists a satisfying truth assignment that sets $x = T$, and another truth assignment that sets $x = F$.

Parkes [32] investigated the number of *unary prime implicates* (UPIs) in satisfiable SAT instances generated near the phase transition. A unary prime implicate is a literal entailed by the formula. That is, for a satisfiable formula F, x is a UPI exactly when $F \implies x$. In terms of the backbone $B(F)$, x is a UPI $\iff$ $x \in B(F)$. Parkes observed that instances containing 85-95% of their variables in UPIs took longer for the popular local search algorithm WSAT to find a solution. As UPIs appear at the phase transition, this is one possible explanation for the appearance of hard SAT instances in this region.

In Achlioptas *et al.* [2], the authors use the quasigroup completion problem to generate satisfiable SAT instances. They observe an easy-hard-easy pattern to the instances which is correlated to the size of the backbone. When the backbone is very small or very large, the instances tend to be easy to solve, while those in between these extremes are quite difficult.

In a similar vein, Singer *et al.* [33] argue that the high cost random SAT instances for WSAT are those with large backbones that are *fragile*. A *fragile backbone* is one that almost disappears if any small set of clauses is removed from the instance. The authors show that the clauses most often unsatisfied during local search are precisely those that have the largest effect on the size of the backbone. They also propose a hypothesis to explain how WSAT becomes caught in local minima while solving instances with a large fragile backbone. By demonstrating why popular algorithms have difficulty with these SAT instances, the authors hope to spur the discovery of more efficient algorithms.

The analysis of the 2-SAT transition in Bollobás *et al.* [6] depended upon using the average density of the spine as an *order parameter* (see section 2.3). With this tool, they are able to show (with high probability) the location of the 2-SAT transition for finite problem instances; recall that in the limit the phase transition occurs when the ratio of clauses to variables is 1. If $\alpha = m/n$, then the probability of satisfiability is greater than $1-\delta$ if $\alpha < 1-\Theta(n^{-\frac{1}{3}})$ and less than δ if $\alpha > 1+\Theta(n^{-\frac{1}{3}})$; the implicit constants in Θ depend upon δ. Furthermore, they are able to prove that the 2-SAT phase transition is continuous. The authors state that they expect the spine will be a useful tool for use in analyzing satisfiability problems.

In Monasson *et al.*, the authors summarize results they obtained previously. The spin glass model outlined in definition 1 (see section 2.4.1) is used as an order parameter to find the SAT/UNSAT transition in $2+p$-SAT. Statistical physics methods are employed to suggest that for some $p_0 \in [0.4, 0.416]$, if $p < p_0$ then the transition is continuous and occurs at a clause to variable ratio of $1/(1-p)$; this is in agreement with rigorous results shown by Achlioptas, Kirousis, Kranakis and Krizanc [3].

Culberson and Gent [12] have shown that a frozen structure (i.e., a backbone-like structure) appears suddenly near the phase transition for graph coloring. They then use this structure to estimate the probability of colorability in random graphs.

Thus, the backbone has at least two important uses:

- For use as an order parameter in statistical physics analysis of satisfiability and other NP-complete problems.
- As a way to explain why algorithms have difficulties solving instances selected from the phase transition.

2.4.1 Backbone Definitions

Researchers have suggested a variety of definitions to formalize this concept:

1. Monasson, Zecchina, *et al.* [31] look at the SAT model in terms of a physical system known as a *diluted spin glass model*. In this, we have N spin variables S_i which may take on the values 1 or -1: $S_i = 1$ if the corresponding SAT variable x_i is true, $S_i = -1$ if x_i is false. To each configuration (assignment) of the spin variables, they associate an energy, E, equal to the number of clauses violated by the corresponding truth assignment. A ground state configuration is one which minimizes the energy; thus, for a satisfiable instance, the ground state configurations will correspond to satisfying truth assignments. Let N_{GS} denote the number of ground states. For each i, let $m_i = \frac{1}{N_{GS}} \sum_{g=1}^{N_{GS}} S_i^g$ where S_i^g is the configuration of the ith spin variable in

the gth ground state. Hence, m_i is the average value of the spin S_i over all energy-minimizing configurations. Each $m_i \in [-1, 1]$. Furthermore, $m_i = \pm 1$ if and only if the corresponding boolean variable takes on the same value in all ground states. The *backbone* is exactly those x_i such that $m_i = \pm 1$.

2. Bollobás, *et al.* define the *spine* in [6]. Given a formula F in conjunctive normal form, the spine $S(F)$ is the set of literals x such that there is a satisfiable subformula H of F with $H \wedge x$ unsatisfiable,

$$S(F) = \{x | \exists H \subseteq F, H \text{ is SAT and } H \wedge x \text{ is UNSAT}\}$$

3. Our definition states that the literal x is in the backbone of the formula F if the formula $F \wedge \overline{x}$ is unsatisfiable.

$$B(F) = \{x | F \wedge \overline{x} \text{ is UNSAT}\}$$

All these definitions define the same backbone for satisfiable instances. Since we will be concerned mainly with formulas with solutions, definition 3 is the best choice due to its simplicity.

Definition 1 is obviously quite unwieldy, which is a consequence of the authors' desire to analyze SAT instances with tools from statistical physics. This extra baggage is unnecessary if one's analysis does not use these techniques. Their definition also suffers from the fact that it is non-monotonic; the backbone can grow and shrink as clauses are added to the instance. In particular, this occurs when adding a clause increases the energy in the ground state. However, this definition allows fractional values of frozenness, which could be of some use in certain applications. For example, if the variable corresponding to m_i is true in 95 of 100 satisfying assignments, then $m_i = (95 + 5(-1))/100 = 0.9$.

Definition 2 is the most powerful. The spine increases monotonically as clauses are added to an instance. The spine also has a meaningful interpretation for unsatisfiable instances when the definition is extended to monotonic graph properties (see lemma 3.1.1 in section 3.1 for more details). However, computing the spine for an unsatisfiable instance is computationally prohibitive since, for every literal x, we must look at *all* subformulae $H \subseteq F$ and check two conditions: that H is satisfiable, and that $H + x$ is unsatisfiable.

Chapter 3

Unfrozen Problems

As we saw in section 2.4, the development of a backbone is considered by many researchers to be correlated with difficult instances at the phase transition. While there is evidence to support this, it does not prove that the existence of a backbone is necessary for an instance to be difficult. Are instances without backbones easy to solve? Furthermore, how difficult is it to check for the existence of a backbone? If it could be done in polynomial time, then we would potentially have a tool to determine which instances are likely to be intractable before attempting to solve them.

We will prove that the answer to both of these questions is negative. That is,

- For many NP-complete properties, determining whether or not a given instance has a backbone is NP-complete.
- Finding a solution to a backbone-free instance is NP-hard.

Hence, it seems unlikely that the backbone is the sole cause of hard instances. In fact, in this chapter we show that even determining whether or not an instance of independent set, SAT, or Hamilton cycle has a backbone is itself an NP-complete problem. What is more, finding a solution to such an instance is itself NP-hard. To begin, we first must define the notion of the backbone of a monotone graph property.

3.1 The Backbone Extended to Monotonic Graph Properties

For SAT, the backbone is the set of literals whose addition as singleton clauses would render the instance unsatisfiable. This notion has an extension to the set of edges that may not be added to a graph if it is to maintain a given property X.

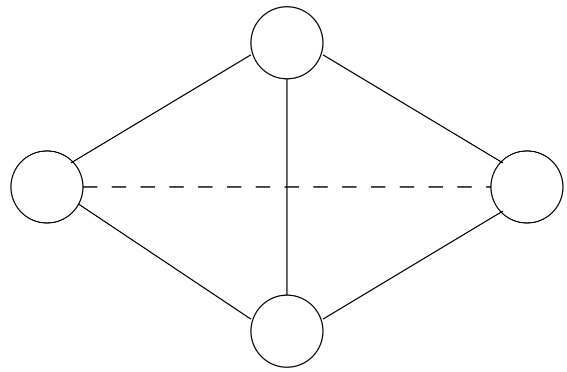

Figure 3.1: With respect to 3-colorability, the dashed edge is frozen out (i.e., it is in the frozen set)

Let X be a *monotone down* graph property (see section 2.3). For a graph $G = (V, E)$, we will define the *frozen set* of G with respect to X by:

$$F(G, X) = \{e \in E \cup \overline{E} | G + e \notin X\}$$

Note that it is not necessary in the above definition to have $G \in X$. If $G \notin X$, then $F(G, X) = E^*$, where $E^* = E \cup \overline{E}$. Although this is not particularly useful, it is less complicated than trying to restrict the definition of $F(G, X)$ to apply only to graphs with the property X. Later definitions that build on the frozen set are simpler than they would be if we required that $G \in X$.

For a graph $G \in X$, we refer to edges in $F(G, X)$ as the *frozen out* edges—those edges that cannot be added to G without taking it outside of X. The dashed edge in figure 3.1 is frozen out with respect to the property of 3-colorability. Without the edge, the graph is 3-colorable, while its addition results in a K_4 graph which is not 3-colorable. A *frozen in* edge is one whose addition to G makes absolutely no difference with respect to X. To be precise, an edge $e \in E^*$ is *frozen in* in G if for every $H \supseteq G$, $H \in X \Longrightarrow H + e \in X$. The dashed line in figure 3.2 is a frozen in edge (with respect to 3-coloring). Since vertices a and b must have the same color, it follows that the endpoints of the dashed line will have different colors in any valid 3-coloring, it follows that an colorable supergraph will remain 3-colorable even if this edge is added.

It should be apparent that the frozen set is very similar to the backbone of SAT instances–one simply replaces edges with literals. In a similar way, the spine (see definition 2 in section 2.4.1) of a graph G with respect to the monotone down graph property X can be

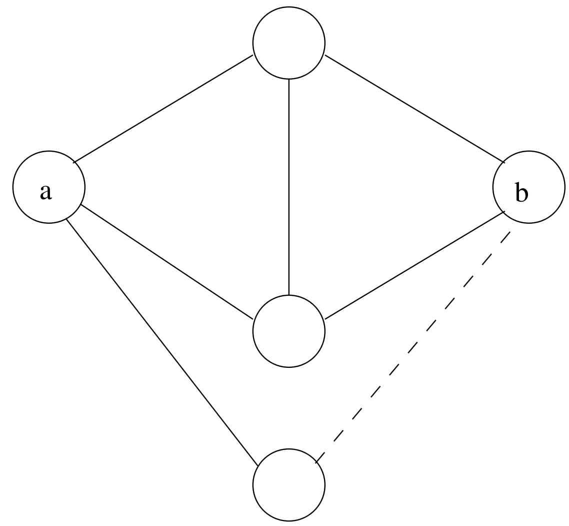

Figure 3.2: With respect to 3-colorability, the dashed edge is frozen in

defined as:

$$S(G,X) = \{e \in E \cup \overline{E} | \exists H \subseteq G \text{ s.t. } H \in X, H + e \notin X\}$$

This definition of the spine has meaning even for unsatisfiable instances:

Lemma 3.1.1 *Let X be a monotone down property such that the empty graph has property X. Then for a graph $G \notin X$, an edge $e \in E$ is in $S(G,X)$ if and only if e is in a minimal subgraph of G that does not have property X.*

Proof: Suppose $e \in S(G,X)$. Let $H \subseteq G$ be such that $H \in X$, $H + e \notin X$. Let H' be a minimal subgraph of $H + e$ such that $H' \notin X$; then $e \in E(H')$, as $H \in X \implies H' - e \in X$ since $H' - e \subseteq H$ and X is monotone down.

On the other hand, if H is a minimal subgraph of G containing e that is not in X, then $G \supseteq (H - e) \in X$ which means $e \in S(G,X)$. ∎

To illustrate the lemma, see figure 3.3. The spine identifies the edges that are preventing the graph from being 3-colorable. The solid edges are in the spine, since they are part of a minimal uncolorable subgraph (a K_4). The dashed line is not in the spine, since every uncolorable subgraph containing that edge itself contains an uncolorable subgraph without it. Note that since the graph is uncolorable, the frozen set $F(G, \text{3-COL})$ would contain all possible edges.

We say that G is *unfrozen with respect to* X if and only if $F(G,X) = \emptyset$; we denote this by $G \in \mathbf{U}(X)$. Thus, graphs in $\mathbf{U}(X)$ are "backbone-free" with respect to the property X.

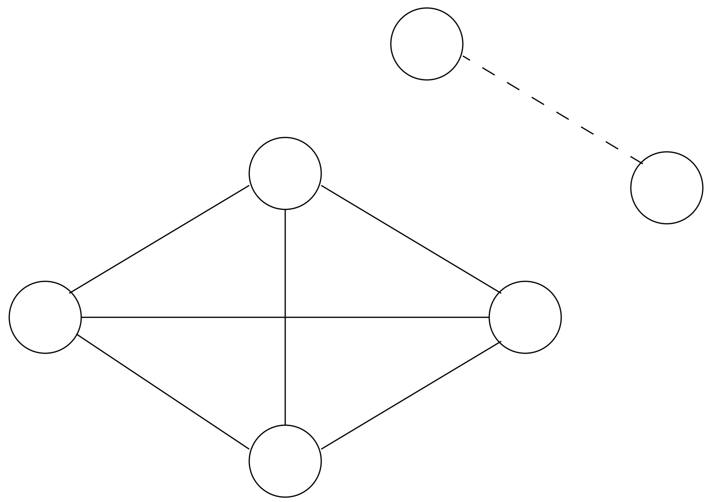

Figure 3.3: With respect to 3-colorability, the solid edges are in the spine while the dashed edge is not.

Note that $\mathbf{U}(X)$ is also a monotone down graph property; hence, $\mathbf{U}(\mathbf{U}(X))$ is well-defined. In general, for a monotone down property X we recursively define $\mathbf{U}^i(X)$ by:

$$\begin{aligned} \mathbf{U}^0(X) &= X \\ \mathbf{U}^{i+1}(X) &= \mathbf{U}(\mathbf{U}^i(X)) \end{aligned}$$

Equivalently, $G \in \mathbf{U}^i(X)$ if for any i edges $e_1, e_2, \ldots, e_i$, we have $G + e_1 + e_2 + \cdots + e_i \in X$.

THEOREM **3.1.1** *Let X be a monotone down graph property in* NP. *Then for fixed i,* $\mathbf{U}^i(X) \in \mathrm{NP}$.

Proof: Proof is by induction on i. For $i = 0$, $\mathbf{U}^0(X) = X \in \mathrm{NP}$ by assumption. Now, assume true for $\mathbf{U}^i(X)$, i.e. $\mathbf{U}^i(X) \in \mathrm{NP}$. Then a certificate proving $G \in \mathbf{U}^{i+1}(X)$ consists of $\binom{n}{2}$ certificates for $\mathbf{U}^i(X)$ showing that $G + e \in \mathbf{U}^i(X)$ for each possible edge e. Since each certificate for $\mathbf{U}^i(X)$ has size polynomial in the size of G and $i + 1$ is a constant, it follows that the certificate for $\mathbf{U}^{i+1}(X)$ is also polynomially-sized. ■

We will also use the notation $\mathbf{U}(X)$ for certain monotone properties that are not graph properties. In general, $\mathbf{U}(X)$ means, "The instance remains in X even if we add any single constraint of a standard form to the instance." The standard form of a constraint depends on X: for 3-SAT, we will use a 3-clause since it is the clauses that restrict the possible values of the variables. We will also occasionally use the notation $Q \in \widehat{\mathbf{U}}^i(X)$ to mean that an arbitrary assignment to i of the variables in Q can be extended to a satisfying assignment of all the variables in Q. Thus, using the definitions in section 2.4, a 3-SAT instance has no backbone if it is in $\widehat{\mathbf{U}}$(3-SAT).

Additionally, to make all the properties we consider monotone down, we will say that $G \in \mathbf{HC}$ if there is a Hamiltonian cycle in the *complement* G^c of G. This is merely a notational convenience so that we can avoid defining $\mathbf{U}(X)$ for monotone up properties. We will do the same for CLIQUE and **HP**.

Note that if $X \in \mathrm{P}$ then $\mathbf{U}(X) \in \mathrm{P}$ as well. By monotonicity, it follows that for fixed $i, j \geq 0$, $\mathbf{U}^i(X) \in \mathrm{P} \Longrightarrow \mathbf{U}^{i+j}(X) \in \mathrm{P}$.

We can define an analagous notion to the frozen set for graphs that do not have a certain property.

Definition: Let X be a monotone down graph property. Then the *conflict set* for a graph G is the set

$$C(G, X) = \{e \in E | G - e \in X\}$$

Analgous to the unfrozen property $U(X)$, we have the property $D(\neg X)$ defined by $G \in D(\neg X) \iff (C(G, X) = \emptyset)$. We define $D^i(X)$ by

$$\begin{aligned} D^0(\neg X) &= \neg X \\ D^{i+1}(\neg X) &= D(D^i(\neg X)) \end{aligned}$$

Equivalently, $G \in D^i(\neg X)$ if there exist i edges $e_1, e_2, \ldots, e_i$ such that $G - e_1 - e_2 - \cdots - e_i \in X$. We define D on the negation $\neg X$ of X simply so that $D^i(\neg X)$ is a monotone down graph property. ■

We should note that there are at least two other notions used when the concept of determining if a graph is backbone-free with respect to a property X is discussed.

Definition:

- The problem $\mathbf{BB}(G, e, X)$ answers whether or not $G + e$ is has property X.
- The problem $\mathbf{BB}(G, X)$ requires an algorithm that returns an edge e in $F(G, X)$, or $\emptyset$ if no such edge exists.

■

THEOREM **3.1.2** *Let G be a graph and X a monotone down graph property. Then,* *1.* $X \propto_T \mathbf{BB}(G,e,X)$, $\mathbf{BB}(G,e,X) \propto_T X$

2. $\mathbf{U}(X) \propto_T \mathbf{BB}(G,X) \propto_T \mathbf{BB}(G,e,X)$

Proof: For (1), notice that $F(G,X) \cap E \neq \emptyset$ if and only if $G \notin X$. Thus, the question of whether or not $G \in X$ can be answered by checking $\mathbf{BB}(G,e,X)$ for each $e \in E$. The opposite reduction $\mathbf{BB}(G,e,X) \propto_T X$ is clear. As for (2), $\mathbf{BB}(G,X) = \emptyset \iff G \in \mathbf{U}(X)$, so $\mathbf{U}(X) \propto_T \mathbf{BB}(G,X)$. Furthermore, $\mathbf{BB}(G,X) \propto_T \mathbf{BB}(G,e,X)$ since $\binom{n}{2}$ calls to an algorithm for $\mathbf{BB}(G,e,X)$ allows us to answer $\mathbf{BB}(G,X)$. ■

Since X and $\mathbf{BB}(G,e,X)$ are polynomially equivalent by part 1 of theorem 3.1.2, it is clear that asking for the elements of $F(G,X)$ for an NP-complete property X is itself NP-complete. Thus, if one equates the question, "Does G have a backbone?" with the ability to answer $\mathbf{BB}(G,e,X)$, then the question of how difficult it is to identify backbone-free instances has a very simple answer!

On the other hand, there is no obvious way to use $\mathbf{BB}(G,X)$ to answer $\mathbf{BB}(G,e,X)$ since we have no control over which edge $\mathbf{BB}(G,X)$ returns. It is less clear how an algorithm for $\mathbf{U}(X)$ would allow us to solve $\mathbf{BB}(G,e,X)$ or X. In fact, since there exist NP-complete problems X such that $\mathbf{U}(X) \notin$ NP-complete unless P = NP (see chapter 4), it is very likely that $X \not\propto_T \mathbf{U}(X)$ for all X.

3.1.1 Related Research

Some researchers have considered graph properties which remain invariant under the addition or deletion of edges or vertices. In [29], Lindgren constructs an infinite class of graphs which are non-Hamiltonian, yet become Hamiltonian upon the elimination of any single vertex and its adjacent edges. Harary [22] introduced notation for graph properties that either changed or remained the same when a fixed number of vertices or edges are added or removed from a graph. Other researchers [8, 15, 23] have investigated properties such as chromatic sum, edge clique cover number, minimum and maximum degree, and isomorphism.

Although superficially similar to our work on $\mathbf{U}(X)$, these papers do not address the complexity of identifying graphs with these properties. Instead, they take an *extremal graph theory* approach; that is, for a property X, they determine the maximum (minimum) number of edges that an n-vertex graph $G \in X$ may have. Although extremal graph theory is an important area of research (see, for instance, [5, 34]), a literature survey did not discover any publications dealing with the complexity of deciding whether or not a given instance is backbone-free.

3.2 k-SAT

We first define the NP-complete problem Not-All-Equal-k-SAT (NAE-k-SAT).

An instance $Q = (V, C)$ of NAE-k-SAT consists of a set of boolean *variables* $V = \{x_1, x_2, \ldots, x_n\}$ and *clauses* $C = \{c_1, c_2, \ldots, c_m\}$. A *literal* is either a variable x or its negation $\overline{x}$. Each clause c_i contains k literals, $c_i = (l_{i1}, l_{i2}, l_{i3})$. A *satisfying assignment* $f : V \to \{T, F\}$ is a truth assignment of the variables such that every clause contains at least one true and at least one false literal.

Not-All-Equal-3-SAT has the important property that if f is a satisfying truth assignment, then so is the assignment $\overline{f}$ defined by $\overline{f}(v) = \overline{f(v)}\ \forall v \in V$.

THEOREM **3.2.1** *Both* $\widehat{\mathbf{U}}(k - SAT)$ *and* $\mathbf{U}(k - SAT)$ *are* NP-complete.

Proof: That the problems are in NP was shown in section 3.1. We reduce from NAE-k-SAT to $\widehat{\mathbf{U}}(k - \mathrm{SAT})$. Let Q be an instance of NAE-k-SAT defined over the variables $x_1, \ldots, x_n$. Construct a k-SAT instance S over the same variables as follows. For every clause $(l_1, l_2, \ldots, l_k)$ in Q, we insert the clauses $(l_1, l_2, \ldots, l_k)$ and $(\overline{l_1}, \overline{l_2}, \ldots, \overline{l_k})$ into S. Note that any variable assignment to Q will be a correct not-all-equal assignment to $(l_1, l_2, \ldots, l_k)$ if and only if the same variable assignment satisfies $(l_1, l_2, \ldots, l_k)$ and $(\overline{l_1}, \overline{l_2}, \ldots, \overline{l_k})$ in S. Thus, this is a polynomial time reduction from NAE-k-SAT to k-SAT.

On the other hand, if S is satisfiable, then any satisfying assignment can be mapped to another one by inverting the value of every variable. Thus, any literal can be added as a singleton clause to S, and this new instance would still be satisfiable. Thus, S is satisfiable if and only if $S \in \widehat{\mathbf{U}}(k-\mathrm{SAT})$. As $\widehat{\mathbf{U}}(k-\mathrm{SAT}) \subseteq \mathbf{U}(k-\mathrm{SAT})$, we also have $S \in \mathbf{U}(k-\mathrm{SAT})$. The theorem follows. ■

Notice that the proof shows that S is satisfiable if and only if it is also in $\widehat{\mathbf{U}}(k - \mathrm{SAT})$. Hence, finding a satisfying assignment for an instance in $\widehat{\mathbf{U}}(k - \mathrm{SAT})$ is NP-hard.

We see that $\mathbf{U}^i(k - \mathrm{SAT}) \in \mathrm{P}$ for i sufficiently large.

THEOREM **3.2.2** *If* $i \geq 2^k - 1$, *then* $\mathbf{U}^i(k - SAT) \in \mathrm{P}$.

Proof: Let $i \geq 2^k - 1$, and let F be an instance of k-SAT that contains at least one clause $c = (l_1, l_2, \ldots, l_k)$. (If F is empty, it should be clear that identifying whether or not it is in

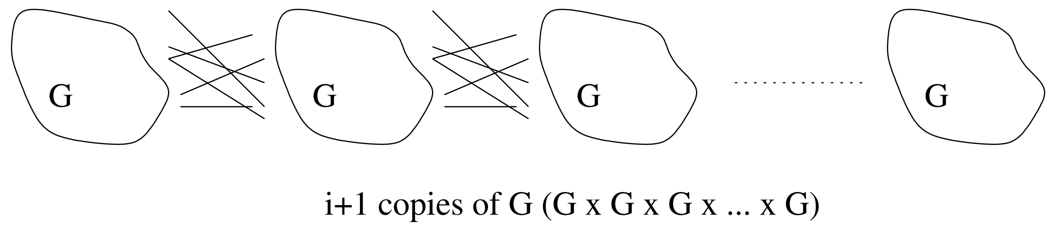

Figure 3.4: Construction used in the reduction $\mathbf{IS} \propto_m \mathbf{U}^i(\mathbf{IS})$

$\mathbf{U}^i(k-\text{SAT})$ can be done in polynomial time). If we now add the other 2^k-1 clauses on the same variables as those in c, the instance becomes unsatisfiable. Thus, to check if a formula F is in $\mathbf{U}^i(k-\text{SAT})$, we need only check whether or not it contains a clause. Clearly, this can be done in polynomial time. ■

3.3 Independent Set

THEOREM **3.3.1** $\mathbf{U}^i(\mathbf{IS}) \in$ NP-complete *for all* $i \geq 0$.

Proof: Proof is via a reduction from **IS**. Given a graph G in which we wish to check the existence of an independent set of size k (a k**-set**), construct a graph G' by making $i+1$ copies of G and inserting all possible edges between the copies (see figure 3.4). If G contains a k**-set**, then G' contains at least $i+1$ independent sets of size k which have no vertices in common, so $G' \in \mathbf{U}^i(k\textbf{-set})$. On the other hand, if G does not have a k**-set**, then neither does G', which implies that $G' \notin \mathbf{U}^i(k\textbf{-set})$. ■

3.4 Hamilton Cycle

THEOREM **3.4.1** $\mathbf{U}^i(\mathbf{HP}) \in$ NP-complete, $\forall i \geq 0$.

Proof: Note: In section 3.1, we defined **HP** such that $G \in \mathbf{HP}$ if G^c has a Hamiltonian path. This made the property monotone down with respect to adding edges in G. In this proof, we will (for clarity's sake) remove edges from the complement of the graph under consideration, as adding edges to G is equivalent to removing edges from G^c.

Since Hamilton Path is NP-complete, there is nothing to prove for $i=0$. Thus, let $i \geq 1$. We already know that $\mathbf{U}^i(\mathbf{HP}) \in$ NP. We will complete the proof via a reduction from **HP**. Let G be any graph, and construct a new graph $G' = (i+1)G^c \times K_{2i}$ (see figure 3.5). Then $G \in \mathbf{HP} \iff G'^c \in \mathbf{U}^i(\mathbf{HP})$.

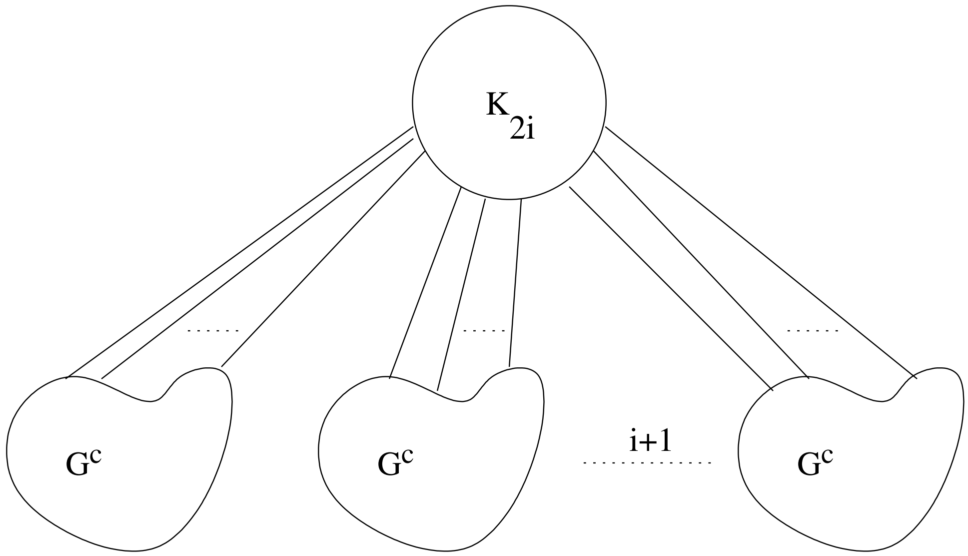

Figure 3.5: Construction used in the reduction $\mathbf{HP} \propto_m \mathbf{U}^i(\mathbf{HP})$

Suppose $G \notin \mathbf{HP}$. Then covering the vertices in each copy of G^c requires at least two disjoint paths. Thus, it takes $\geq 2(i+1)$ disjoint paths to cover the $i+1$ copies of G^c. To create a Hamiltonian path in G', we would need to connect these disjoint paths together by visiting a vertex in the clique in between each subpath. But this would require at least $2i+1$ vertices in the clique. Hence, $G \notin \mathbf{HP} \Longrightarrow G'^c \notin \mathbf{HP} \Longrightarrow G'^c \notin \mathbf{U}^i(\mathbf{HP})$.

Now, let $G \in \mathbf{HP}$. We must show that there is a Hamiltonian path in G' even if an arbitrary set of i edges is first removed. There are three classes of edges that can be removed: edges inside the copies of G^c, edges in the clique, and edges with one endpoint in the clique and one endpoint in a copy of G^c. Call these edge sets E_1, E_2, and E_3 respectively, with $|E_1| = h$, $|E_2| = k$, $|E_3| = j$ and $i = h + j + k$.

Since $G \in \mathbf{HP}$, in the absence of edge deletions we can cover the $i+1$ copies of G^c with $i+1$ disjoint paths. By removing E_1 from G', in the worst case we will need to use $i+h+1$ disjoint paths $\mathcal{P} = \{P_1, P_2, \ldots, P_{i+h+1}\}$. We will construct a path in G' that traverses each of these paths in order, visits a vertex in the $2i$-clique between successive paths, and then finishes by tracing a path in the remaining vertices in the clique. That is, the Hamiltonian path will look like $P_1 - x_1 - P_2 - x_2 - \cdots - P_{i+h} - x_{i+h} - P_{i+h+1} - x_{i+h+1} - x_{i+h+2} - \cdots - x_{2i}$ where the x's are vertices in the clique K_{2i}.

Let V_1 be the set of vertices in the clique K_{2i}. Let $V_2 \subseteq V_1$ be a set of k vertices such that each edge in E_2 has an endpoint in V_2 (i.e., V_2 covers E_2). Also, order the paths in $\mathcal{P}$

so that for $t > j$, neither endpoint of P_t is adjacent to an edge in E_3.

After removing E_3 from G', there are still at least $2i - j \geq j$ vertices in V_1 adjacent to every vertex in the paths. Let V_3 be any j of these vertices. For $1 \leq t \leq j$, connect paths P_t and P_{t+1} together with a vertex in V_3. For $j + 1 \leq t \leq j + |V_2 \setminus V_3|$, connect P_t and P_{t+1} together with a vertex from $V_2 \setminus V_3$; note that $j + |V_2 \setminus V_3| \leq j + |V_2| = j + k \leq i + h$, so we will be able to visit all the vertices in V_2 by the end of this step. At this point, every unvisited vertex in the clique, $V_4 = V_1 \setminus (V_2 \cup V_3)$, is adjacent to every vertex in all the paths, so for $j + |V_2 \setminus V_3| + 1 \leq t \leq i + h$ we can connect P_t and P_{t+1} together with vertices in V_4. Finally, if necessary we connect the free endpoint of P_{i+h+1} with any unvisited vertex in V_4. We can now complete the path by visiting the remaining vertices in V_1 because $K_{2i}[V_4]$ is a clique. ■

Remark: Note that by adding an extra vertex to the clique in the proof, we can modify the proof to obtain a reduction $\mathbf{HP} \propto_m \mathbf{U}^i(\mathbf{HC})$.

Remark: Note that any Hamiltonian path in the graph G' constructed in the proof of theorem 3.4.1 must contain a Hamiltonian path of at least one of the copies of G^c as a subpath. Hence, simply finding an $\mathbf{HP}$ in a graph $G \in \mathbf{U}^i(\mathbf{HP})$ is as hard as finding a Hamiltonian path in an arbitrary graph.

Chapter 4

The Unfrozen Structure of k-Coloring

A quick argument shows that unlike Hamiltonian cycle and independent set, it cannot be the case that $\mathbf{U}^i(k-\mathrm{COL}) \in$ NP-complete for all $i \in \mathbb{N}$. If $i \geq \binom{k+1}{2}$, then for any graph G containing $k+1$ or more vertices, we can construct a K_{k+1} by adding i or fewer edges to G; as a $k+1$-clique is not k-colorable, we see that the only graphs in $\mathbf{U}^i(k-\mathrm{COL})$ for $i \geq \binom{k+1}{2}$ are those with k or fewer vertices.

In this chapter, we will prove in theorem 4.2.1 that $\mathbf{U}(k-\mathrm{COL})$ is NP-complete. Thus, the value of i for which $\mathbf{U}^{i-1}(k-\mathrm{COL}) \notin$ P and $\mathbf{U}^i(k-\mathrm{COL}) \in$ P must be between 2 and $\binom{k+1}{2}$. We prove in section 4.1 that this upper bound can be improved to $\binom{k}{2}$. Although there is still a gap between the i for which $\mathbf{U}^i(k-\mathrm{COL})$ known to be NP-complete and those for which it is definitely in P, we have some intuition as to where the dividing line may be. In section 4.6 we present our conjectures.

4.1 The Polynomial Boundary for Coloring

In what follows, for a graph $G = (V_G, E_G)$, $n_G = |V_G|$ and $m_G = |E_G|$.

Lemma 4.1.1 *For $k \geq 3$, $0 \leq i < \binom{k+1}{2}$ we have the following:*

$$\mathbf{U}^i(k-COL) \subseteq \left\{ G \,\middle|\, |V| \leq k \text{ or } \forall V' \subseteq V, |V'| \leq k+1 \Rightarrow |E(G[V'])| < \binom{k+1}{2} - i \right\}$$

Proof: Let $G \in \mathbf{U}^i(k-\mathrm{COL})$. If $|V| \leq k$, then G can always be colored with k colors. On the other hand, let $|V| \geq k+1$, and take any $V' \subseteq V$, $|V'| = k+1$. Since G is unfrozen, it follows that adding i edges to $G[V']$ cannot make a $k+1$-clique, which means $|E(G[V'])| < \binom{k+1}{2} - i$, as desired. ■

Lemma 4.1.2 *If $G \in \mathbf{U}^{\binom{k}{2}}(k - COL)$, then all connected components of G contain $\leq k$ vertices.*

Proof: Let H a connected subgraph of G on $k+1$ vertices. (Note that any connected graph of size $> k+1$ contains a connected subgraph of size $k+1$.) Since it is connected, H must contain at least k edges. But by lemma 4.1.1, $|E(H)| < \binom{k+1}{2} - \binom{k}{2} = k$. Contradiction. ∎

Theorem **4.1.1** *For $k \geq 3$ we have*
$\mathbf{U}^{\binom{k}{2}}(k - COL) =$
$\{G \mid |V| \leq k \text{ or } \forall V' \subseteq V, |V'| \leq k+1 \Rightarrow |E(G[V'])| \leq k-1\}$

Proof: Lemma 4.1.1 provides the forward inclusion. Now, let

$$G \in \{G \mid |V| \leq k \text{ or } \forall V' \subseteq V, |V'| \leq k+1 \Rightarrow |E(G[V'])| \leq k-1\}$$

If $|V| \leq k$, then $G \in \mathbf{U}^i(k - \mathrm{COL})$ for all $i \geq 0$. Thus, we may assume that $|V| \geq k+1$. If $G \notin \mathbf{U}^{\binom{k}{2}}(k - \mathrm{COL})$, then there exists a set E' of edges, $|E'| \leq \binom{k}{2}$ such that $G' = (V, E \cup E')$ is $k+1$-chromatic. Let $H' \subseteq G'$ be a $k+1$-critical subgraph of G'. Since H' is a critical graph, its minimum degree is at least k, which means that $m_{H'} \geq \frac{n_{H'}k}{2}$. Finally, let $H = G[V_{H'}]$. Note that $n_H = n_{H'}$, and that H is the subgraph of H' whose edges are those that were originally in G. Thus,

$$\begin{aligned} m_H &\geq m_{H'} - \binom{k}{2} \\ &= \frac{n_H k}{2} - \left[\binom{k+1}{2} - k\right] \\ &= \frac{k\left[n_H - (k+1)\right]}{2} + k \end{aligned} \tag{4.1}$$

We will arrive at a contradiction by inductively showing that $m_H < \frac{k[n_H - (k+1)]}{2} + k$ for all subgraphs H of G. By our choice of G, we know this is the case if $|V_H| \leq k+1$.

Now, choose any subgraph H of G with $|V_H| > k+1$.

Let $S \subseteq V_H$, $|S| = k+1$ be such that for all $T \subseteq V_H$, $|T| = k+1$ we have $|E(H[S])| \geq |E(H[T])|$. By our choice of G, $|E(H[S])| \leq k-1$. Furthermore, by lemma 4.1.2, every component in H contains $\leq k$ vertices.

Let $v \in V_H \setminus S$. Since $|S| = k+1$ and $|E(H[S])| \leq k-1$, it follows that there are at least 2 vertices u_1, u_2 in $H[S]$ with degree ≤ 1. For $i = 1, 2$, it follows that v is adjacent to at most one vertex in $S \setminus \{u_i\}$; if not, $T_i = \{v\} \cup (S \setminus \{u_i\})$ is a set of $k+1$ vertices in H such that $|E(H[T_i])| > |E(H[S])|$, which violates the maximality of S. Since v may be adjacent to both of the u_i, v may be adjacent to at most 2 vertices in S.

Note that if $v, w \in V_H \setminus S$ are in different connected components of G, then at most one of the two vertices can be adjacent to more than one vertex in S. If v is adjacent to two vertices $u_1, u_2 \in S$, then u_1, u_2 must both have degree 1 in $H[S]$. Since v, w are in different components, w is not adjacent to u_1. But then $T = \{w\} \cup (S \setminus \{u_1\})$ is a set of $k+1$ vertices with $|E(H[T])| > |E(H[S])|$, which again would violate the maximality of S.

Now, let $\{C'_1, C'_2, \ldots, C'_t\}$ be the connected components of H that contain at least one vertex outside of S. Define C_i as $C'_i[V_H \setminus S]$. Furthermore, we ensure that if there is a vertex $v \in V_H \setminus S$ such that v is adjacent to more than one vertex in S, then $v \in C_1$.

We now have two cases to consider:

Case 1 ($t = 1$; i.e., there is only one component). In this case, $V_H = S \cup V_{C_1}$. Each of the i vertices in C_1 may be adjacent to up to 2 vertices in S. The total number of edges in H is equal to the number of edges in $H[S]$ plus the edges between S and C_1 plus the edges entirely within C_1. However, since the i vertices are in the same component, and since there are at most $k-1$ edges in any component of G, we see that the latter two quantities can not add up to more than $k-1$. Thus, if we add i vertices $\{v_1, v_2, \ldots, v_i\}$ from C_1 to S, we get at most

$$(k-1) + \min\left\{k-1, 2i + \frac{i(i-1)}{2}\right\}$$

edges in H. Thus, $m_H \leq (k-1) + (k-1) = 2k-2$. But if $i \geq 2$,

$$\begin{aligned}
\frac{k\left[n_H - (k+1)\right]}{2} + k &= \frac{k\left[(k+1+i) - (k+1)\right]}{2} + k \\
&= \frac{ki}{2} + k \\
&\geq 2k \\
&> 2k - 2
\end{aligned}$$

This forces $i = 1$. But then $n_H = k+2$, and $(k-1) + (2i) + \frac{i(i-1)}{2} = k+1$ which is less than the required $\frac{k}{2} + k$ edges as $k \geq 3$.

Case 2 ($t \geq 2$) Let $i = |V(C_t)|$. By induction, $H[V_H \setminus V_{C_t}]$ contains fewer than

$$\frac{k\left[n_H - (k+1) - i\right]}{2} + k$$

edges. Since no vertex in C_t is adjacent to more than one vertex in S, and since there are no edges between the C's, adding the i vertices in C_t will give us no more than $\min\left\{i + \frac{i(i-1)}{2}, k-1\right\}$ extra edges. If $i = 1$, then we get at most one more edge, so

$$m_H < \frac{k\left[n_H - (k+1) - 1\right]}{2} + k + 1$$

$$
\begin{aligned}
&= \frac{k\left[n_H-(k+1)\right]}{2}+\frac{k}{2}+1 \\
&< \frac{k\left[n_H-(k+1)\right]}{2}+k \quad \text{since } k \geq 3
\end{aligned}
$$

This contradicts equation 4.1, so $i > 1$. On the other hand, if $i \geq 2$, then the addition of C_t must add strictly more than

$$
\frac{k\left[n_H-(k+1)\right]}{2}+k-\left(\frac{k\left[n_H-(k+1)-i\right]}{2}+k\right) \quad = \quad \frac{ki}{2}
$$

edges to the current graph. But for $i \geq 2$, $\frac{ki}{2} \geq k$ which is more than the $k-1$ edges C_t can provide.

This case analysis shows that H cannot contain enough edges to be a $k+1$ chromatic graph, even if we are allowed to add $\binom{k}{2}$ edges to it. ■

4.1.1 Implications

Theorem 4.1.1 provides a polynomial time algorithm to identify graphs in $\mathbf{U}^{\binom{k}{2}}(k-\mathrm{COL})$ (for fixed k). A graph G is in this class if and only if every subset of $k+1$ vertices in G induces fewer than k edges in G. This algorithm requires $O(\binom{k}{2}\binom{n}{k+1}) = O(k^2 n^{k+1})$ time. Since for any monotone graph property X we have $X \in \mathrm{P} \implies \mathbf{U}(X) \in \mathrm{P}$, this means $\mathbf{U}^{\binom{k+1}{2}-z}(k-\mathrm{COL}) \in \mathrm{P}$ for $z \leq k$.

A dynamic programming approach may be used to solve the problem in $O(n)$ time (for fixed k). Such an algorithm is given in figure 4.1. The algorithm looks at all subgraphs of each component C of G, and determines the maximum number of edges $\alpha[i]$ induced by i vertices from C, where $i = 0, \ldots, |C|$. Note that $|C| \leq k$, so there are no more than 2^{k+1} subgraphs to consider. If $A[j]$ is the maximum number of edges induced in G using j vertices (without using any vertices from C), then the maximum number if we include C is:

$$
\max_{0 \leq i \leq j} \{A[j-i]+\alpha[i]\}
$$

Finding all the components of G can be done in $O(n)$ time. Since the sum of the sizes of all the components of G is n, it follows that line 9 is reached exactly n times. The **for** loop on line 10 computes $O(2^k)$ subgraphs of C, and counting the edges requires $O(k^2)$ time. Hence, the loop takes $O(k^2 2^k)$ time and is reached n times for an overall complexity of $O(k^2 2^k n)$. Although this is an overestimate, a more detailed analysis arrives at the same O complexity. Over all components, line 16 is executed $O(nk)$ times. Thus, the algorithm takes $O(n + k^2 2^k n + kn) = O(k^2 2^k)$ time.

```
/* A[i] will be the maximum number of */
/* edges induced by i vertices */
for i = 1 to k + 1 do
    A[i] ← 0
endif
for each component C of G do
    /* A graph in U^(k choose 2)(k − COL) has no */
    /* component with more than k vertices in it */
    if |C| > k then
        return FALSE
    else
        for i = 1 to |C| do
            /* α[i] is the maximum number of */
            /* edges induced by i vertices from C */
            α[i] ← 0
            for each U ⊆ C, |U| = i do
                α[i] ← max {|E(G[U])|, α[i]}
            end
        end

        /* Update A */
        for i = k + 1 downto 1 do
            for j = 1 to |C| do
                A[i] = max {A[i], A[i − j] + α[j]}
            end
        end
    endif
end
if A[k + 1] ≥ k then
    return FALSE
else
    return TRUE
endif
```

Figure 4.1: An $O(k^2 2^k n)$ algorithm for solving $\mathbf{U}^{\binom{k}{2}}(k-\mathrm{COL})$

For $k+1 \leq z \leq \binom{k+1}{2}$, there exist graphs which pass the subset test, yet are not in $\mathbf{U}^{\binom{k+1}{2}-z}(k-\text{COL})$. To see this, by a result of Erdös ([16]), we know there exists a $k+1$ chromatic graph G with girth larger than $k+1$. Then every subgraph of G containing $k+1$ or fewer vertices certainly has $\leq k < z$ edges, yet $G \notin \mathbf{U}^{f(k,z)}(k-\text{COL})$. This observation leads to the conjectures stated in section 4.6.

4.2 $\mathbf{U}(k - \mathbf{COL})$ is NP-complete

THEOREM **4.2.1** $\mathbf{U}(k - COL) \in$ NP-complete *for all* $k \geq 3$.

The theorem will be proved over the next several sections. We will do so via a conversion from NAE-3SAT (see section 3.2) to $3-\text{COL}$. Given an instance $Q = (V, C)$ of NAE-3SAT, we will construct a graph G such that Q has a satisfying assignment of its variables V if and only if G is 3-colorable. We will then complete the proof by showing that G is 3-colorable if and only if $G + e$ is 3-colorable for any $e \in E(G^c)$; i.e., G is 3-colorable if and only if $G \in \mathbf{U}(3-\text{COL})$. Finally, in section 4.5, we will show how the result can be extended to show that $\mathbf{U}(k-\text{COL}) \in$ NP-complete for $k > 3$.

4.3 Construction of the $\mathbf{U}(k - \mathbf{COL})$ reduction graph G

Given an instance $Q = (V, C)$ of NAE-3SAT, we will construct a graph G such that every valid 3-coloring of G corresponds to a satisfying assignment of Q. G will be constructed from many subgraphs called *gadgets*. The gadgets will represent the literals and clauses in Q. A valid 3-coloring of G will (via the gadgets):

1. Assign truth value to every literal in Q (section 4.3.2)
2. Ensure each variable and its negation are assigned different values (section 4.3.3).
3. Ensure that for every clause not all of the literals are assigned the same value (sections 4.3.5,4.3.4).

Furthermore, a satisfying assignment to Q can be used to construct a valid 3-coloring of G.

4.3.1 Notation

The colors used to 3-color the graph will be $\{1, 2, 3\}$. A single special vertex $\boxed{\mathbf{3}}$ is assumed to be given color 3. If the other vertices in a gadget are given labels for reference purposes,

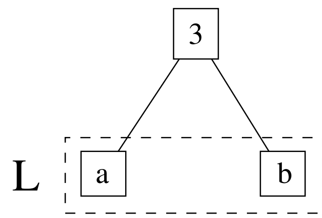

Figure 4.2: A literal ***L***

those labels will be alphabetic.

4.3.2 Literals

A literal L in Q is represented in G as two independent vertices joined to the distinguished vertex $\boxed{\mathbf{3}}$; see figure 4.2. Rather than calling the two possible states of L *true* (T) and *false* (F), we will instead denote them by *same* (**S**) and *different* (**D**). In a 3-coloring of G, L has *value* **S** if the independent vertices a and b are assigned the same color, and *value* **D** if they are assigned different colors. Since a and b are adjacent to $\boxed{\mathbf{3}}$, only colors 1 and 2 can be used to color L.

Vertex a	Vertex b	Literal Value
1	1	**S**
1	2	**D**
2	1	**D**
2	2	**S**

Table 4.1: Value of a literal gadget based upon how it is colored.

As ordered pairs, the colorings $(1,1)$ and $(2,2)$ both give L value **S**, and $(1,2)$ and $(2,1)$ both give L value **D** (see table 4.1). Unless specified otherwise, when we say that a literal has a certain value, we do not force any particular one of the two possible colorings of the literal. Thus, in the following sections, we must show that if the literals attached to a gadget have appropriate values, then *any* coloring of the literals that encodes those values can be extended to a coloring of the entire gadget.

In the graphs that follow, a literal gadget will be denoted by two independent vertices surrounded by a box. The vertex $\boxed{\mathbf{3}}$ will normally not be shown; it will be implicitly understood that the independent vertices in the literal can only be colored with 1 or 2. Note that for every variable x in Q, both x and $\overline{x}$ will appear as literal gadgets in G.

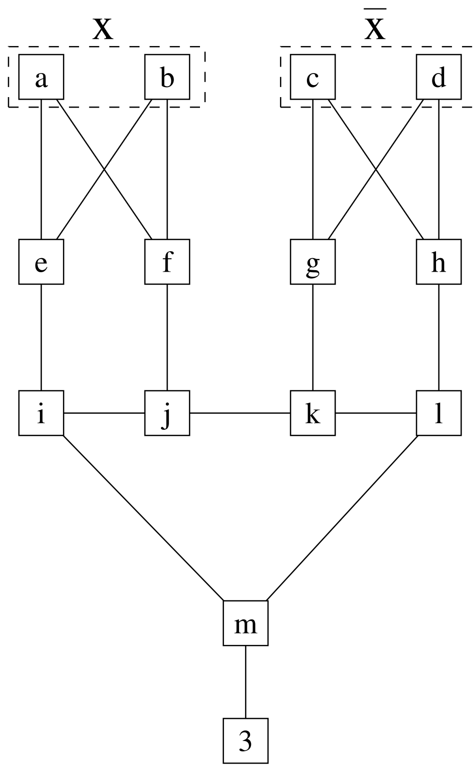

Figure 4.3: $\boldsymbol{x}$ and $\overline{\boldsymbol{x}}$ cannot both have value **D**

4.3.3 A variable and its negation

The gadgets in figures 4.3 & 4.4 are used to ensure that a variable x and its negation $\overline{x}$ are given different values in any valid 3-coloring of G.

Figure 4.3 ensures that G cannot be 3-colored if both x and $\overline{x}$ have value **D**. If they do, then we see that all the vertices e, f, g, h are forced to have color 3. But then the 5-cycle $ijklm$ cannot be colored, since three colors are required but only two are available. Table 4.2 shows that the gadget *can* be colored provided the two literals have different values. By symmetry, we need only consider the case where $x = \mathbf{S}$, $\overline{x} = \mathbf{D}$.

Literal x		Literal $\overline{x}$		Valid coloring for gadget								
a	b	c	d	e	f	g	h	i	j	k	l	m
1	1	1	2	2	3	3	3	3	1	2	1	2
1	1	2	1	2	3	3	3	3	1	2	1	2
2	2	1	2	1	3	3	3	3	1	2	1	2
2	2	2	1	1	3	3	3	3	1	2	1	2

Table 4.2: Valid colorings of figure 4.3

On the other hand, figure 4.4 ensures that x and $\overline{x}$ do not both take value **S**. If they

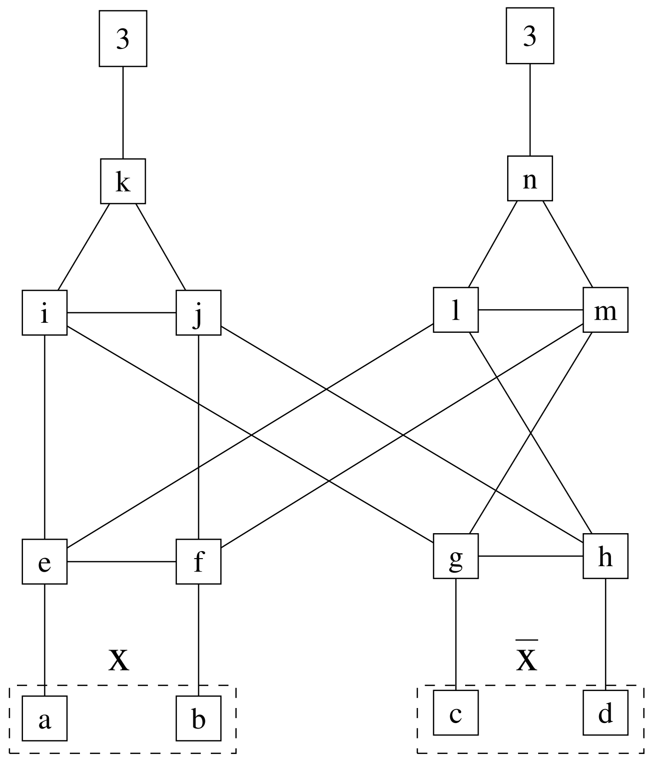

Figure 4.4: $\boldsymbol{x}$ and $\overline{\boldsymbol{x}}$ cannot both have value **S**

do both have value **S**, then two of the vertices in the set $\{e, f, g, h\}$ are forced to be colored with color 3. But then one of the triangles ijk or lmn cannot be colored, as all three vertices will be adjacent to a color 3 vertex. Provided x and $\overline{x}$ have different values, then the gadget can be colored with three colors, as table 4.3 shows.

Literal x		Literal $\overline{x}$		Valid coloring for gadget									
a	b	c	d	e	f	g	h	i	j	k	l	m	n
1	1	1	2	2	3	2	1	3	2	1	3	1	2
1	1	2	1	2	3	1	2	3	1	2	3	2	1
2	2	1	2	1	3	2	1	3	1	2	3	2	1
2	2	2	1	1	3	1	2	3	2	1	3	1	2

Table 4.3: Valid colorings of figure 4.4

4.3.4 Literals in a clause do not all have value S

Given a clause (L_1, L_2, L_3) in Q, we prevent all three literals from taking the value **S** by using the gadget shown in figure 4.6. In this figure, each literal is adjacent to two vertices connected by an edge. Then, for every possible choice of picking one vertex from each vertex pair connected to a literal, we attach it to a “Not-All-3” gadget of the type shown in

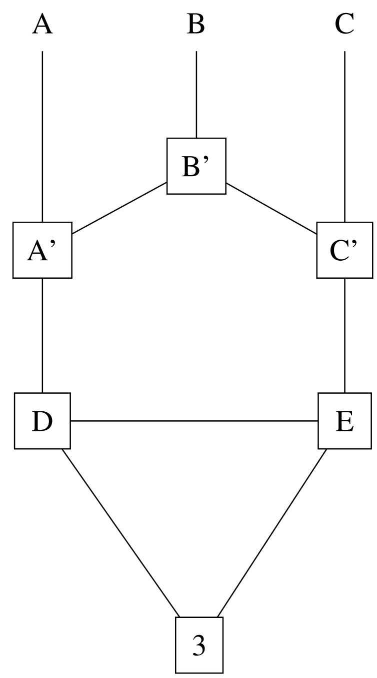

Figure 4.5: Not-All-3 Gadget. Not all of A, B, C can be colored 3

figure 4.5. That is, we attach the following 8 subsets of vertices to the Not-All-3 gadgets: $\{ace, acf, ade, adf, bce, bcf, bde, bdf\}$

In the figure of the Not-All-3 gadget, A, B, and C are vertices outside the gadget. One Not-All-3 gadget will be attached to ace in figure 4.6, one will be attached to acf, and so on. We see that figure 4.5 can be 3-colored if and only if at least one of the vertices A, B, or C is not given color 3.

If a literal has value **S**, then one of the two vertices connected to the literal is forced to be colored 3. If all literals have value **S**, then one of the Not-All-3 gadgets will not be colorable. Provided at least one literal has value **D**, we can assign its adjacent vertices colors 1 and 2, which ensures every Not-All-3 gadget is colorable. Hence, the clause gadget in figure 4.6 is colorable if and only if at least one literal has value **D**.

4.3.5 Literals in a clause do not all have value D

To prevent all the literals in a clause from having value **D**, we use the gadget shown in figure 4.7. We see that if every literal is colored with two different colors, then every vertex in the 7-cycle is adjacent to a vertex of color 3. But then we only have two colors available (1 and 2) to color the cycle, which is impossible. Provided at least one literal has value **S**, we can set at least one of the vertices outside the cycle to either 1 or 2, which allows us to

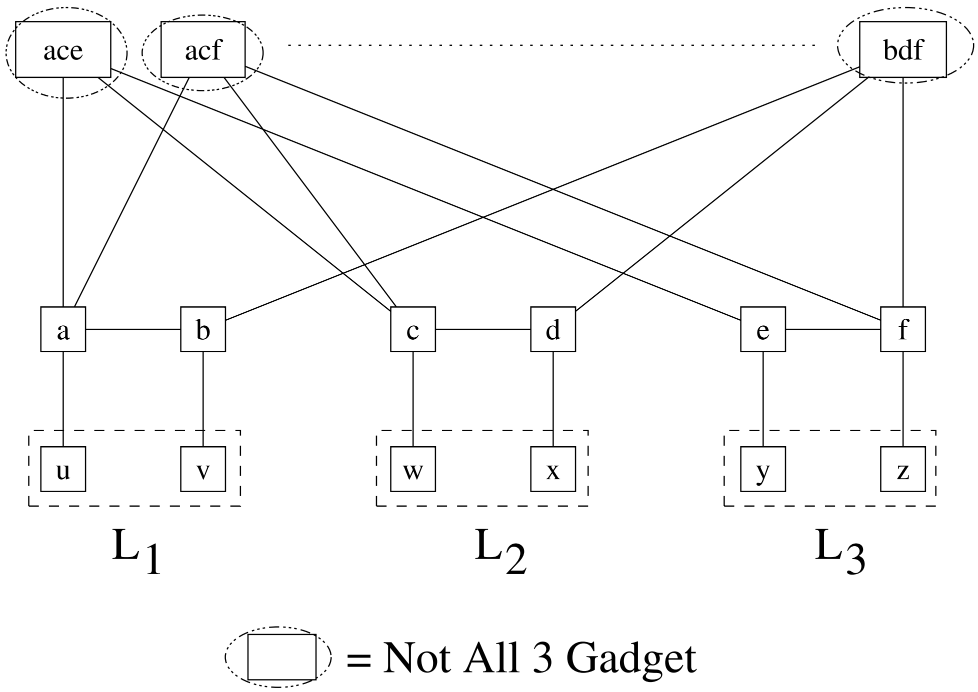

Figure 4.6: All three literals $\boldsymbol{L_1, L_2, L_3}$ cannot have value *same*. Note that there are 8 Not-All-3 gadgets: $\boldsymbol{\{ace, acf, ade, adf, bce, bcf, bde, bdf\}}$

color the 7-cycle.

Hence, this gadget is 3-colorable if and only if not all the literals in the clause have value **D**.

4.3.6 Summary

Given an instance $Q = (V, C)$ of NAE-3SAT, we create a graph G by constructing a literal gadget (fig. 4.2) for each variable and its negation in V. We then attach complementary literals to the gadgets given in figures 4.4 & 4.3; this ensures they take on different values. Then, for every clause $c = (L_1, L_2, L_3) \in C$, we attach its literals to the gadgets in figures 4.6 & 4.7. These two gadgets force there to be literals with differing values in each clause. Altogether, we see G is 3-colorable if and only if Q has a satisfying assignment. Furthermore, given a satisfying assignment to Q we can find a valid coloring for G, and vice versa.

4.4 The colorability of G is unaffected by edge addition

We notice that provided the literal gadgets are colored in a way that corresponds to a satisfying assignment of the underlying NAE-3SAT instance Q, then we are guaranteed to

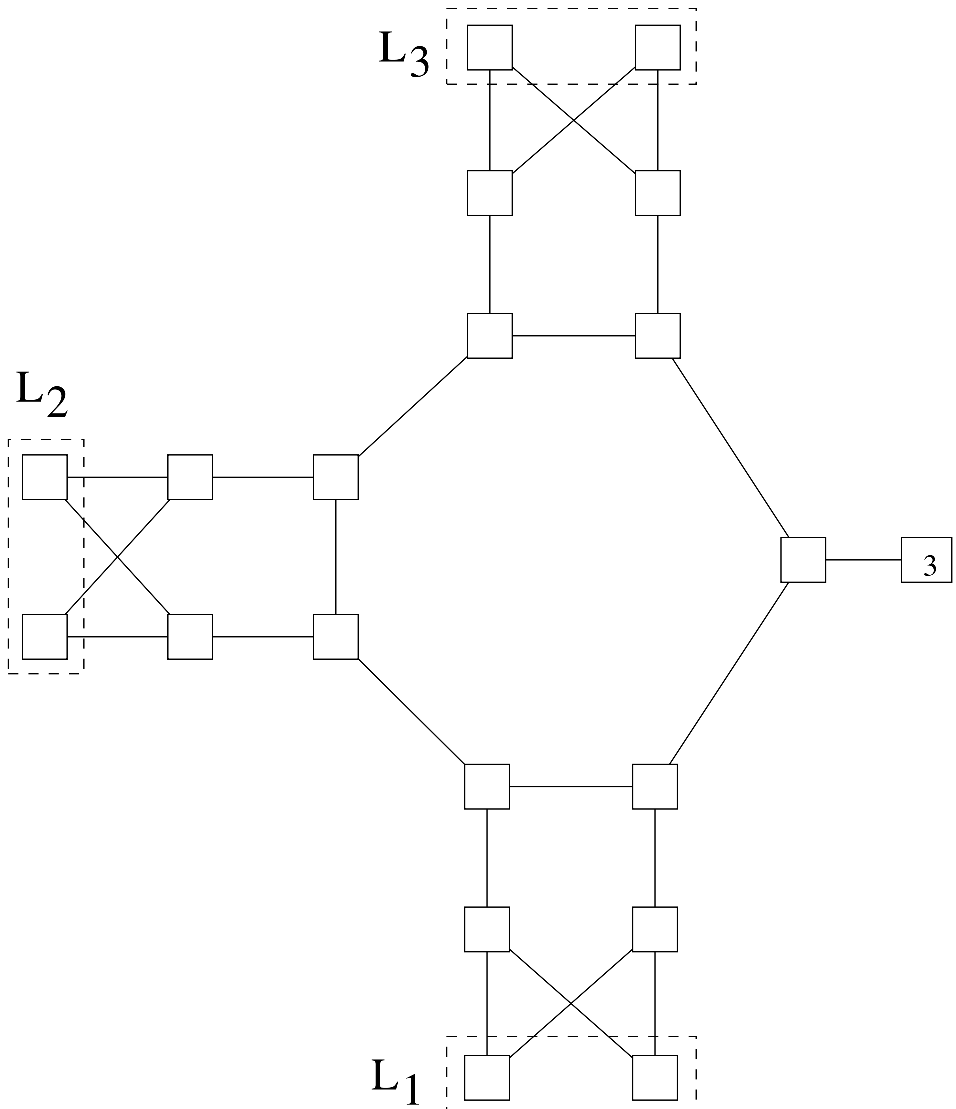

Figure 4.7: All three literals $\boldsymbol{L_1, L_2, L_3}$ cannot have value *different*

be able to 3-color all the gadgets in G. Furthermore, the gadgets are not affected by the particular choice of encoding. That is, if literal L is supposed to have value same (**S**), then we can color its vertices as either $(1, 1)$ or $(2, 2)$. If G could be colored when L was encoded as $(1, 1)$, then it can also be colored when it is encoded as $(2, 2)$.

Additionally, if Q is a satisfiable NAE-3SAT instance, then we can invert the values of every literal (that is, exchange **S** and **D**) in G, and still have a colorable graph. Also, once the literals adjacent to a given gadget are colored, the gadget itself can be colored *independently* of every other gadget in G.

These facts give G a certain amount of flexibility to remain 3-colorable even when an arbitrary edge is added to the graph. Our basic strategy to fix problems introduced by a new edge can be summarized as follows:

1. If necessary, invert the values on the literals.
2. If necessary, change the encoding on a few literals: $(1, 1) \leftrightarrow (2, 2)$, $(1, 2) \leftrightarrow (2, 1)$.
3. Use the fact that gadgets can be independently colored to ensure that one endpoint of the new edge has a choice of two possible colors *independent* of the color of the other endpoint.

Item 3 tells us that for a new edge between gadgets, we will be able to color the vertices on either end of the edge different colors. For edges within gadgets, we will exhibit colorings of those gadgets that show every fixed pair of vertices within that gadget can be given different colors.

4.4.1 Edges connected to literals

If an edge is added between vertices a and b in a literal gadget (fig. 4.2), then the literal is forced to have value **D**. Flip the value of all literals if needed to fix this problem.

If an edge is added between two literal gadgets, change the colors on one of the literals (if needed) between $(1, 1) \leftrightarrow (2, 2)$ or $(1, 2) \leftrightarrow (2, 1)$ as needed to make the edge irrelevant.

Finally, if an edge is added between either of the vertices in a literal and a vertex in any other gadget, fix the problem in the other gadget. (See the next sections).

4.4.2 Edges to the gadgets that enforce $x \neq \overline{x}$

Consider an edge with exactly one endpoint inside the gadget (fig. 4.3) that forces at least one of x and $\overline{x}$ to have value **S**. In what follows, we can set the value of a particular literal by flipping the value of all literals. We do not consider the vertices in the literals to be part of the gadget.

	$x = \mathbf{S}$		$\overline{x} = \mathbf{D}$		Valid coloring of gadget								
	a	b	c	d	e	f	g	h	i	j	k	l	m
α	1	1	1	2	2	3	3	3	3	1	2	1	2
β	1	1	1	2	3	2	3	3	2	3	1	2	1

Table 4.4: Different colorings of the gadget in fig. 4.3 given that the literals have certain fixed values.

By comparing colorings α and β in table 4.4, we see that every vertex in the gadget except for g and h can be given one of two different colors independently of any other gadget. (If we wish to give g and h the same option, we just flip the values of all literals so that $\overline{x} = \mathbf{S}$, $x = \mathbf{D}$.) Note that the colorings on the literals are *fixed*, so connections between the gadget and any literal vertex in G can also be repaired.

If both endpoints are within the gadget (fig. 4.3), we notice that coloring α in table 4.4 allows any edge within the gadget except for those in the set $\{ek, em, fg, fh, fi, gh, gi, hi, jl, km\}$. Of these 10, all but $\{fi, gh, km\}$ have different colored endpoints under coloring β. These last three edges can be accommodated by swapping the values of x and $\overline{x}$ (and every other literal as well), and then using a symmetry argument. (For instance, by symmetry fi is the same as gl, which means that edge fi can be accommodated by setting $x = \mathbf{D}$, $\overline{x} = \mathbf{S}$).

Next, consider adding an edge to G which has exactly one endpoint in the gadget (fig. 4.4) that ensures at least one of the literals x and $\overline{x}$ has value $\mathbf{D}$. As in the previous case, we will *fix* the colorings on the two literals by flipping the entire solution and recoloring x and $\overline{x}$ as required. We will set $x = (1, 1) = \mathbf{S}$, $\overline{x} = (1, 2) = \mathbf{D}$.

	$x = \mathbf{S}$		$\overline{x} = \mathbf{D}$		Valid coloring of gadget									
	a	b	c	d	e	f	g	h	i	j	k	l	m	n
α	1	1	1	2	2	3	2	1	3	2	1	3	1	2
β	1	1	1	2	3	2	2	1	1	3	2	2	3	1

Table 4.5: Different colorings of the not both same gadget in fig. 4.4 given that the literals have certain fixed values.

In table 4.5, we see that every vertex except for g and h can be given one of two different colors independently of vertices in any other gadget once the literal colorings have been set. If we have an edge connected to g or h, we just flip the values on x and $\overline{x}$ and use symmetry to show that g and h can be given different values.

As for an edge with both endpoints in the gadget, we notice that coloring α in table 4.5 allows any edge within the gadget except for those in the set $\{ej, eg, en, fl, gn, gj, hk, hm, if, il, jn, km\}$. Of these 12, all but ej and fl have different colored endpoints under coloring β. These last two edges can be accommodated by swapping the values of x and $\overline{x}$ (and every other literal as well).

Hence, if G is colorable, then $G + e$ is also 3-colorable for any edge e with at least one endpoint in a gadget of the types given in figures 4.4 and 4.3.

4.4.3 Edges attached to clause gadgets

Not-All-D-Gadget

Consider the clause gadget shown in figure 4.7 which ensures that not all of the literals in a clause (L_1, L_2, L_3) can have value *different* (**D**). We will first consider the problem of adding edges *within* the gadget; as usual, the vertices in the literals do not count as part of the gadget.

Note that if the gadget can be colored, then it is always possible to color all but one of the vertices (call the exception "v") outside the 7-cycle with color 3. In order for vertex v to have a color other than 3, it will need to be adjacent to a literal with value **S**; any of the six vertices outside the cycle that are not the $\boxed{\mathbf{3}}$ vertex can be v by flipping the values of all the literals (if necessary).

Case I (Edge with both endpoints inside the 7-cycle): Color one of the endpoints 3; this can always be done by ensuring that it is adjacent to the vertex v mentioned above. 2-color the rest of the cycle with colors 1 and 2. Since the edge is incident on a vertex of color 3 and a vertex of color 1 or 2, it is not violated.

Case II (Edge with one point in the cycle, one outside) Ensure that the endpoint outside the cycle is *not* v. Then there exists a way to give the endpoint in the cycle color 1, and the endpoint outside the cycle (which is adjacent to one of the literals) color 3.

Case III (Edge with both endpoints outside the cycle) Let one of the endpoints be the special vertex v. Then that vertex will have color 1 or 2, while the other endpoint has color 3.

Case IV (Edge with one endpoint outside the gadget, one inside the 7-cycle). Provided the vertex in the cycle is not adjacent to the vertex v, we can give the cycle vertex either color 1 or 2 independently of the other endpoint. Thus, the graph can be colored so that the edge is not violated.

Case V (Edge with one endpoint outside the gadget, one in the gadget but outside the cycle) Notice that provided the vertex in the gadget is adjacent to a literal with value **S**, it can be given one of two colors: 3 and one other which depends on how the literal is colored. Thus, the gadget can be colored independently of the other endpoint in a way that ensures the new edge is not violated.

Not-All-S-Gadget

We will first investigate the properties of the Not-All-3 gadget shown in figure 4.5. Remember that our full clause gadget (figure 4.6) contains eight different Not-All-3 gadgets. Each of the Not-All-3 gadgets is attached at its connection points (A, B, C) to (in turn) one of the two vertices adjacent to each of the three literals L_1, L_2, L_3. Note that if the connection vertices are colored $A = 1, B = 2, C = 3$, then the 5-cycle within the gadget can be colored in two different ways:

Coloring of cycle given $A = 1, B = 2, C = 3$				
A'	B'	C'	D	E
3	1	2	2	1
2	3	1	1	2

Table 4.6: Valid colorings of the Not-All-3 gadget (fig. 4.5)

Notice that once the literal values have been fixed, any vertex in the Not-All-3 gadget can be given one of two colors independently of any other gadget. Thus, if we add an edge with one edge in the gadget and one outside it, we can still color G so that the edge is not violated. As for edges entirely within a Not-All-3 gadget, we see that the two colorings given above allow us to accommodate any new edge except for $C'D$. This edge can be dealt with by choosing the connection points to be colored as $A = 3, B = 2, C = 1$.

Note that it is always possible to set the connections on a Not-All-3 gadget to $A = 1, B = 2, C = 3$. C is adjacent to literal L_3; thus, set L_3 to value **S** by flipping the values of all literals (if necessary). Then, we can recolor literals L_1 and L_2 (while still keeping their values (**S** or **D**) constant) so that A and B are colored 1 and 2 respectively. From previous results, we know that these recolorings will still allow all other gadgets in G to be 3-colored.

For edges with at least one endpoint in the clause gadget (figure 4.6) that is not also in one of the Not-All-3 gadgets (fig. 4.5), we merely need to ensure that one endpoint is incident on one of the vertices next to the literal with value **S**. In this case, the vertex can be given either color 3 or one of the set $\{1, 2\}$. This can be done independently of any other gadget, except for the eight Not-All-3 gadgets shown in the diagram. This technique also fixes the case where an edge is added with both endpoints inside the gadget. If the edge is connected to one of the eight Not-All-3 gadgets, we use the techniques shown above to resolve the problem.

4.5 Extending the Result to $\mathbf{U}(k - \mathrm{COL})$

For $k > 3$, we first construct the graph G as shown in section 4.3. We then construct the graph $G' = G \times K_{k-3}$, where K_{k-3} is a clique on $k-3$ vertices. It is easy to see that G' is k-colorable if and only if G is 3-colorable. Furthermore, since the only edges that can be added to G' are between vertices in the copy of G, we see that $G' \in \mathbf{U}(k - \mathrm{COL})$ if and only if $G \in \mathbf{U}(3 - \mathrm{COL})$. Hence, it follows that $\mathbf{U}(k - \mathrm{COL}) \in$ NP-complete.

4.6 Probing the Polynomial Boundary of Coloring

By theorems 4.2.1 and 4.1.1 we know that $\mathbf{U}^i(k-\mathrm{COL})$ is NP-complete for $i \leq 1$, and is in P for $i \geq \binom{k+1}{2} - k = \binom{k}{2}$. Thus, unless P = NP, for each $k \geq 3$ there exists an i_k, $2 \leq i_k \leq \binom{k}{2}$ such that $\mathbf{U}^{i_k}(k - \mathrm{COL}) \in$ P but $\mathbf{U}^{i_k - 1}(k - \mathrm{COL}) \notin$ P. Determining i_k appears to be very difficult. In this section, we will conjecture likely values, and explain why we think these values are likely.

To motivate our first conjecture, we will first need to prove the following result. It is known that arity-3 hypergraph 2-coloring is NP-complete(see [19]). We now show that this problem remains NP-complete even if we restrict our attention to the instances where the hypergraph remains colorable after any two vertices are arbitrarily assigned colors; that is, $\widehat{\mathbf{U}}^2(\mathbf{H3K2}) \in$ NP-complete.

THEOREM **4.6.1** $\widehat{\mathbf{U}}^2(\mathbf{H3K2})$ *is* NP-complete.

Proof: We will begin by constructing a many-to-one reduction $3 - \mathrm{COL} \propto_m \mathbf{H3K2}$. Afterwards, we will show that if the constructed hypergraph is 2-colorable, then it also has the property that an arbitrary coloring of any two vertices can be extended to a valid 2-coloring.

Let $G = (V, E)$ be the graph in an arbitrary instance of 3-COL. We wish to find a function $c : V \to \{1, 2, 3\}$ such that if $vw \in E$ then $c(v) \neq c(w)$. For each $v \in V$ we create three vertices v_1, v_2, v_3 in our hypergraph, and add the hyperedge (v_1, v_2, v_3).

Recall that in **H3K2**, every vertex is colored either 1 or 2, and that the hypergraph is properly colored if and only if every hyperedge is not monochromatic. Thus, given a valid

v_2	v_3	x_v	y	x_w	w_2	w_3
1	1	2	2	1	1	2
					2	1
1	2	1	2	1	1	2
					2	1

Table 4.7: Ways to color w_2 and w_3 given certain colorings on v_2 and v_3 such that the hyperedges in equation 4.2 can be properly colored. By inverting the colors, we get all colorings of v_2 and v_3. By symmetry, we can swap the vs and ws.

coloring of the hyperedge (v_1, v_2, v_3), we can unambiguously define the coloring function c by

$$c(v) = \begin{cases} 1 & \text{if } f(v_2) = f(v_3) \\ 2 & \text{if } f(v_1) = f(v_3) \\ 3 & \text{if } f(v_1) = f(v_2) \end{cases}$$

where f is the two-coloring of our hypergraph.

For each edge $vw \in E$, we need to ensure $c(v) \neq c(w)$. We will show how to enforce the constraint $(c(v) \neq 1 \text{ OR } c(w) \neq 1)$; the other two colors are handled similarly. First, we add three new vertices to the graph: x_v, x_w, and y. Then the five hyperedges in equation 4.2 are inserted.

$$\begin{matrix} (x_v, v_2, v_3) & & (x_w, w_2, w_3) \\ & (x_v, y, x_w) & \\ (y, v_2, v_3) & & (y, w_2, w_3) \end{matrix} \tag{4.2}$$

If the hypergraph is properly colored by f, then these hyperedges enforce the condition $f(v_2) = f(v_3) \Longrightarrow f(w_2) \neq f(w_3)$. To see this, suppose $f(v_2) = f(v_3) = 1$. Then this forces $f(x_v) = f(y) = 2$, which in turn forces $f(x_w) = 1$. But since $f(x_w) \neq f(y)$, it follows that $f(w_2) \neq f(w_3)$. By inverting the colors, we find the same condition holds if $f(v_2) = f(v_3) = 2$. By negating both sides of the implication, we also have $f(w_2) = f(w_3) \Longrightarrow f(v_2) \neq f(v_3)$.

On the other hand, table 4.7 shows that provided either $c(v) \neq 1$ or $c(w) \neq 1$, the vs and ws may be colored arbitrarily. Thus, the value of $c(v)$ does not force which particular colors to use; $c(v) = 1$ can be encoded as $f(v_2) = f(v_3) = 1$ or $f(v_2) = f(v_3) = 2$, and our choice does not affect how we encode $c(w)$ for any other vertex w.

Thus, the original graph is 3-colorable if and only if the constructed hypergraph is 2-colorable. We have shown that $3 - \text{COL} \propto_m$ **H3K2**. Now, we show that $3 - \text{COL} \propto_m$ $\widehat{\mathbf{U}}^2$(**H3K2**) by proving that if the hypergraph is 2-colorable, then it can still be 2-colored even if we arbitrarily precolor two of its vertices.

Assume the hypergraph is colorable; hence, there exists a valid coloring c for G. By the symmetry of 3-coloring, we can permute the three color classes induced by c to arrive at

v_2	v_3	x_v	y	x_w	w_2	w_3
2	2	$\boxed{1}$	$\boxed{1}$	$\overline{2}$	1	2
1	2				2	1
2	1					
1	2	$\boxed{1}$	$\boxed{2}$	$\overline{1}$	1	2
2	1				2	1
1	2	$\boxed{1}$	$\overline{2}$	$\boxed{1}$	1	2
2	1				2	1
1	2	$\boxed{1}$	1	$\boxed{2}$	1	2
2	1				2	1
2	2					
1	2	$\boxed{1}$	2	$\boxed{2}$	1	2
2	1				2	1
					1	1

Table 4.8: Effects of precoloring two vertices. $\boxed{1}$, $\boxed{2}$ indicate precolored vertices; $\overline{1}, \overline{2}$ are colorings forced by the precolorings. By symmetry, one can derive the situation that occurs if y and x_w are precolored. By switching 1s and 2s we get all possible precolorings.

five other proper colorings of G. Thus, we need only show that if we precolor two vertices, then either we can slightly modify the encoding of a few vertices and use the original value for c, or we can accomodate the precolored vertices by permuting the color classes.

If we choose to precolor two vertices in the hypergraph that correspond to a single vertex v in G (i.e., v_1, v_2, or v_3), then at worst this forces a particular value for $c(v)$. This can be overcome by appropriately permuting the color classes. If we precolor vertices corresponding to two different vertices v, w in G (say v_i and w_j), then we can color the remaining vs and ws in such a way that they encode the original values of $c(v)$ and $c(w)$.

Now, suppose we precolor two of the three vertices x_v, y, x_w corresponding to the edge vw in G; see table 4.8. We see that it is always possible to color both vertex pairs (v_2, v_3) and (w_2, w_3) with the colors $(1, 2)$. Then, by coloring v_1 and w_1 appropriately, we can ensure that $c(v) \neq c(w)$. If the colors assigned are different than the original coloring of G, permute the color classes.

Finally, suppose we precolor one of the three vertices x_v, y, x_w corresponding to an edge vw in G as in table 4.9. Note that if no other vertex is precolored, there exists a way to encode any valid coloring of the edge vw; i.e., $\{c(v) = 1, c(w) = 2\}$, $\{c(v) = 2, c(w) = 1\}$, $\{c(v) = 2, c(w) = 3\}$, and so on. What happens now depends on the second vertex we precolor:

Case I (u_i, $i \in \{1, 2, 3\}$ for some u in G): If $u = v$ or $u = w$, we see from table 4.9 that since the pairs (v_2, v_3) and (w_2, w_3) can in every case be freely colored $(1, 2)$ or $(2, 1)$,

v_2	v_3	x_v	y	x_w	w_2	w_3
1	2	$\boxed{1}$	1	2	1	2
2	1				2	1
2	2					
1	2	$\boxed{1}$	2	2	1	1
2	1				1	2
					2	1
1	2	1	$\boxed{1}$	2	1	2
2	1				2	1
2	2					

Table 4.9: Effects of precoloring one vertex. $\boxed{1}$, $\boxed{2}$ indicate precolored vertices. By symmetry, one can derive the situation that occurs if x_w is precolored. By switching 1s and 2s we get all possible precolorings.

we can accomodate u_i and still ensure that $c(v) \neq c(w)$. If u is not v or w, then coloring u_i has no effect. We will be able to use the same c values for u, v, and w, with perhaps small changes in how the values are encoded.

Case II (One of the vertices x_t, y', x_u corresponding to some edge tu in G): If tu and vw have different endpoints, then we can use the same coloring c for G, with perhaps a few changes in the colorings of the t_i, u_i, v_i, w_i ($i \in \{1,2,3\}$) used to encode c.

If $u = v$, $t \neq w$, we again have two cases: either both precolored vertices are influencing the same pair of vertices that represent $v = u$ (say (v_2, v_3)), or they influence different pairs (say (v_1, v_2) for the first precolored vertex and (v_2, v_3) for the second). In either case, we can always assume $c(v) = 2$ (permute color classes if necessary) and encode it as $f(v_1) = 2, f(v_2) = 1, f(v_3) = 2$, since by table 4.9 both colorings $(1,2)$ and $(2,1)$ are available for (v_i, v_j). If our coloring has $c(t) = c(w)$, then we encode as follows:

- If both precolored vertices force (v_2, v_3), then by table 4.9 there is a way to color the hypergraph vertices such that $f(t_2) = f(t_3)$, $f(w_2) = f(w_3)$, and $f(t_1) \neq f(t_2)$ and $f(w_1) \neq f(w_2)$ which gives us $c(t) = c(w) = 1$; switch color classes 1 and 3 if necessary.

- If the precolored vertices force (t_1, t_2) and (w_2, w_3), then by table 4.9 we can give t_1 and t_2 the same color and at the same time have $f(w_2) \neq f(w_3)$. Then by coloring t_3 and w_1 appropriately, we get $c(t) = c(w) = 3$; switch color classes 1 and 3 if necessary.

Now, suppose our coloring must have $c(t) \neq c(w)$. As before, we assume $c(v) = 2$.

- If (v_2, v_3) is forced by both precolored vertices, there exists a coloring of the hypergraph such that $f(t_2) = f(t_3)$ and $f(w_2) \neq f(w_3)$; see table 4.9. Now, color t_1, w_1 such that

$f(t_1) \neq f(t_2)$ and $f(w_1) = f(w_2)$ giving us $c(t) = 1$, $c(w) = 3$. Permute color classes if necessary.

- Assuming forcings on (t_1, t_2) and (w_2, w_3), color such that $f(t_1) = f(t_2)$ and $f(w_2) = f(w_3)$. By ensuring $f(t_2) \neq f(t_3)$ and $f(w_1) \neq f(w_2)$ it follows that $c(t) = 3$, $c(w) = 1$.

Case III (One vertex influences (v_1, v_2) and (w_1, w_2), while the other influences (v_2, v_3) and (w_2, w_3)): In this case, one can see by table 4.9 that it is possible to color under these constraints such that $f(v_1) = f(v_2)$ and $f(w_2) = f(w_3)$ with $f(v_3) \neq f(v_1)$ and $f(w_1) \neq f(w_2)$. This means $c(v) \neq c(w)$, as desired. ■

CONJECTURE **4.6.1** $\mathbf{U}^2(k - COL)$ *is* NP-complete.

This conjecture is on fairly solid ground via a reduction from $\widehat{\mathbf{U}}^2(\mathbf{H3K2})$. Let G be the 4-chromatic, girth 6 graph (due to Descartes [14]) constructed as follows. Let $I = \{v_1, v_2, \ldots, v_{19}\}$ be an independent set of size 19. Construct $\binom{19}{7}$ disjoint copies of the 7-cycle C_7, and match the vertices of each cycle with a different subset $S \subseteq I$ with $|S| = 7$. G contains $19 + 7\binom{19}{7} = 352735$ vertices and $14\binom{19}{7} = 705432$ edges. Let $\alpha = (a, v'_1, v'_2, v'_3, v'_4, v'_5, v'_6)$, $\beta = (b, v'_7, v'_8, v'_9, v'_{10}, v'_{11}, v'_{12})$, $\delta = (c, v'_{13}, v'_{14}, v'_{15}, v'_{16}, v'_{17}, v'_{18})$ be the 7-cycles in G with all three vertices a, b, c adjacent to v_{19}, and v'_i adjacent to $v_i \in I$ for $1 \leq i \leq 18$. Let G' be the graph that results by removing the edges $v'_3v'_4$, $v'_9v'_{10}$ and $v'_{15}v'_{16}$ from G. Then G' is 3-colorable, and remains so even if any two of these edges are added back to the graph.

We now construct the graph in figure 4.8 by creating two copies of G' and joining them together by merging together the copies of the vertices v'_3, v'_9, and v'_{15} and adding an edge between the vertices corresponding to v'_4, v'_{10}, and v'_{16}. Other than this, the two copies of G' are disjoint. It can be checked that this new graph has girth 6.

We see that coloring v'_3 and v'_4 differently has the same effect as adding an edge between those vertices. Hence, in any valid 3-coloring of the graph in figure 4.8, at least one of the pairs $\{v'_3, v'_4\}$, $\{v'_9, v'_{10}\}$, and $\{v'_{15}, v'_{16}\}$ must be monochromatic. However, not all three can be monochromatic, since if they are then all three of the pairs $\{v'_3, v''_4\}$, $\{v'_9, v''_{10}\}$, and $\{v'_{15}, v''_{16}\}$ have different colors, which means that the second copy of G' cannot be colored. Thus, if we encode the two colors used in **H3K2** as "same" and "different" (the same way that we encoded NAE-3SAT in the proof of theorem 4.2.1), then each hyperedge of a

H3K2 instance can be encoded using a copy of this large graph. We will then have a valid hypergraph 2-coloring if and only if our constructed graph is 3-colorable.

The final (and most difficult) step would be to show that provided the graph is 3-colorable, then it remains so even if one arbitrarily adds any two edges to the graph. This would complete the proof that $\widehat{\mathbf{U}}^2(\mathbf{H3K2}) \propto_m \mathbf{U}^2(3-\mathrm{COL})$; the result could be extended to k-COL by the same method used in section 4.5. At the time of this writing, not all the cases have been analyzed. Although we do not foresee any problems, we cannot consider this to be more than a conjecture that is quite likely to be true.

On less sure footing is the conjecture that theorem 4.1.1 tells us the exact point where $\mathbf{U}^i(k-\mathrm{COL})$ becomes polynomial. Once i drops below $f(k,k) = \binom{k}{2}$, graphs in $\mathbf{U}^i(k-\mathrm{COL})$ are not required to consist of small components. Furthermore, the existence of large girth $k+1$-chromatic graphs [16] tells us that checking for subsets of $k+1$ vertices that induce a clique after i edges are added is insufficient — there are graphs not in $\mathbf{U}^{\binom{k}{2}-1}(k-\mathrm{COL})$ that pass this test. Without proof, we state our suspicions as conjecture 4.6.2.

Conjecture **4.6.2** $\mathbf{U}^i(k-\mathit{COL}) \notin \mathrm{P}$ *for* $i < \binom{k}{2}$.

Proving that $\mathbf{U}^{\binom{k}{2}-1}(k-\mathrm{COL})$ is NP-complete via a many-to-one reduction is likely to be quite difficult. Such a reduction must involve the construction of large girth components, since any component containing $k+1$ or more vertices must have girth greater than $k+1$: a smaller girth would violate the conditions of lemma 4.1.1. Note that such a reduction could not be made up of only small disconnected components, since if every component of G has size $\leq k$ then its membership in $\mathbf{U}^{\binom{k}{2}-1}(k-\mathrm{COL})$ can be computed by brute force in

$$O\left(\binom{\binom{n}{2}}{\binom{k}{2}-1} k^{k\binom{k}{2}}\right) = O\left(n^{k^2}\right) \quad \text{(for fixed } k\text{)}$$

which is polynomial in n for fixed k.

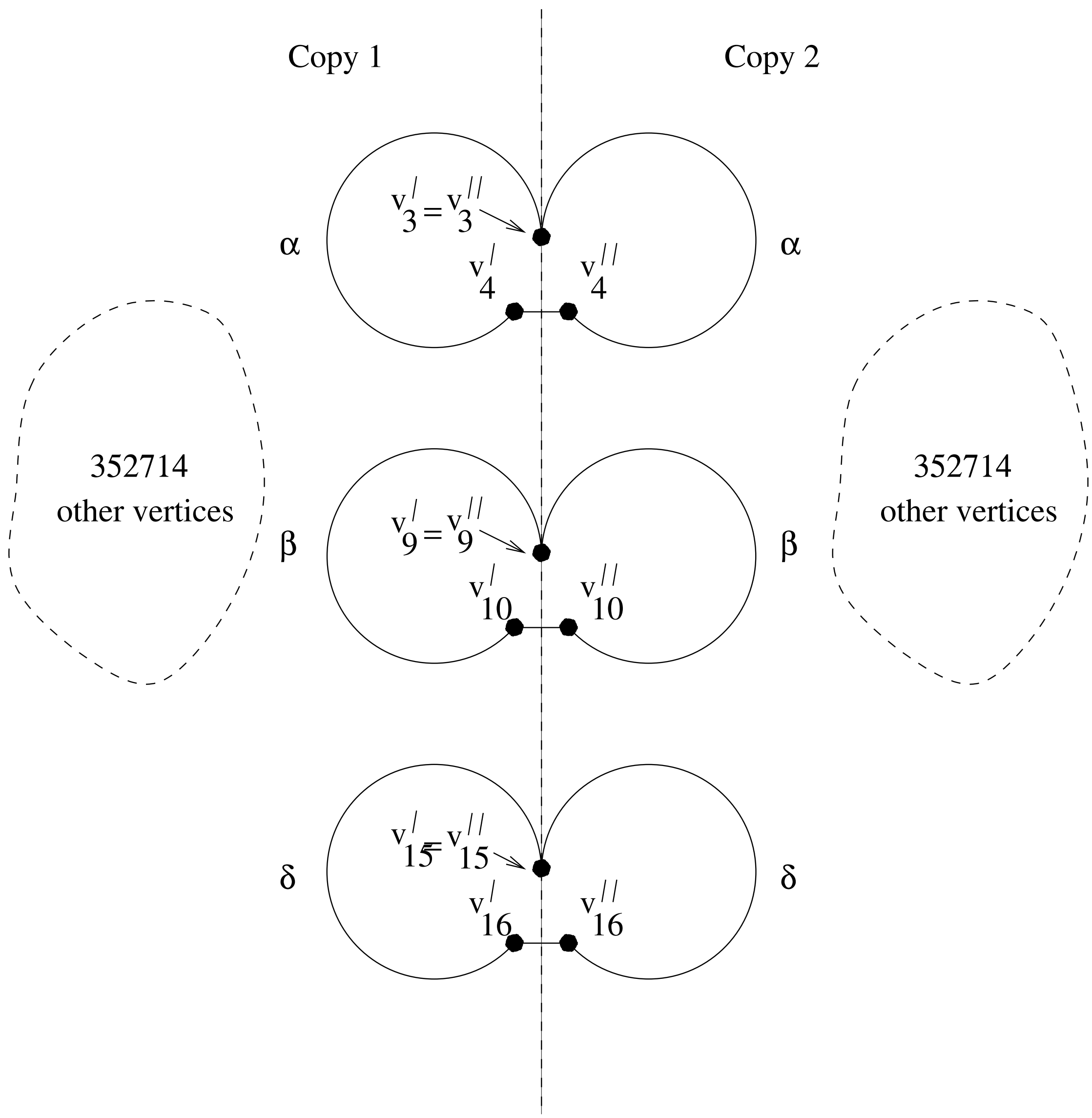

Figure 4.8: Connections between the two copies of G'

Chapter 5

Maximal Properties

It is well known that there exist subsets of instances of NP-complete problems that are not themselves NP-complete. For example, the satisfiability problem instances in conjunctive normal form with maximum clause length two can be solved in $O(n+m)$ time. Certain classes of graphs (for example, *chordal graphs*) can be optimally k colored in polynomial time [34]. In this chapter, we will investigate those instances that are maximally constrained with respect to a monotone property X.

Definition: Let X be a monotone down graph property. We define the *maximally constrained* property $\max_c(X)$ as being those instances $G \in X$ such that for all $e \in \overline{E}$, $G + e \notin X$. ■

This definition can be extended to most other (non-graph) monotone properties by considering instances that are maximally constrained with respect to the addition of some natural structure. For instance, we define $\max_c$(3-SAT) as those satisfiable 3-SAT instances which become unsatisfiable if any 3-clause is added to the instance.

Maximally constrained problems are often considered in extremal graph theory, insofar as researchers often wish to answer questions such as, "What is the maximum number of edges that a graph with property X may have?" [34]. To prove that a certain upper bound m_0 on the number of edges is exact, one must normally construct a graph $G \in X$ with $|E| = m_0$; by our definition such a G is also in $\max_c(X)$. Note that even if the bound is exact, there can exist maximally constrained graphs with fewer than m_0 edges. However, as in previous chapters we are concerned with the complexity of identifying maximally constrained instances rather than computing bounds over the set of all maximal instances.

Complexity is more of an issue in the constraint satisfaction problem (CSP) community. For example, in [20] the authors use a constrainedness measure of an ensemble of instances to help explain the phase transition phenomenon (see section 2.3). Most descriptions of the phase transition appeal to the intuition that the hard instances occur at the midpoint between easy "under-constrained" and "over-constrained" problem instances.

It is apparent that every $X \in \text{P}$ has $\max_c(X) \in \text{P}$. For many of the "standard" NP-complete problems, $\max_c$ versions are in P.

THEOREM **5.0.2** *The* $\max_c$ *versions of the following standard* NP-complete *problems can be solved in polynomial time: k-COL,* **HC**, **HP**, *3-SAT, k-arity hypergraph 2-coloring, independent set.*

Proof: For k-COL, note that a $G \in \max_c(k - \text{COL})$ must be a k-partite complete graph, which can be identified and colored in polynomial time. As we have defined **HC** on the complement graph, $G \in \max_c(\mathbf{HC})$ if and only if $G^c \cong C_n$, a simple cycle on n vertices. Again, this condition is trivial to check; the case for **HP** is similar. Let Q be a maximal 3-SAT instance on the n variables $x_1, x_2, \ldots, x_n$ with satisfying truth assignment $t : \{x_1, x_2, \ldots, x_n\} \to \{T, F\}$. Then, by maximality, for every three variables a, b, c, Q must contain all seven possible clauses on a, b, c that do not contradict t. Thus, Q has a structure that is recognizable in polynomial time, and its single satisfying solution can also be determined with polynomial work. Hypergraph 2-coloring is similar: let c be a valid 2-coloring of the vertices $V = \{v_1, \ldots, v_n\}$. By maximality, for every three vertices u, v, w, (u, v, w) is *not* a hyperedge if and only if $c(u) = c(v) = c(w)$ Thus, in a $\max_c$ instance the color classes can be identified in polynomial time. As for independent set, a maximal instance is a graph that contains exactly one independent set of size k, and every vertex u that is not in the independent set is connected to every other vertex in the graph. Again, a polynomial time algorithm to identify this situation exists. ■

In fact, the $\max_c$ version of every "standard" NP-complete problem that we have looked at is in P. Does this mean $\max_c(X) \in \text{P}$ for every $X \in \text{NP}$? Unless isomorphism is in P, the answer is no by theorem 5.1.1, which gives an example of an NP-complete problem whose maximal version is isomorphism complete. Furthermore, theorem 5.2.1 shows that maximal problems can in fact be NP-hard.

Note that if $\max_c(X) \notin \text{P}$, then it is not clear if it is even in NP, even if X itself is in NP. Although showing that a maximal instance has property X is in NP, showing that it is *not* in X after adding any edge is in CO-NP. Hence, $\max_c(X) \in \text{CO-NP} \cup \text{NP}$, which implies it could be outside of NP.

k-Complementary Subgraph (k-CSG)
Input: A graph $G = (V, E)$ and an integer k.
Problem: Does there exist a subset $V' \subseteq V$, $|V| = k$ such that $G[V'] \subseteq G^c[V \setminus V']$? (The subgraph is not induced).

Figure 5.1: The k-complementary subgraph problem

5.1 Isomorphism

In this section, we consider a variant of the subgraph isomorphism problem called *k-complementary subgraph* (**k-CSG**); see figure 5.1. Notice that, unlike the standard subgraph isomorphism problem, **k-CSG** is a monotone down graph property. It is obvious that **k-CSG** $\in$ NP; in theorem 5.1.2 we will in fact show that it is NP-complete. This will serve as our first problem in NP for which the $\max_c$ version is unlikely to be in P.

THEOREM **5.1.1** $\max_c$(**k-CSG**) *is isomorphism complete.*

To prove the theorem we will make use of the following definitions and results. In what follows, a graph denoted by G_j has vertex set V_j and edge set E_j.

Lemma 5.1.1 *Let $G = G_1 \times G_2$ and $k = |V_1| = |V_2|$. If $G \in$ **k-CSG** then either $V' \subseteq V_1$ or $V' \subseteq V_2$.*

Proof: Suppose V' is composed of m $(0 < m < k)$ vertices from V_1 and $k - m$ vertices from V_2. Then $G[V']$ is a connected graph while $G^c[V \setminus V']$ is disconnected, so $G[V'] \not\subseteq G^c[V \setminus V']$. ■

Lemma 5.1.2 *For* $\max_c$(**k-CSG**), *if $k < \frac{n}{2}$ then the problem can be reduced in polynomial time to a case where $k' = \frac{n'}{2}$*

Proof: If $G = (V, E) \in \max_c$(**k-CSG**), then there exist disjoint subsets of vertices $V', V'' \subseteq V$, $|V'| = |V''| = k$ such that $G[V'] \cong G^c[V'']$. Since G is maximal, every vertex in $V \setminus (V' \cup V'')$ must have degree $|V| - 1$. Thus, to check if a given graph G has the property $\max_c$(**k-CSG**) where $k < \frac{n}{2}$, we can first remove $|V| - 2k$ vertices of degree $|G| - 1$ from G to reduce the problem to one where $k = \frac{|G|}{2}$. If there are not that many vertices of large degree in G, then we conclude $G \notin \max_c$(**k-CSG**). ■

Definition: R is the relation on the vertices V of a given graph G defined by $uRv \iff uv \notin E$. Set $vRv \; \forall v \in V$. ■

Definition: The equivalence relation $\sim$ is the transitive closure of R. ■

Lemma 5.1.3 *If $G_1 \cong G_2$ and $G = G_1 \times G_2^c$, then $\exists$ an equivalence class $C \subseteq V$ under $\sim$ such that $G[C] = G_1$ or $G[C] = G_2^c$.*

Proof: Clearly, if $u \in G_1, v \in G_2^c$, then $u \not\sim v$. Thus, for any equivalence class $\overline{u}$ of $\sim$, either $\overline{u} \subseteq V_1$ or $\overline{u} \subseteq V_2$. Therefore, the equivalence classes of $\sim$ partition both V_1 and V_2. Now, suppose $G_1 \neq G[\overline{u}]$ for any $u \in V$. Then $G_1 = G[\overline{u_1}] \times G[\overline{u_2}] \times \cdots \times G[\overline{u_m}]$ for some vertices $u_1, u_2, \ldots, u_m \in V_1$. But then $G_2^c \cong G_1^c$ is a disconnected graph, which implies $V_2 = [v]$ for some $v \in V_2$. ■

We are now in position to prove theorem 5.1.1.

Proof: (Isomorphism reduces to $\max_c$(k-**CSG**)). Given two graphs G_1, G_2 with $|V_1| = |V_2|$, let $G = G_1 \times G_2^c$ and $k = |V_1| = |V_2|$. If $G_1 \cong G_2$ then setting V' to V_1 gives $G[V'] \subseteq G^c[V \setminus V']$, so $G \in k$-**CSG**. Furthermore, if any edge is added to G then $G[V'] \not\subseteq G^c[V \setminus V']$ since adding an edge to either G_1 or G_2 means that there will be more edges in $G[V']$ than in $G^c[V \setminus V']$. Hence, $G \in \max_c$(k-**CSG**). On the other hand, by lemma 5.1.1 the only possible choice for V' is either V_1 or V_2, so $G \in \max_c(k\text{-}\mathbf{CSG}) \Longrightarrow G_1 \cong G_2$.

($\max_c$(k-**CSG**) reduces to isomorphism.) By lemma 5.1.2, we need only consider the case where $k = \frac{n}{2}$. In this case $G \in \max_c(k\text{-}\mathbf{CSG}) \iff \exists V' \subseteq V$ such that $G = G[V'] \times G[V \setminus V']$ and $G[V'] \cong G^c[V \setminus V']$. Thus, provided we can find V' (or its complement $V \setminus V'$) in polynomial time, we can determine if a given graph $G \in \max_c$(k-**CSG**) by checking if $G_1 = G[V']$ is isomorphic to $G_2 = G^c[V \setminus V']$.

Now, given a graph G, compute its equivalence classes with respect to the relation $\sim$. This can be done in polynomial time. By lemma 5.1.3, if $G \in \max_c$(k-**CSG**), then $\exists$ an equivalence class $\overline{v}$ of size $\frac{n}{2}$. If no such class exists, then $G \notin \max_c$(k-**CSG**). Let $V' = \overline{v}$. Then $G \in \max_c(k\text{-}\mathbf{CSG}) \iff$ every vertex in V' is adjacent to every vertex in $V \setminus V'$ and $G[V'] \cong G^c[V \setminus V']$. ■

We also note

Theorem **5.1.2** k-**CSG** $\in$ NP-complete *for* $k = \frac{n}{2}$.

Proof: We already have k-**CSG** $\in$ NP, so we will complete the proof by showing that **HC** $\propto_m$ k-**CSG**. Let G be any graph, and let G' be the complete join of a cycle on n vertices and the complement of G; $G' = C_n \times G^c$. Then G contains a Hamiltonian cycle if and only if $G' \in k$-**CSG**.

($\Longrightarrow$) If G contains a Hamiltonian cycle, then setting V' to the vertices in the n-cycle gives us $G'[V'] \subseteq G'^c[V \setminus V']$.

($\Longleftarrow$) If $G' \in k$-**CSG**, then by lemma 5.1.1 it follows that either $C_n \subseteq G$, or $G^c \subseteq C_n^c$. If $C_n \subseteq G$ then we are done. On the other hand, if $G^c \subseteq C_n^c$ then for every non-edge in C_n^c there is a corresponding non-edge in $G^c \Longrightarrow C_n \subseteq G$. ∎

5.2 Maximal Problems can be NP-hard

It is known that 3-coloring is NP-complete even if it is restricted to graphs with maximum degree 4 [19]. The maximum degree of a graph G is denoted by $\Delta(G)$. We will consider the complexity of $\max_c(3 - \text{COL} \ \& \ \Delta(G) \leq 4)$. However, we do not know whether or not this problem is in NP.

THEOREM **5.2.1** *The problem* $\max_c(3 - COL \ \& \ \Delta(G) \leq 4)$ *is* NP*-hard.*

Proof: Proof is via a reduction from $3 - \text{COL} \ \& \ \Delta \leq 4$; let $G = (V, E)$ be a 3-colorable graph with maximum degree ≤ 4. We may assume that the minimum degree in G is 3 by iteratively selecting a vertex with degree ≤ 2 and removing it until no such low degree vertices remain. A coloring of the reduced graph can be extended to a coloring of the original graph by adding back the removed vertices in reverse order and coloring each with an available colors; since each was adjacent to at most 2 previously colored vertices, a third color is always available.

Note that we must have an even number of degree 3 vertices in G, as no graph can have an odd number of odd vertices. We now create a graph from G where every vertex has degree 4 by iteratively choosing a pair of degree three vertices and attaching them to a $K_{4,4}$ with one edge missing; see figure 5.2.

Notice that this does not alter the colorability of G. If v and w must be colored the same color, then the $K_{4,4}$ may be 2-colored with the remaining colors. If they are colored differently, then use v's color for the right side and w's color for the left side of the $K_{4,4}$.

Continue modifying the graph until no degree 3 vertices remain. At this point, no edge can be added to the graph without violating the condition that the maximum degree must be less than or equal to 4. Furthermore, the constructed graph is colorable if and only the original graph G was colorable. Hence, we have shown that $(3 - \text{COL} \ \& \ \Delta \leq 4) \propto_m \max_c(3 - \text{COL} \ \& \ \Delta \leq 4)$, as desired. ∎

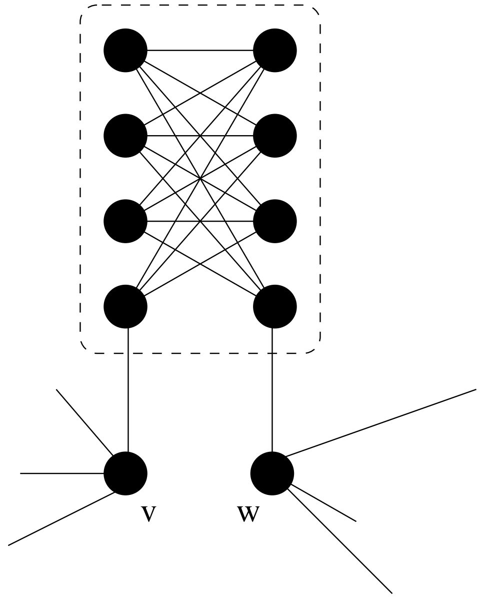

Figure 5.2: Subgraph used during proof that (3−COL & $\Delta \leq 4$) $\propto_m$ $\max_c$(3−COL & $\Delta \leq 4$) to change two degree three vertices (v and w) to degree 4 vertices.

5.3 Another Polynomial Maximal Problem

We saw in section 5.2 that the combined problem $\max_c(3 - \text{COL} \ \& \ \Delta(G) \leq 4)$ is NP-hard. This suggests that we may create other difficult $\max_c$ problems by combining two NP-complete problems. One possibility is $\max_c(3 - \text{COL} \ \& \ \mathbf{HC})$. Remember that we have defined **HC** on the complement graph. It seems conceivable that maximal instances can "max out" on either property, which could lead to a complicated graph structure that is difficult to recognize. However, this is not the case.

Theorem **5.3.1** $\max_c(3 - \mathit{COL} \ \& \ \mathbf{HC}) \in \mathrm{P}$.

Proof: Let $G \in \max_c(3 - \text{COL} \ \& \ \mathbf{HC})$. Since G is 3-colorable, its vertices can be split into 3 independent sets C_1, C_2, C_3 corresponding to the color classes. As G is maximal, all edges possible between the color classes will be in G, with the exception of those non-edges (edges in G^c) needed to have a Hamiltonian cycle in G^c. As the C_is are cliques in G^c, we will have property **HC** provided we can get from one clique to another. Since G is maximal, every non-edge between color classes *must* be used by every Hamiltonian cycle in G^c. Figure 5.3 shows the only two possible configurations of edges between the C_is in G^c.

The reader can check that any additional edges between the C_is are superfluous, and that these really are the only cases. Note that if we have $|V| \geq 7$ then we must have at least

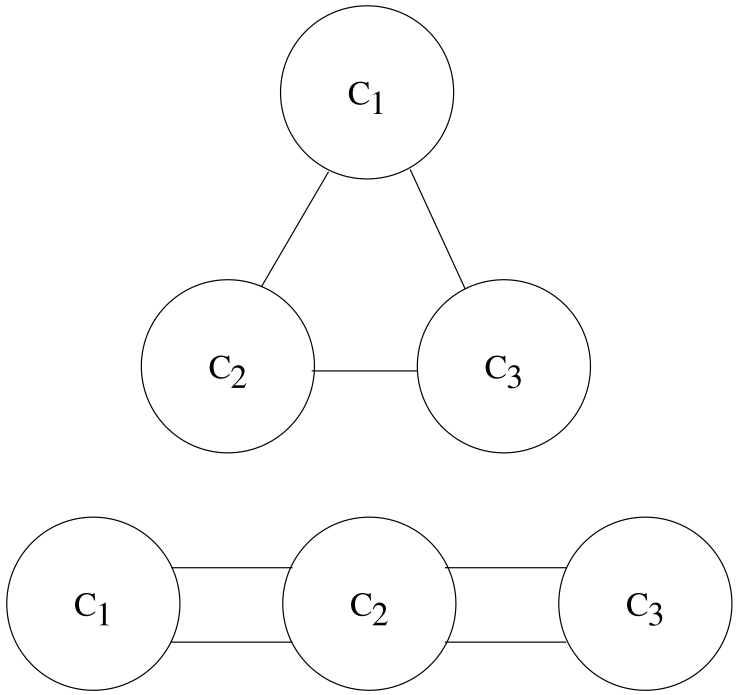

Figure 5.3: The two possible ways for the color classes to be connected in an instance of $\max_c(3 - \text{COL} \ \& \ \mathbf{HC})$ as viewed in G^c.

3 color classes. A single edge means that we must have 2. If we have only two, then one class contains at least 4 vertices, from which we can split off a new class with two vertices x, y by adding edges between these vertices and those that are in their current class.

A naive algorithm could identify graphs in $\max_c(3-\text{COL} \ \& \ \mathbf{HC})$ by assuming that it has one of the two forms shown in figure 5.3 adding all possible combinations of an appropriate number of edges (either 3 or 6) and then checking that the resulting graph is complete 3-partite. Such an algorithm is $O(n^{12})$, which is polynomial. ■

A similar argument shows that $\max_c(k - \text{COL} \ \& \ \mathbf{HC})$ is in P for fixed k. However, the number of configurations such as those shown in figure 5.3 increase exponentially in k. Thus, this proof does not show that the problem is polynomial if k is an input value.

5.4 An Incomplete Identification Algorithm for Graphs in $\max_c(\mathbf{U}(k\text{-}\mathbf{set}))$

In contrast to previous sections, we have been unable to determine the complexity of $\max_c(\mathbf{U}(X))$ for any property $X \in$ NP-complete. In this section we prove some properties about graphs in $\max_c(\mathbf{U}(k\text{-}\mathbf{set}))$, and show how these properties might be used in an algorithm that identifies such graphs. Section 5.5 provides examples of graphs in $\max_c(\mathbf{U}(k\text{-}\mathbf{set}))$.

5.4.1 Graph Properties

We wish to determine whether or not a given graph G is maximal with respect to $U(k\text{-}\mathbf{set})$. Suppose $G \in \max_c(\mathbf{U}(k\text{-}\mathbf{set}))$. Then G has the following properties:

1. For any edge e, $G + e \in k\text{-}\mathbf{set}$.
2. For any edge e_1 such that $e_1 \notin E(G)$, $\exists e_2$ such that $G + e_1 + e_2 \notin k\text{-}\mathbf{set}$.
3. Let v, w be vertices in G. If v and w are never in the same k-**set**, then (v, w) is an edge.
4. Conversely, if $(v, w) \notin E(G)$, then $\exists I \subseteq G$, an independent set of size k in G such that $u \in I$, $v \in I$.

Item 1 holds by the definition of $U(k\text{-}\mathbf{set})$, while 2, 3, and 4 follow by the maximality of G.

5.4.2 Degree Test and Pruning

We prune G by applying property 3; namely, if a vertex v never appears in any k-**set**, then $\deg(v) = |G| - 1$. Hence, we can remove from G any vertex that is adjacent to all others.

By property 4, if a vertex v is not adjacent to some other vertex, then v appears in a k-**set** with that vertex. Clearly, for v to be in a k-**set** it must be non-adjacent to at least $k - 1$ vertices, which implies $\deg(v) < |G| - k + 1$. Thus, after pruning, we check that for all $v \in G$, $0 \leq \deg(v) < |G| - k + 1$.

In what follows, we assume that G has already been pruned, and that the degree constraint is satisfied. Thus, we can assume that *every vertex in G is contained in at least one k-**set** of G*. Furthermore, we will assume that G does not contain a $k + 1$-set.

5.4.3 Ensure $G \in U(k\text{-set})$ (Step One)

Let $\mathcal{I} = \{I_1, I_2, \ldots, I_s\}$ be the family of all distinct k–sets in G. We say an edge $e = (u, v)$ *destroys* an independent set I if both $u \in I$ and $v \in I$. By 1, no single edge can destroy all the sets in $\mathcal{I}$. Then for every edge (u, v) in the complement G^c of G, $\exists I \in \mathcal{I}$ such that either $u \notin I$ or $v \notin I$. It follows that

$$\left|\bigcap_{I\in\mathcal{I}} I\right| \leq 1$$

We will exploit this property when we test that no edge destroys all the k–sets in G. Let $\mathcal{I}_{(u,v)} = \{I \in \mathcal{I} | u \notin I \text{ or } v \notin I\}$. For each edge $(u,v) \in E(G^c)$, we will compute $W_{u,v} = \bigcup_{I\in\mathcal{I}_{(u,v)}} I$. Provided $G \in U(k\textbf{-set})$, we are guaranteed that $W_{u,v} \neq \emptyset$. The *hope* (at this stage, at least!) is that it will be easy to find the k–sets in $W_{u,v}$.

But first: how can we possibly determine $W_{u,v}$ without first finding all the sets in $\mathcal{I}_{(u,v)}$? We will do so by computing $W^c_{u,v} = G - W_{u,v}$ via an auxiliary set S as follows.

Proposition 5.4.1 *Fix a pair $e = (u,v)$. Let $S = N(u) \cup N(v)$. Then $W^c = \bigcap_{y\in S} N(y)$.*

Proof: ($\subseteq$) Let $x \in W^c_{u,v}$. By our assumptions in section 5.4.2, x is contained in some k**-set** J in G. Since $J \notin \mathcal{I}_{(u,v)}$ (recall that $W^c_{u,v} = G - \bigcup_{I\in\mathcal{I}_{(u,v)}} I$), it follows that $u, v \in J$. Thus, any time x appears in an independent set, u and v appear with it. Now, let $y \in S$. By property 3, y can never appear in an independent set that contains both u and v, which means it also cannot be in an independent set with x. Therefore, $(x,y) \in E(G)$.
($\supseteq$) Let $y \in \{x \in V(G) | (x,y) \in E(G)\ \forall y \in S\}$. Let $I \in \mathcal{I}_{(u,v)}$. Since at least one of u or v does not appear in I, $\exists x \in I$ such that $(x,u) \in E(G)$ or $(x,v) \in E(G)$; i.e., $x \in S$. By definition, $(y,x) \in E(G)$, so $y \notin I$. Since the choice of I was arbitrary, it follows that $y \notin W_{u,v}$. ■

Thus, it is possible to determine which vertices are in $W^c_{u,v}$. By removing these vertices from G, we arrive at the induced graph $G\left[\bigcup_{I\in\mathcal{I}_{(u,v)}} I\right] = G[W_{u,v}]$.

5.4.4 Ensure G Is Maximal (Step Two)

We are now working with the induced graph $G' = G\left[\bigcup_{I\in\mathcal{I}_{(u,v)}} I\right]$. Recall that it was constructed by removing all vertices that are only found in independent sets that contain both u and v and then adding an edge $e = uv$ to G.

By assumption, since G is maximal with respect to $U(k\textbf{-set})$, $\exists f \in E(G')$ that destroys every independent set in G'. (If not, $G + (u,v)$ is also in $U(k\textbf{-set})$). Thus, it follows that:

$$\left|\bigcap_{I\in\mathcal{I}_{(u,v)}} I\right| \geq 2$$

This is easily checked; just look for at least two degree 0 vertices in G'.

We are now at the final step: ensure that there really *is* a k**-set** hiding in here. We know that:

1. Every vertex in G' is part of a k**-set**.
2. If $(x, y) \in \overline{G'}$, then x and y appear in a k**-set** together in the full graph G. However, they might have *only* appeared in a set that contained the (now illegal) pair u, v. How can we detect this?

At these point we have come to a dead end. Completing the algorithm or showing that $\max_c(\mathbf{U}(k\textbf{-set})) \in$ NP-complete remains an open problem.

5.5 Examples of Graphs with the $\max_c \mathbf{U}(k\textbf{-set})$ Property

- A $k+1$-set.
- $K_{k,k}$.

Another example of an infinite class of graphs $G \in \max_c(\mathbf{U}(k\textbf{-set}))$ can be constructed as follows. Let $k = 2n$, $n > 2$. Let $I_0 = \{v_1, v_2, \ldots, v_{2n-1}, v_{2n}\}$ be an independent set. We will also make use of vertices from the set $\{u_1, u_2, \ldots, u_{2n}\}$; the vertex set of G will then be $\{v_1, v_2, \ldots, v_{2n-1}, v_{2n}\} \cup \{u_1, u_2, \ldots, u_{2n}\}$. For $1 \leq i \leq n$ let $I_i = (I_0 \setminus \{v_{2i-1}, v_{2i}\}) \cup \{u_{2i-1}, u_{2i}\}$.

Now, insert the edges:

$$\begin{aligned} \forall i \quad \rightarrow \quad & (v_{2i-1}, u_{2i-1}) \\ & (v_{2i}, u_{2i}) \\ & (v_{2i-1}, u_{2i}) \\ & (v_{2i}, u_{2i-1}) \\ \forall i \neq j \quad \rightarrow \quad & (u_{2i-1}, u_{2j-1}) \\ & (u_{2i-1}, u_{2j}) \\ & (u_{2i}, u_{2j-1}) \\ & (u_{2i}, u_{2j}) \end{aligned}$$

This results in a graph G with $n+1$ distinct independent sets, these being $I_0, I_1, \ldots, I_n$. We will now show that $G \in \max_c U(k\textbf{-set})$ by demonstrating that adding any single edge

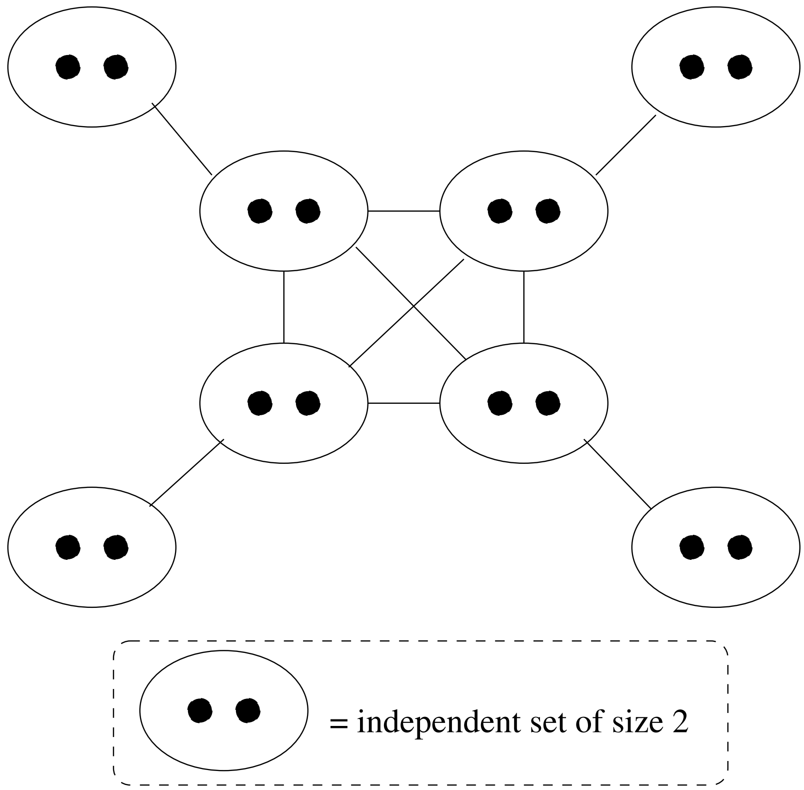

Figure 5.4: A graph in $\max_c(\mathbf{U}(k\textbf{-set}))$ with $k = 8$.

will result in a graph that contains at least one k**-set**, yet all these remaining k–sets can be destroyed by adding a second edge.

For $1 \leq i \leq n$, putting an edge between (v_{2i-1}, v_{2i}) destroys all the independent sets except I_i; of course, this single k**-set** can be eliminated by the addition of another edge.

If (v_i, v_j) is added, then two Is survive, those being whichever contain v_i but not v_j and vice versa. Provided $n > 2$, these will share a pair, so (v_i, v_j) cannot be added to G and still have $U(k\textbf{-set})$ hold.

Similar arguments work for the cases of edges between u_i's, and between v's and u's.

An example of this construction for the case $n = 4$ can be seen in figure 5.4. In that figure, the edges between the 2-sets indicate that all possible edges are inserted between the vertices in the sets.

Chapter 6

Conclusions

The sudden appearance of a backbone has been cited by many researchers as a possible cause of the difficulty of randomly generated problem instances at phase transitions. We have introduced an unfrozen hierarchy of properties $U(X)$ that do not have backbones. By showing that $U(k\text{-SAT})$, $U(k-\text{COL})$, and others are NP-complete, we have shown that the existence of a backbone is not necessary for an instance to be difficult to solve.

For $k \geq 3$, the classes of $\mathbf{U}^i(\text{k}-SAT)$ and $\mathbf{U}^i(k-\text{COL})$ change from being NP-complete, for $i \leq 1$, to being in P for larger i. This introduces an infinite set of problems on the boundary between P and NP. Significant progress was made in determining where the change from NP-complete to P occurs for $\mathbf{U}^i(k-\text{COL})$.

It should be noted that if a randomly generated instance of k-SAT is in $U(\text{k}-SAT)$, then it is likely that we are far from the phase transition: empirically, we know that instances at the phase transition have a backbone. Graphs in the polynomial class of $\mathbf{U}^{\binom{k}{2}}(k-\text{COL})$ are also far from the threshold. Following a result quoted in [34], let $t_k(n) = n^{-k/(k-1)}$. Given a graph $G \in \mathcal{G}(n,p)$, if $\lim_{n\to\infty} p/t_{k+1} = \infty$ then G almost surely contains a $k+1$-tree, which by lemma 4.1.2 implies that $G \notin \mathbf{U}^{\binom{k}{2}}(k-\text{COL})$. Thus, a necessary condition for G to have the property is that $p \in O(n^{-(k+1)/k})$. The expected number of edges in G is then $O(p\binom{n}{2}) = O(\binom{n}{2}n^{-(k+1)/k})$ which is $o(n)$. But the graphs at the phase transition have $\approx 2.3n$ edges (for $k = 3$). It follows that the class $\mathbf{U}^{\binom{k}{2}}(k-\text{COL})$ is very far from the phase transition.

Unfrozenness can also impose implicit minimum size restrictions on graphs in the class. For example, a graph in $\mathbf{U}^{1000}(\mathbf{HC})$ must have minimum degree ≥ 1002; the graph must remain Hamiltonian even if 1000 edges are removed from a single vertex. This already implies that the graph must contain at least 1003 vertices. However, Dirac's well-known result (see [34]) states that if $\delta(G) \geq n(G)/2$, then G is Hamiltonian. Thus, under the assumption P $\neq$ NP, hard instances of $\mathbf{U}^{1000}(\mathbf{HC})$ must have more than 2004 vertices.

We have also demonstrated the existence of maximally constrained problems that are not in P assuming P $\neq$ NP. Although this provides us with another set of interesting problems, it is not clear what relationship this has to the constrainedness measures used in the CSP literature. See section 6.1 for a brief discussion.

6.1 Future Research

Even though hard problems without backbones exist, it should be noted that a backbone can develop during a backtracking search for a solution. After setting a variable during search, it may be the case that any extension of the current partial solution requires other variables to take on particular fixed values; i.e., a backbone is developing. Further research could examine how the backbone develops during search. Is it the case that hard instances without backbones are those that are just a few steps away from having one?

We know of no reductions $X \propto_m \mathbf{U}(Y)$ for an NP-complete problem X that does not also have the property that the instance reduced to has property Y if and only if it also has property $U(Y)$. It would be interesting to know if this is the only kind of reduction possible, or if there exist reductions where there exist instances not in X that correspond to instances that have property Y but not property $U(Y)$. Also, is it possible that determining a solution to an instance X given that it has property $\mathbf{U}(X)$ can be in P even if $X \in$ NP-complete?

It remains an open problem to determine the values of i_k and j_k for which $\mathbf{U}^{i_k}((\text{k}-SAT))$ and $\mathbf{U}^{j_k}(k-\text{COL})$ are in P while $\mathbf{U}^{i_k-1}((\text{k}-SAT))$ and $\mathbf{U}^{j_k-1}(k-\text{COL})$ are not in P. This is expected to be quite difficult to do.

We were also unable to determine the complexity of $\max_c(\mathbf{U}(X))$ for an NP-complete problem X. (An exception is $\max_c(\Delta(G) \leq 4 \ \& \ 3-\text{COL})$, which consists of all graphs with maximum degree 3, minimum degree 2, and at most one vertex with degree 2). This could be due to the fact that most "easy" maximal instances have the property that the number of solutions is very small — the constraints clearly enforce certain solutions. For example, an instance of $\max_c(k-\text{COL})$ has exactly $3! = 6$ solutions which are found by permuting the color classes. On the other hand, even a maximally constrained instance of $U(k-\text{COL})$ has many solutions, since a solution is a separate coloring for every edge. Even if there are only two colorings of the graph, for a large number of possible edges $e \in \overline{E}$ *both* colorings may work for $G + e$; this potentially leads to an exponential number of certificates.

Bibliography

[1] Dimitris Achlioptas. Mick gets more than pie. In *DIMACS Workshop on Probabilistic Analysis of Algorithms for Hard Problems*, 1999.

[2] Dimitris Achlioptas, Carla Gomes, Henry Kautz, and Bart Selman. Generating satisfiable problem instances. In *Proceedings of AAAI-2000*. AAAI Press / The MIT Press, 2000. Available at http://www.research.att.com/~kautz/papers-ftp/index.html.

[3] Dimitris Achlioptas, Lefteris Kirousis, Evangelos Kranakis, and Danny Krizanc. Rigorous results for $(2+p)$-sat. In *Proceedings of RALCOM 97*, pages 1–10, 1997.

[4] Béla Bollobás. *Random Graphs*. Academic Press, 1985.

[5] Béla Bollobás. *Modern Graph Theory*. Number 184 in Graduate Texts in Mathematics. Springer Verlag, New York, 1998.

[6] Béla Bollobás, Christian Borgs, Jennifer T. Chayes, Jeong Han Kim, and David B. Wilson. The scaling window of the 2-sat transition. Technical Report MSR-TR-99-41, Microsoft Research, 1999. Available from Microsoft Research, http://www.research.microsoft.com/scripts/pubs/view.asp?TR_ID=MSR-TR-99-41.

[7] Gilles Brassard and Paul Bratley. *Fundamentals of Algorithmics*. Prentice-Hall, Inc., 1996.

[8] Robert C. Brigham and Ronald D. Dutton. Changing and unchanging invariants: the edge clique cover number. *Congressus Numerantium*, 70:145–152, 1990.

[9] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the really hard problems are. In *Proceedings of the 12th International Joint Conference on A.I. (IJCAI-91)*, volume 1, pages 331–337. Morgan Kaufmann, 1991.

[10] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd annual ACM symposium on the theory of computing*, pages 151–158. Association for Computing Machinery, 1971.

[11] Stephen A. Cook and David G. Mitchell. Finding hard instances of the satisfiability problem: A survey. In Dingzhu Du, Jun Gu, and Panos M. Pardalos, editors, *Satisfiability Problem: Theory and Application*, volume 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 1–17. American Mathematical Society, 1997.

[12] Joseph Culberson and Ian P. Gent. Frozen development in graph coloring. Technical Report APES-19-2000. Submitted. Available as APES Research Report. http://www.apes.cs.strath.ac.uk/apesreports.html.

[13] Joseph Culberson and Basil Vandegriend. The $\mathrm{G}_{n,m}$ phase transition is not hard for the Hamiltonian cycle problem. *Journal of Artificial Intelligence Research*, 9:219–245, 1998.

[14] Blanche Descartes. Solution to advanced problem 4526, proposed by Peter Ungar. *The American Mathematical Monthly*, 61:352–353, 1954.

[15] Ronald D. Dutton, Sirisha R. Medidi, and Robert C. Brigham. Changing and unchanging of the radius of a graph. *Linear algebra and its applications*, 217:62–82, 1995.

[16] Paul Erdös. Graph theory and probability. *Canadian Journal of Mathematics*, 11:34–38, 1959.

[17] John Franco and R.P. Swaminathan. Average case results for satisfiability algorithms under the random-clause-width-model. *Annals of Mathematics and Artificial Intelligence*, 20:357–391, 1997.

[18] Ehud Friedgut. Sharp thresholds of graph properties, and the k-sat problem. *Journal of the American Mathematical Society*, 12:1017–1054, 1999.

[19] Michael R. Garey and David S. Johnson. *Computers and Intractability*. W.H. Freeman and Company, 1979.

[20] Ian Gent, Ewan MacIntyre, Patrick Prosser, and Toby Walsh. The constrainedness of search. In *Proceedings of AAAI-96*, pages 246–252, 1996.

[21] A. Goldberg. On the complexity of the satisfiability problem. *Courant Computer Science Report No. 16*, 1979.

[22] Frank Harary. Changing and unchanging invariants for graphs. *Bulletin of the Malaysian Mathematical Society*, 5(2):72–78, 1982.

[23] Teresa W. Haynes, Linda M. Lawson, Robert C. Brigham, and Ronald D. Dutton. Changing and unchanging of graphical invariants: minimum and maximum degree, maximum clique size, node independence number and edge independence number. *Congressus Numerantium*, 72:239–252, 1990.

[24] Tad Hogg and Colin P. Williams. The hardest constraint problems: a double phase transition. *Artificial Intelligence*, 69:359–377, 1994.

[25] Gabriel Istrate. Computational complexity and phase transitions. In *15th Annual Conference on Computational Complexity*, 2000.

[26] Alexis Kaporis, Lefteris Kirousis, and Yannis Stamatiou. A note on the non-colorability threshold of a random graph. *The Electronic Journal of Combinatorics*, 7(R29), 2000.

[27] Lefteris Kirousis, Evangelos Kranakis, Danny Krizanc, and Yannis Stamatiou. Approximating the unsatisfiability threshold of random formulas. Technical Report TR-96-27, University of Carleton, 1996.

[28] Richard E. Ladner. On the structure of polynomial time reducibility. *Journal of the Association for Computing Machinery*, 22:155–171, 1975.

[29] W. F. Lindgren. An infinite class of hypohamiltonian graphs. *American Mathematical Monthly*, 74:1087–1089, November 1967.

[30] David Mitchell, Bart Selman, and Hector Levesque. Hard and easy distributions of SAT problems. In *Proceedings of AAAI-92*, pages 459–465. AAAI Press / The MIT Press, 1992.

[31] Rémi Monasson, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman, and Lidror Troyansky. Determining computational complexity from characteristic "phase transitions". *Nature*, 400:133–137, July 1999.

[32] Andrew J. Parkes. Clustering at the phase transition. In *Proceedings of AAAI-97*, pages 340–345. AAAI Press / The MIT Press, 1997.

[33] Josh Singer, Ian P. Gent, and Alan Smaill. Backbone fragility and the local search cost peak. *Journal of Artificial Intelligence*, 12:235–270, 2000.

[34] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 1996.