UNIVERSIDADE FEDERAL DO CEARÁ
CENTER OF SCIENCES
DEPARTMENT OF COMPUTING
POST-GRADUATION PROGRAM IN COMPUTER SCIENCE
MASTER DEGREE IN COMPUTER SCIENCE

RODRIGO NOGUEIRA LIMA DAVID

HARD INSTANCES FOR THE MAXIMUM CLIQUE PROBLEM WITH HIGH PROBABILITY

FORTALEZA
2025

RODRIGO NOGUEIRA LIMA DAVID

HARD INSTANCES FOR THE MAXIMUM CLIQUE PROBLEM WITH HIGH PROBABILITY

Dissertation submitted to the Post-Graduation Program in Computer Science of the Center of Sciences of the Universidade Federal do Ceará, as a partial requirement for obtaining the title of Master in Computer Science. Concentration Area: Theory of Computation

Advisor: Prof. Dr. Victor Almeida Campos

FORTALEZA
2025

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

D275h David, Rodrigo.
Hard Instances for the Maximum Clique Problem with High Probability / Rodrigo David. – 2025.
69 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2025.
Orientação: Prof. Dr. Victor Almeida Campos.

1. Clique Máxima. 2. Branch & bound. 3. Grafos aleatórios. 4. Coloração de vértices. I. Título.
CDD 005

RODRIGO NOGUEIRA LIMA DAVID

HARD INSTANCES FOR THE MAXIMUM CLIQUE PROBLEM WITH HIGH PROBABILITY

Dissertation submitted to the Post-Graduation Program in Computer Science of the Center of Sciences of the Universidade Federal do Ceará, as a partial requirement for obtaining the title of Master in Computer Science. Concentration Area: Theory of Computation

Approved on: March 7th 2025

EXAMINATION BOARD

Prof. Dr. Victor Almeida Campos (Advisor)
Universidade Federal do Ceará (UFC)

Prof. Dr. Rudini Menezes Sampaio
Universidade Federal do Ceará (UFC)

Prof. Dr. Wladimir Araújo Tavares
Universidade Federal do Ceará (UFC)

Prof. Dr. Fabrício Siqueira Benevides
Universidade Federal do Ceará (UFC)

Prof. Dr. Guilherme Oliveira Mota
Universidade de São Paulo (USP)

ACKNOWLEDGEMENTS

I would like to first express my deepest gratitude to my family for their unwavering support and encouragement; specially to my mother, Sandra, for all the investment she made in me — far beyond financial — believing in my potential every step of the way. I also want to thank all the friends I made during my undergraduate and master's studies, especially Amanda, my girlfriend, who stood by me in the most difficult moments and was often responsible for the lighter ones.

I am grateful to everyone at ParGO for creating such a welcoming and stimulating environment, particularly Professor Victor for advising me throughout this work and since my undergraduate Scientific Initiation. I also thank Professors Fabrício, Guilherme, Rudini, and Wladimir for taking the time to be part of the examination committee and for providing constructive feedback on this work. A special thanks as well to Professor Manoel, who, although not my advisor during my master's, was during part of my undergraduate studies and, besides introducing me to ParGO, continued to guide me occasionally.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

_____ ... _____

Agradeço primeiramente à minha família pelo apoio e suporte constantes; especialmente à minha mãe, Sandra, por todo o investimento — muito mais do que financeiro — que fez em mim. Também, a todos os amigos que fiz na graduação e no mestrado, especialmente à Amanda, minha namorada, que esteve comigo nos momentos mais difíceis e foi muitas vezes responsável pelos mais leves.

Agradeço a todos do ParGO por criarem um ambiente tão acolhedor e estimulante, especialmente ao professor Victor por ter me orientado nesse trabalho e por já o fazer desde a minha iniciação científica na graduação. Também agradeço aos professores Fabrício, Guilherme, Rudini e Wladimir por cederem seus tempos participando da banca e fornecendo comentários construtivos para o trabalho. Um obrigado também ao professor Manoel, que embora não tenha sido meu orientador no mestrado, foi durante um período da minha graduação e, além de ter me introduzido ao ParGO, continuou me orientando informal e esporadicamente.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

"I, at any rate, am convinced that GOD does not throw dice."

(Albert Einstein)

"But still, it cannot be for us to tell HIM how HE is to run the universe."

(Niels Bohr)

RESUMO

O problema da CLIQUE MÁXIMA é um problema clássico de otimização em grafos com o objetivo de encontrar uma clique máxima em um grafo de entrada. A despeito da existência de diversos resultados teóricos de dificuldade, estudos empíricos sugerem que o problema costuma ser mais fácil do que o esperado. Nesse trabalho, exploramos uma classe de algoritmos *Branch & Bound* para resolver CLIQUE MÁXIMA e como eles tornam o problema mais tratável na prática. Nós abordamos a relação entre cliques e colorações próprias de vértices e determinamos com alta probabilidade o crescimento assintótico do número de colorações em grafos aleatórios. Usando este resultado junto a uma redução polinomial de coloração para clique da literatura, construímos novas instâncias de CLIQUE MÁXIMA e analisamo-las. Ademais, examinamos uma família de instâncias da literatura que induz tempo exponencial em uma subclasse de algoritmos *Branch & Bound* amplamente utilizada, que utiliza um limite superior cromático para podar ramos. Nós propomos um método de pré-processamento que habilita esses algoritmos a resolverem tais instâncias em tempo linear no tamanho delas. Além disso, introduzimos uma construção aleatorizada que produz grafos resistentes ao pré-processamento e que ainda exibem tempo de execução exponencial para esses algoritmos, mesmo caso o limite superior utilize o número cromático fracionário, que é uma cota mais apertada. Por fim, executamos testes computacionais para validar nossas análises.

Palavras-chave: clique máxima; branch & bound; grafos aleatórios; coloração de vértices.

ABSTRACT

The MAXIMUM CLIQUE problem is a classic graph-theoretical optimization problem with the objective of finding a maximum clique in a given input graph. Despite numerous theoretical hardness results, empirical studies suggest that the problem is often easier than expected. In this work, we explore a class of Branch & Bound algorithms for solving MAXIMUM CLIQUE and how they make the problem more tractable in practice. We approach the relationship between cliques and proper vertex colorings and derive the asymptotic growth of the number of colorings in random graphs with high probability. Using this result paired with a coloring-to-clique polynomial reduction in the literature, we generate new MAXIMUM CLIQUE instances and analyze them. Moreover, we examine a family of instances from the literature that induce exponential runtime on a widely adopted subclass of Branch & Bound algorithms that use a chromatic upper bound to prune branches. We propose a preprocessing method that enables these algorithms to solve such instances in linear time on their size. Furthermore, we introduce a randomized construction that produces graphs resistant to this preprocessing and still exhibit exponential runtime for these algorithms, even if the upper bound uses the fractional chromatic number instead, which is a tighter bound. Finally, we run some computational tests to validate our analyses.

Keywords: maximum clique; branch & bound; random graphs; vertex coloring.

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$\lg n$	Base 2 logarithm, i.e., $\lg n = \log_2 n$
$\ln n$	Natural logarithm, i.e., $\ln n = \log_e n$
$n!$	Factorial, i.e., $n! = 1 \cdot 2 \cdot \cdots \cdot n$
$n!!$	Double factorial, i.e., $n!! = 1 \cdot 3 \cdot 5 \cdots n$ if n is odd and $n!! = 2 \cdot 4 \cdot 6 \cdots n$ if n is even
$\binom{n}{k}$	Binominomial coefficient, i.e., $\binom{n}{k} = \frac{n!}{n!(n-k)!}$
$\exp(x)$	Exponential function, i.e., $\exp(x) = e^x$
$\mathbb{N}$	Set of natural numbers, i.e., $\mathbb{N} = \{1, 2, 3, \dots\}$
$\mathbb{R}$	Set of real numbers
2^X	Power set of the set X, i.e., the set of all its subsets
$\langle I \rangle$	Length of the binary string I
$1 \pm o(1)$	A term that is not constant, but is lower and upper bounded by constants

CONTENTS

1 INTRODUCTION

A clique in a graph is a set of vertices in which any two elements are adjacent. In the MAXIMUM CLIQUE problem, the objective is to find the largest clique in a given graph. This is a classic graph-theoretical optimization problem with applications in robotics [1, 2], computer vision [3, 4], distributed systems [5, 6], coding theory [7, 8, 9], bioinformatics [10, 11] and many other research areas.

Algorithms for MAXIMUM CLIQUE date as early as 1957 [12], while the first complexity results were published in 1972 [13] and mathematical programming formulations in 1974 [14]. In the '90s, the interest in this problem grew rapidly — partly due to its popularization and advances in the late '80s, but also because of the Second DIMACS Challenge, which had MAXIMUM CLIQUE as one of the three proposed problems — and many algorithmic results followed, such as [15, 16, 17, 18, 19]. To this day, clique algorithms remain an active area of research and many general combinatorial optimization techniques have been developed aiming to deal with increasingly larger instances for this problem [20, 21].

In this work, however, we shift the focus from the study of efficient algorithms to that of hard instances, as the problem seems to be much easier in practice than its theoretical worst case [22]. This has been done in the literature [23, 24, 25], but most of the work is heuristic, in the sense that the instances provided are said to be challenging because they exploit weaknesses of efficient algorithms and these claims are supported only by computational experiments.

To the best of our knowledge, only one such work appeals to empirical arguments *and* a formal proof of hardness [26]. Our aim is to build on this theoretical approach and provide other challenging graph constructions together with proofs that they are indeed hard

1.1 Hardness results

The MAXIMUM CLIQUE problem is NP-hard [13] and $n^{1-\varepsilon}$-inapproximable in polynomial time (unless P = NP) for any $\varepsilon > 0$, where n denotes the number of vertices in the input graph [27]. Its decision version consists in deciding, given an input graph G and an integer k, if G has a clique of size at least k. Besides being NP-complete, it is also W[1]-complete under the natural parameterization over k [28].

Although MAXIMUM CLIQUE is drawn intractable by this (non-exhaustive) list of theoretical hardness results, several authors report exact algorithms that are able to tackle large instances of practical interest for several application domains in reasonable time [15, 29, 20, 21]. This interesting contrast has been studied before [22, 30]. Carmo and Züge [22] approach a widely used class of Branch and Bound (B&B) algorithms and show that their time complexity is highly concentrated around the sub-exponential $n^{\Theta(\lg n)}$ growth rate in the $\mathcal{G}(n, p)$ model for any constant p by counting the number of cliques in such graphs.

This result gives an intuition on why the problem is often easier in practice, given that its worst case seems to be rare, but raises questions regarding the structure of such hard instances. If a randomly sampled graph is not expected to demand exponential time to be solved, what does a graph that comes close to the worst case look like?

1.2 Relation with Vertex Colorings

The MINIMUM VERTEX COLORING problem, in which the objective is to partition the vertices of a graph in the least number of parts in such a way that no adjacent vertices are on the same part, shares some connections with MAXIMUM CLIQUE. This is another classic NP-hard problem [13] and, moreover, it is also $n^{1-\varepsilon}$-inapproximable in polynomial time for any $\varepsilon > 0$ (unless P = NP), where n denotes the number of vertices in the input graph [27].

Because they are linked by a min-max inequality, vertex colorings and cliques tend to appear together in algorithms, being upper or lower bounds for each other. Moreover, in 1992, the second DIMACS Implementation Challenge was held to encourage the development of algorithmic results on three problems, two of them being MINIMUM VERTEX COLORING and MAXIMUM CLIQUE. Comparing the selected papers, the conclusion was that the coloring problem was much harder than the clique one [31]. In this work, we also explore some relations between cliques and vertex colorings.

We study the number of colorings in random graphs in a similar way to what was done with cliques to argue that enumerating the former is, indeed, harder than enumerating the latter, hence backing up the claim about the hardness disparity between the problems as most algorithms were already enumerative at the time (as were since the first algorithm for MAXIMUM CLIQUE and still are today). Furthermore, we adapt a reduction in the literature to provide nondeterministic instances with more cliques than a $\mathcal{G}(n, p)$ random graph for constant p and count the expected number of cliques in denser $\mathcal{G}(n, p)$ graphs that match the number of edges of these instances.

1.3 Algorithms with an upper bound based on the chromatic number

Even though the worst case for MAXIMUM CLIQUE is not expected to be solved in polynomial time, presenting instances that attend to this complexity in practice is a non-trivial issue. Lavnikevich [26] focus on an even more restricted class of algorithms — B&B with an upper bound based on the chromatic number, which are widely adopted and considered to be the best among state-of-the-art algorithms — and introduces a family of graphs with n vertices, for $n \equiv 0 \pmod 5$, that require $\Omega(2^{n/5})$ steps to be solved by any such algorithm. These instances, however, are artificial, in the sense that it would be very unlikely to find one of those graphs in a real problem and, besides that, their recognition is straightforward polynomial.

We argue they need not be explicitly recognized. We propose a simple preprocessing heuristic that enables algorithms to solve Lavnikevich's instances in linear time in their size (quadratic in n, as they have a high edge density) while still being useful for many other inputs. We also describe a randomized construction based on Lavnikevich's graphs that outputs an instance that still exhibits exponential time behavior, need not have a number of vertices that is strictly a multiple of 5 and is unaffected by the proposed preprocessing. We show further that it still demands exponential time even when the algorithm uses a specific infra-chromatic upper bound, namely the fractional chromatic number.

1.4 Structure of this work

Chapter 2 lays out some basic definitions on graph theory, computational complexity, combinatorial optimization and (discrete) probability theory. Chapter 3 introduces the B&B algorithms that we study throughout the text and presents some of their basic properties. Chapter 4 reviews graph instances in the literature that concern the MAXIMUM CLIQUE problem. Chapter 5 presents the study of vertex colorings and their connections with cliques. Chapter 6 focus on a more particular class of B&B algorithms, which use an upper bound based on the chromatic number. Chapter 7 describes computational experiments regarding the algorithms and instances in the text. Finally, Chapter 8 concludes the work and approaches future research topics.

2 PRELIMINARIES

We begin with a few definitions and basic results that are essential to this work.

2.1 Graph Theory

We follow conventions adopted by reference books [32, 33]. A *graph* G is defined by a triple (V, E, ψ) consisting of a set of vertices, a set of edges and an incidence function that maps each edge into two (not necessarily distinct) vertices called *endpoints*, respectively. We write $V(G)$ (resp. $E(G)$) for the set of vertices (resp. edges) of a graph G. The *order* (resp. *size*) of a graph G, written $v(G)$ (resp. $e(G)$), is the number of vertices (resp. edges) in G. An edge that is mapped into two equal vertices is called a *loop* and edges that are mapped into the same pair of vertices are said to be *parallel* (or *multiple edges*). A graph is said to be *simple* if it has neither loops nor multiple edges. All graphs in this work are presumed to be simple, unless stated otherwise. The *edge density* of a graph G is the ratio $e(G)/\binom{n}{2}$ and G is said to be *dense* if this quantity is close to 1 or *sparse* if it is close to 0.

We abuse the notation and refer to an edge by its image through ψ, but there should be no major issues with this practice if the graph is simple. When vertices u and v of a graph G are endpoints of some edge, we write $uv \in E(G)$ and say that u and v are *adjacent* (or *neighbors*). We write $N(v)$ for the set of all neighbors of a vertex v. Two (simple) graphs G and H are said to be *isomorphic* if there is a bijection $f\colon V(G) \to V(H)$ such that $uv \in E(G)$ if, and only if, $f(u)f(v) \in E(H)$ and we write $G \cong H$ do indicate this.

The *degree* of a vertex v, denoted by $d(v)$, is the number of edges in which v is an endpoint. In a simple graph, $d(v) = |N(v)|$. We denote by $\delta(G)$ and $\Delta(G)$ the minimum and maximum degrees of G (taken over all its vertices), respectively. If all vertices in a graph G have degree k, we say that G is *k-regular*.

A *clique* (resp. *independent set*) in a graph G is a set of vertices in which any two of them are adjacent (resp. no two of them are adjacent). The sizes of the largest clique and largest independent set in G are denoted by $\omega(G)$ and $\alpha(G)$, respectively. A *matching* in G is a set of edges that share no endpoints. The vertices that are endpoints to some edge in a matching M are said to be *saturated* by M, the others are said to be *unsaturated*. A *perfect* matching saturates all vertices in the graph.

A (proper) *vertex coloring* of G is a function that maps each vertex into a color in such a way that adjacent vertices are mapped into different colors (for all our purposes a color is an integer). A *k-coloring* is a vertex coloring that uses exactly k colors. The set of all vertices with some given color is called a *color class* and we sometimes refer to a coloring by its color classes. The fewest colors needed by any vertex coloring of a graph G is the *chromatic number* of G, written $\chi(G)$. Given two graphs G and H such that $v(G) \equiv 0 \pmod{v(H)}$, an *$H$-factor* of G is a collection of $v(G)/v(H)$ copies of H whose vertex sets partition $V(G)$.

A *subgraph* H of a graph G is a graph where $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$ and H's incidence function is a restriction of G's incidence function and we write $H \subseteq G$ to indicate this. Given two graphs G and H, we say G is *H-free* if no subgraph of G is isomorphic to H. A subgraph H *spans* G if $V(H) = V(G)$, and we call H a *spanning* subgraph. A set $S \subseteq V(G)$ *induces* a subgraph of G, which we denote by $G[S]$, such that $V(G[S]) = S$ and $E(G[S]) = \{uv \mid u, v \in S \text{ and } uv \in E(G)\}$. If a subgraph H is equal to $G[S]$ for some set S, we call H an *induced* subgraph.

An *orientation* of a graph G is a function $\sigma\colon E(G) \to V(G) \times V(G)$ that maps an edge uv into an *arc* which is either (u, v) or (v, u) (the order of the vertices now matters). An *oriented graph* is a pair (G, σ) of a graph and an orientation. It is common to represent an oriented graph by drawing G and for each edge uv we draw an arrow from u to v if $\sigma(uv) = (u, v)$ and vice-versa. The *indegree* $d^-(v)$ (resp. *outdegree* $d^+(v)$) of a vertex v in a oriented graph (G, σ) is the number of edges uv such that $\sigma(uv) = (u, v)$ (resp. $\sigma(uv) = (v, u)$).

A *path* (resp. *cycle*) on n vertices, denoted by P_n (resp. C_n), is a graph whose vertices can be linearly (resp. cyclically) ordered and two vertices are adjacent in the graph if they are consecutive in the ordering. An *oriented path* (resp. *oriented cycle*) on n vertices is an oriented graph whose vertices can be linearly (resp. cyclically) ordered and (u, v) is an arc if v comes immediately after u in the order. A *complete* graph on n vertices, denoted by K_n, is a graph in which every two nodes are adjacent. A complete graph on 3 vertices is called a *triangle*. An *empty* graph is a graph that has no edges. The *trivial* graph is the empty graph with only one vertex. A *bipartite* graph is a graph in which its vertex set can be partitioned into two independent sets. More generally, a *k-partite* graph (or simply a *multipartite* graph) is a graph in which its vertex set can be partitioned into k independent sets.

A graph G with at least two vertices is said to be *balanced* if

$$\frac{e(G)}{v(G)-1} = \max\left\{\frac{e(H)}{v(H)-1} \;\middle|\; H \text{ is a proper induced subgraph with at least two vertices}\right\}.$$

Moreover, if

$$\frac{e(G)}{v(G)-1} > \frac{e(H)}{v(H)-1}, \qquad \text{for all proper induced subgraph } H \text{ with at least two vertices,}$$

we say G is *strictly balanced*. Complete graphs, for instance, are strictly balanced, while empty graphs are not. In particular, K_2 is strictly balanced by vacuity.

A graph G is said to be *connected* if between any two vertices u and v there is a path in G starting in u and ending in v, G is *disconnected* otherwise. A graph G is said to be *k-connected* if the removal of at most $k-1$ vertices cannot disconnect the graph. A *component* of a graph is a maximal connected subgraph.

The *complement* of a graph G, written $\overline{G}$, is the graph with the same set of vertices of G in which any two vertices are adjacent if they are not adjacent in G. The *line graph* of a graph G, written $L(G)$, is a graph with $V(L(G)) = E(G)$ such that two vertices of $L(G)$ are adjacent

if their respective edges in G share an endpoint. The *disjoint union* (resp. *join*) of two graph G and H, denoted by $G + H$ (resp. $G \vee H$), is the graph whose vertex set is $V(G) \cup V(H)$ and edge set is $E(G) \cup E(H)$ (resp. $E(G) \cup E(H) \cup \{uv \mid u \in V(G), v \in V(H)\}$). We assume that the vertex sets of G and H are always disjoint, and so are the edge sets, in these operations. The *union* between two graphs G and H with non-disjoint sets of vertices is denoted by $G \cup H$ and is defined analogously to the disjoint union.

2.2 Computational Complexity

We again adopt conventions used by reference books [34, 35]. Given two functions $f, g: \mathbb{N} \to \mathbb{R}$, we say $f(n) = \mathcal{O}(g(n))$ (resp. $f(n) = \Omega(g(n))$) if there exist a constant $c > 0$ and an integer n_0 such that $|f(n)| \leq c|g(n)|$ (resp. $|f(n)| \geq c|g(n)|$) for all $n \geq n_0$. Similarly, we say $f(n) = o(g(n))$ (resp. $f(n) = \omega(g(n))$) if for every $\varepsilon > 0$ there is an integer n_0 such that $|f(n)| \leq \varepsilon|g(n)|$ (resp. $|f(n)| \geq \varepsilon|g(n)|$) for every $n \geq n_0$. We also say that $f(n) = \Theta(g(n))$ if $f(n) = \mathcal{O}(g(n))$ and $f(n) = \Omega(g(n))$.

We denote by $\text{poly}(n)$ the set of all functions $f: \mathbb{N} \to \mathbb{R}$ such that $f(n) = \mathcal{O}(n^k)$ for some $k \in \mathbb{N}$. We say a function $f: \mathbb{N} \to \mathbb{R}$ has *exponential growth* (or simply is *exponential*) if $f(n) = 2^{\Omega(n^\varepsilon)}$ for some $\varepsilon > 0$. If $f(n) = 2^{o(n^\varepsilon)}$ for all $\varepsilon > 0$, we say f is *subexponential*.

A (combinatorial) *optimization problem* is defined by a set of binary strings called *instances*, a set of binary strings called *feasible solutions* for each instance, an *objective function*, which maps a pair consisting of an instance and a feasible solutions into a rational number called *objective value*, and an indication that specifies whether the problem is a *maximization* or a *minimization* one. For each instance, the objective in a maximization (resp. minimization) problem is to find a feasible solution that maximizes (resp. minimizes) the objective function (among all possible feasible solutions).

A *decision problem* is defined by a set of binary strings called *instances* that can be partitioned into two others, the set of *positive instances* and the set of *negative instances*, such that for each instance the objective is to decide whether it is a positive or a negative instance. For each minimization problem (resp. maximization problem), there is an *associated decision problem* (or simply its *decision version*) in which its set of instances consists of pairs formed by an instance of the optimization problem and a rational k and the positive instances are those that admit some feasible solution with objective value at most k (resp. at least k).

We denote by NP the class of decision problems Π that admit an algorithm V called *polynomial verifier* that takes as inputs an instance I of Π and a binary string C such that $\langle C \rangle = \text{poly}(\langle I \rangle)$, runs in $\text{poly}(\langle I \rangle)$ time and satisfies $V(I, C) = 1$ for some C if I is a positive instance or $V(I, C) = 0$ for all C, otherwise. A *polynomial reduction* from a decision problem Π to another decision problem Π_0 is an algorithm that takes as input an instance I of Π and outputs in $\text{poly}(\langle I \rangle)$ time an instance I_0 of Π_0 that is a positive instance for Π_0 if and only if I is a positive instance for Π. A decision problem Π_0 is said to be NP-*hard* if for each problem

$\Pi \in$ NP there exists a polynomial reduction from Π to Π_0. A decision problem Π is said to be NP-*complete* if $\Pi \in$ NP and Π is NP-hard. We say an optimization problem is NP-hard if its decision version is NP-hard.

We denote by P the class of all decision problems Π that admit an algorithm that takes as input an instance I of Π and in poly($\langle I \rangle$) time outputs 1 if I is a positive instance or 0 otherwise. It follows from the definition that P $\subseteq$ NP, because any algorithm that decides a problem $\Pi \in$ P in polynomial time can be made a polynomial verifier by ignoring the auxiliary binary string and solving the problem directly. It is widely believed that P $\neq$ NP, although no proof has ever been found (but neither has a polynomial time algorithm that solves any NP-hard problem, which would imply that P = NP).

2.3 Combinatorial Optimization

The definitions here follow conventions adopted by books such as the one due to Wolsey [36]. A *linear programming problem* is an optimization problem where the objective function and the constraints on the instances can be represented by linear functions and inequalities. A (linear) *integer programming problem* is a linear programming problem with additional constraints of integrality for some variables. The *linear relaxation* of an integer programming problem is a linear programming problem obtained by dropping the integrality constraints on the original problem.

A classic integer programming model for Maximum Clique in a graph G is

$$\begin{aligned} \max & \sum_{v \in V(G)} y_v \\ \text{s.t:} & \sum_{v \in I} y_v \leq 1, \qquad \forall I \in \mathcal{I}(G) \\ & y_v \in \{0,1\}, \qquad \forall v \in V(G) \end{aligned} \tag{2.1}$$

where y_v is an indicator variable with value 1 if the vertex v is chosen in the clique or 0 otherwise and $\mathcal{I}(G)$ is the set of all independent sets of the graph G. Even though adding the constraints $\sum_{v \in I} y_v \leq 1$ only when $|I| = 2$ yields a more compact model, adding for all I has the advantage that its linear relaxation gives a specific upper bound on $\omega(G)$, which we call the *fractional clique number* of G, denoted by $\omega_f(G)$.

Another classic integer programming model, now for the Minimum Vertex Coloring problem in a graph G, is

$$\begin{aligned} \min & \sum_{I \in \mathcal{I}(G)} x_I \\ \text{s.t:} & \sum_{\substack{v \in I, \\ I \in \mathcal{I}(G)}} x_I \geq 1, \qquad \forall v \in V(G) \\ & x_I \in \{0,1\}, \qquad \forall I \in \mathcal{I}(G) \end{aligned} \tag{2.2}$$

where x_I is an indicator variable with value 1 if the independent set I is a color class in the coloring or 0 otherwise. The linear relaxation of this model gives a lower bound on $\chi(G)$, which we call the *fractional chromatic number* of G, denoted by $\chi_f(G)$. The linear relaxations of the models above are said to be *dual* to each other and satisfy, among many other properties, $\omega_f(G) = \chi_f(G)$.

A very common approach when solving optimization problems is to use *Branch and Bound* algorithms, which enumerate feasible solutions by exhaustively trying all possibilities. This is achieved by splitting the set of feasible solutions, analyzing smaller subsets and comparing the objective values in order to find an optimal solution. To speed up further, the algorithm keeps lower and upper bounds on the quality of the best solution of a subset to discard subsets that cannot improve these bound (and, hence, cannot contain an optimal solution). In a maximization problem, for instance, a lower bound can be any feasible solution, found either by some previous state during the enumeration or by applying some heuristic, while an upper bound can be achieved by some relaxation of the problem or by a valid inequality specific to the problem.

A B&B algorithm solves repeatedly a *subproblem* that is similar to the original problem being solved, it is usually some variation that generalizes the inputs in order to make the covering of the feasible solutions set easier. Once the algorithm takes on a subproblem, it checks if one of its feasible solutions can have a better objective value than the best solution found so far. If no such solution exists, the subproblem is discarded as any possible solution given by it would be suboptimal. Otherwise, the algorithm *branches* the subproblem into others, following some specific rule, and solves them recursively to obtain the best solution for the current subproblem. This is repeated until all feasible solution have been either enumerated or safely discarded.

The algorithm's execution is usually associated with a decision tree that keeps track of all the decisions made — the *search tree*. The root of the tree is the original instance and any given node has as children nodes associated with the subproblems generated by the branching rule. When the algorithm decides to discard a subproblem, we say its associated node's subtree is *pruned*, as all its descendant nodes corresponding to subproblems are not explored and, thus, are not in the tree.

2.4 Probability Theory

The definitions here also follow loosely those in reference books [37, 38]. A (discrete) *probability space* is defined by a pair $(\Omega, \mathbb{P})$, where Ω is a countable set called the *sample space*, whose elements are called *outcomes* and $\mathbb{P}\colon 2^{\Omega} \to [0, 1]$ is a function that maps an *event*, which is a subset of Ω, into the probability that it happens. The function $\mathbb{P}$ satisfies $\mathbb{P}[\Omega] = 1$ and for any $\mathcal{F} \subseteq \Omega$ such that all its elements are pairwise disjoint,

$$\mathbb{P}\left[\bigcup_{A \in \mathcal{F}} A\right] = \sum_{A \in \mathcal{F}} \mathbb{P}[A].$$

We write $\overline{A}$ for the *complement* of an event A, i.e., the event $\Omega \setminus A$. The events A_1, $A_2, \ldots, A_k$ are said to be *independent* if

$$\mathbb{P}\left[\bigcap_{i \in I} A_i\right] = \prod_{i \in I} \mathbb{P}[A_i], \quad \forall I \subseteq \{1, \ldots, k\}.$$

The *conditional probability* of an event A happening given that event B happened (assuming $\mathbb{P}[B] > 0$) is defined as

$$\mathbb{P}[A \mid B] = \frac{\mathbb{P}[A \cap B]}{\mathbb{P}[B]}.$$

We say a sequence of events $(A_n)_{n \in \mathbb{N}}$ occurs *asymptotically almost surely* (a.a.s.) if $\mathbb{P}[A_n] \to 1$ when $n \to \infty$. In particular, if $\mathbb{P}[A_n] \geq 1 - n^{-\Omega(1)}$, we say A_n happens *with high probability* (w.h.p.). An event happening w.h.p. implies it happening a.a.s., but the converse need not hold.

The *monotonicity* property of $\mathbb{P}$ states that if two events A and B satisfy $A \subseteq B$, then $\mathbb{P}[A] \leq \mathbb{P}[B]$. Given events $A_1, A_2, \ldots, A_n$, the *union bound* is a common upper bound to the probability of their union, given by

$$\mathbb{P}[A_1 \cup A_2 \cup \cdots \cup A_n] \leq \sum_{i=1}^{n} \mathbb{P}[A_i],$$

where equality occurs when the events are disjoint.

A *random variable* X on a sample space Ω is a function $X \colon \Omega \to \mathbb{R}$. We use mainly discrete random variables, which take on only a countable number of values. The set of events $\{\omega \in \Omega \mid X(\omega) = a\}$ is denoted by $\{X = a\}$ and we define

$$\mathbb{P}[X = a] = \sum_{\substack{\omega \in \Omega, \\ X(\omega) = a}} \mathbb{P}[\omega].$$

The random variables $X_1, X_2, \ldots, X_k$ are said to be *independent* if for any $a_1, a_2, \ldots, a_k$

$$\mathbb{P}\left[\bigcap_{i \in I} \{X_i = a_i\}\right] = \prod_{i \in I} \mathbb{P}[X_i = a_i], \quad \forall I \subseteq \{1, \ldots, k\}.$$

The *expected value* (or *expectation*) of a discrete random variable X, denoted by $\mathbb{E}[X]$, is given by

$$\mathbb{E}[X] = \sum_{\omega \in \Omega} X(\omega)\mathbb{P}[\omega].$$

Particularly, if X is always a non-negative integer, we write

$$\mathbb{E}[X] = \sum_{k \geq 0} k\mathbb{P}[X = k].$$

The expected value is a linear operator, i.e., for any $k \in \mathbb{N}$, $a_1, a_2, \ldots, a_k \in \mathbb{R}$ and random variables $X_1, X_2, \ldots, X_k$,

$$\mathbb{E}[a_1X_1 + a_2X_2 + \cdots + a_kX_k] = a_1\mathbb{E}[X_1] + a_2\mathbb{E}[X_2] + \cdots + a_k\mathbb{E}[X_k].$$

The *Markov inequality* states that any non-negative random variable X satisfies

$$\mathbb{P}[X \geq t\mathbb{E}[X]] \leq \frac{1}{t}, \qquad \text{for all } t > 0.$$

A random variable X is said to follow an *uniform* distribution over the set $\{a, a+1, \ldots, b\}$ if

$$\mathbb{P}[X = k] = \begin{cases} 1/(b-a+1), & \text{if } k \in \{a, a+1, \ldots, b\} \\ 0, & \text{otherwise.} \end{cases}$$

If X is a uniform random variable, then $\mathbb{E}[X] = (a+b)/2$. A random variable X is said to follow a *geometric* distribution with parameter $p \in (0, 1]$ if

$$\mathbb{P}[X = k] = (1-p)^{k-1}p, \qquad \text{for any } k \in \mathbb{N}.$$

If X is a geometric random variable, then $\mathbb{E}[X] = 1/p$.

When Ω is a set of graphs, we say that the outcomes are *random graphs*. We focus mainly on two models. In the first, denoted by $\mathcal{G}(n, p)$, the outcomes are graphs on n vertices where each edge appears independently of others with probability p (usually as a function of n) and, thus, the probability of the outcome being a fixed graph G is equal to

$$p^{e(G)}(1-p)^{\binom{n}{2}-e(G)}.$$

In the second, denoted by $\mathcal{G}(n, m)$, the outcomes are graphs on n vertices, m edges and all such graphs are equaly likely to be sampled, hence, the probability of the outcome being a fixed graph G is

$$\binom{\binom{n}{2}}{m}^{-1}$$

if G has n vertices and m edges or 0, otherwise. If a graph G belongs to the $\mathcal{G}(n, p)$ (resp. $\mathcal{G}(n, m)$) model, we write $G \sim \mathcal{G}(n, p)$ (resp. $G \sim \mathcal{G}(n, m)$).

2.5 Useful definitions, bounds and asymptotics

The *Lambert function* $W\colon [0, +\infty) \to \mathbb{R}$ satisfies the functional equation

$$W(x)\exp(W(x)) = x, \qquad \forall x \geq 0. \tag{2.3}$$

Equation (2.3) has a real solution for all $x \geq -1/e$, but we define $W(x)$ only when $x \geq 0$ for practical purposes, as the real solution is unique if $x \geq 0$. We refer the reader to a survey due to Corless et al. [39] for a thorough introduction to some properties and uses of this function. In particular, Hoorfar and Hassani [40] show that

$$W(z) = \ln z - \ln\ln z + \Theta(\ln\ln z/\ln z). \tag{2.4}$$

The *exponential function* $\exp\colon \mathbb{R} \to \mathbb{R}$ is defined as

$$\exp(x) = \lim_{n\to+\infty} \left(1 + \frac{x}{n}\right)^n.$$

Using the Binomial Theorem, we can rewrite this in terms of an infinite series, namely

$$\exp(x) = \sum_{n\geq 0} \frac{x^n}{n!},$$

from whence follows that

$$\exp(k) \geq \frac{k^k}{k!}, \qquad \forall k \in \mathbb{N}. \tag{2.5}$$

Moreover, by the convexity of $x \mapsto \exp(x)$, we get $\exp(x) \geq 1 + x$, as $x \mapsto 1 + x$ is the tangent line to its graph on $x = 0$, hence,

$$\exp(x) \geq 1 + x \quad \text{and} \quad \exp(-x) \geq 1 - x, \qquad \forall x \geq 0. \tag{2.6}$$

A simple lower bound for the binomial coefficient $\binom{n}{k}$ is given by

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n}{k} \cdot \frac{n-1}{k-1} \cdots \frac{n-k+1}{1} \geq \left(\frac{n}{k}\right)^k, \tag{2.7}$$

because for any $k \leq n$ we have

$$\frac{i}{n} \leq \frac{i}{k} \iff 1 - \frac{i}{n} \geq 1 - \frac{i}{k} \iff \frac{n-i}{k-i} \geq \frac{n}{k}.$$

A common upper bound can be derived using (2.5), as

$$\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k!} \leq \left(\frac{en}{k}\right)^k. \tag{2.8}$$

3 A BRANCH AND BOUND FRAMEWORK

B&B methods are widely regarded as the most efficient way to address not only MAXIMUM CLIQUE, but also several other NP-hard problems [41]. In this chapter, we study a specific family of such procedures, whose main idea — to branch using *pivot vertices* — is due to Bron and Kerbosch [42]. Although first proposed in 1973, this branching rule is still used in most state-of-the-art algorithms. We discuss some of its properties and how they help us understand the contrast between theoretical hardness results and the abundance of efficient algorithms for MAXIMUM CLIQUE.

3.1 Basic structure of the algorithm

First, we define the instances of the subproblem that the algorithm solves.

Definition 3.1. A *clique subinstance* of a graph G (or simply a *subinstance*) is a pair (Q, R) of disjoints subsets of $V(G)$ where Q is a clique and $Q \subseteq N(u)$, for each $u \in R$. In each subinstance (Q, R), the objective is to find the largest clique Q^* of G such that $Q \subseteq Q^* \subseteq Q \cup R$. An instance is *solved* if $R = \emptyset$.

An instance now comes with a initial clique Q and the objective of the subproblem is to find the largest clique containing Q by examining only vertices in R. Notice that an instance of the original problem MAXIMUM CLIQUE with input graph G corresponds to the subinstance $(\emptyset, V(G))$.

The *branching* step defines how any B&B method divides an instance into "smaller" ones — its *children*. In order to branch a clique subinstance, the algorithm will consider two cases. Intuitively, for any $v \in R$, if Q^* is the largest clique containing Q, then either $v \in Q^*$ or $v \notin Q^*$ and we will enumerate the possibilities as follows. If a subinstance (Q, R) of G is not already solved, then $R \neq \emptyset$, a *pivot* vertex $v \in R$ is chosen and this subinstance branches into two others:

1. $(Q \cup \{v\}, R \cap N(v))$, which considers all cliques that contain v (and do not contain any vertices that are not adjacent to v);

2. $(Q, R \setminus \{v\})$, which considers all cliques that *do not* contain v.

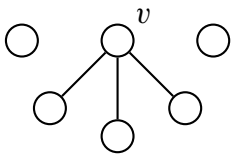
We do not specify how the algorithm chooses pivot vertices, but many implementations use degree-based [15, 43] or coloring-based [44, 45] criteria.

One of the key steps in any B&B method is the *bound* phase, where the algorithm computes an upper bound (in a maximization problem, or a lower bound in a minimization one) on the value of the best solution available to some instance and decides whether or not to branch. We do not define which bound rule is to be used yet, but the most common ones are coloring-based (see Chapter 6).

We call any algorithm that implements this scheme a *standard algorithm*. All results herein apply to any standard algorithm, regardless of its pivot choices and upper bounds. A standard algorithm that does not have a bounding rule simply enumerates all possible cliques. Figure 1 illustrates the branching of a subinstance in a standard algorithm.

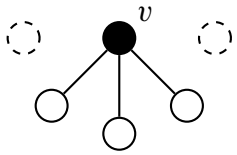
Figure 1 – A branching step on a subinstance (Q, R) with pivot v on a standard algorithm.

(a) The parent subinstance (Q, R) and the pivot v among vertices of R.



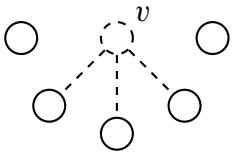
(Q, R)

(b) The left child of (Q, R), obtained by adding v to the current clique and removing from R vertices non-adjacent to v.



$(Q \cup \{v\}, R \cap N(v))$

(c) The right child of (Q, R), obtained by removing v from R without adding it to the current clique.



$(Q, R \setminus \{v\})$

Source: prepared by the author.

The set of subinstances considered by some execution of a B&B algorithm naturally induces a tree. For standard algorithms, its structure depends on the choice rule for pivots and the upper bound used. The root node of this tree is the subinstance $(\emptyset, V(G))$ and the children of a node are given by its children subinstances according to the branching rule. If a node does not branch, either because it is solved or because the algorithm pruned its children due to the upper bound, it is a leaf. We now define a similar tree structure, but that is associated with graphs instead of algorithms.

Definition 3.2. Given a graph G, a *clique search tree* T of G is a binary tree such that:

1. The root of T is the subinstance $(\emptyset, V(G))$;
2. The leaves of T are all subinstances in which $R = \emptyset$;
3. The left and right children of an internal node (Q, R) are $(Q \cup \{v\}, R \cap N(v))$ and $(Q, R \setminus \{v\})$, for some $v \in R$, respectively.

A clique search tree of a graph G can be seen as the result of an execution of a standard algorithm with no bounding rule. A graph can have many clique search trees, as they

are defined by the pivot choices, but they all have the same size (see Proposition 3.3). We say an execution $\mathcal{E}$ of a standard algorithm is *contained* in a clique search tree T of a graph G if the subinstances analyzed in $\mathcal{E}$ induce a connected subgraph in T, this subgraph contains the root of T and every pivot choice is the same in any subinstance of $\mathcal{E}$ and its equivalent node in T. In the next section, we describe how to analyze standard algorithms through clique search trees.

3.2 Some properties of standard algorithms

We consider clique search trees as a description of a worst-case scenario for any standard algorithm. The following proposition gives an intuition as to why this is true.

Proposition 3.3 (Carmo and Züge [22]). *Let G be a graph and $\mathcal{C}$ be the set of all its cliques. If T is a clique search tree of G, then T has $2|\mathcal{C}| - 1$ nodes. Furthermore, each execution of a standard algorithm for* Maximum Clique *on G is contained in some clique search tree of G.*

Indeed, by Proposition 3.3, the size of a clique search tree is an upper bound on the number of instances considered by any standard algorithm. Therefore, if the clique search trees are not too large, standard algorithms need not evaluate too many nodes. Carmo and Züge [22] approach the gap between theoretical and empirical hardness results regarding Maximum Clique by analyzing the average behavior of B&B algorithms through clique search trees.

Lemma 3.4 (See e.g. Carmo and Züge [22]). *For any $n \in \mathbb{N}$ and constant $p \in (0,1)$, if $G \sim \mathcal{G}(n,p)$, then the average number of cliques in G is at most $n^{2+c_p \lg n}$, where $c_p = -1/\lg p$.*

In other words, the expected number of cliques in a random graph (for constant p) is $n^{\mathcal{O}(\lg n)}$. Now, as the size of any clique tree grows at most as fast as the number of cliques in the graph, their size is, on average, subexponential. This explains why standard algorithms seem to be much faster on average than what they are expected to be in the worst case, for if the time to process a single node is subexponential, the final execution time is still expected to be subexponential and far from the worst case.

Chvátal [46] defines a structure that is similar to a clique search tree. The *f-driven* tree (the "f" stands for a function that selects pivots and, thus, defines the structure of the tree) is a binary tree whose nodes are subinstances for Maximum Independent Set and the children of a node are defined in an analogous way to those of a clique search tree node. Pittel [47] proves that the size of a f-driven tree on $\mathcal{G}(n,p)$ graphs for constant p is subexponential w.h.p. and Carmo and Züge [22] point out that the same could be argued for clique search trees.

Theorem 3.5 (Pittel [47], Carmo and Züge [22]). *For any $n \in \mathbb{N}$, constant $p \in (0,1)$ and $\varepsilon > 0$, if $G \sim \mathcal{G}(n,p)$ and T is a clique search tree of G, then*

$$\mathbb{P}(n^{(0.25-\varepsilon)c_p \lg n} \leq |T| \leq n^{(0.5+\varepsilon)c_p \lg n}) \geq 1 - \exp(-c \ln^2 n),$$

where c is a positive value depending on ε and $c_p = -1/\lg p$.

Theorem 3.5 strengthens the explanation of the easiness in practice of MAXIMUM CLIQUE. Not only the expected number of cliques in a $\mathcal{G}(n, p)$ random graph for constant p is $n^{\mathcal{O}(\lg n)}$, but the number of cliques itself is $n^{\Theta(\lg n)}$ w.h.p. So even if a standard algorithm uses no upper bound, it still is very unlikely that it takes exponential time to solve an instance, provided it uses at most subexponential time processing a subinstance. In other words, although the worst case is indeed exponential, it is rare enough not to be too impactful.

These results raise the following question: "If almost always a maximum clique can be found in subexponential time in $\mathcal{G}(n, p)$ with constant p, when does it take more time to find it?" We investigate some answers in the next chapter.

4 INSTANCES IN THE LITERATURE

In this chapter, we discuss some clique-related results that can be extended to graph constructions. We split them into two categories: theoretical and practical. The first category includes constructions designed to achieve specific properties, such as maximizing the number of cliques or maximal cliques in graphs of a given size and order, with formal proofs confirming them. The second one, on the other hand, focus on instances that were conceived as inherently hard for MAXIMUM CLIQUE, following heuristic rules without any formal proof of hardness, relying instead in empirical tests to display their difficulty.

4.1 Theoretical instances

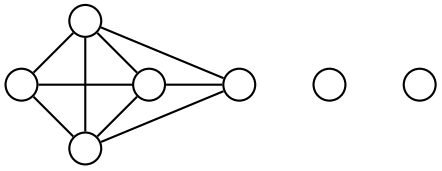
When looking for hard instances to MAXIMUM CLIQUE, a natural first approach is to find graphs that at least exhibit an asymptotically higher number of cliques compared to $\mathcal{G}(n, p)$ with constant p. In this sense, a complete graph on n vertices is best possible with 2^n cliques and, indeed, any standard algorithm without an upper bound will take exponential time to solve these instances. However, it is easy to find a maximum clique in K_n (just find any maximal clique) and this can be exploited by algorithms to quickly prune all branches using common upper bound rules (e.g. vertex coloring heuristics) and terminate.

If a high edge density is to be avoided, Wood [48] proves that the highest possible number of cliques in a graph with n vertices and m edges is $2^d + 2^\ell + n - d - 1$, where d and ℓ are defined as the unique integer solution to

$$m = \binom{d}{2} + \ell, \qquad 0 \leq \ell \leq d - 1,$$

and shows instances that attend to this number. Such an instance consists of a graph with n vertices which has the largest possible complete subgraph with at most m edges (using d vertices) and all the remaining edges have one common endpoint outside the large clique and one endpoint inside the large clique, any other vertex is isolated. Figure 2 gives an example.

Figure 2 – A graph with 7 vertices, 9 edges and $2^4 + 2^3 + 7 - 4 - 1 = 26$ cliques, the maximum possible for any graph with these parameters, as described by Wood [48].

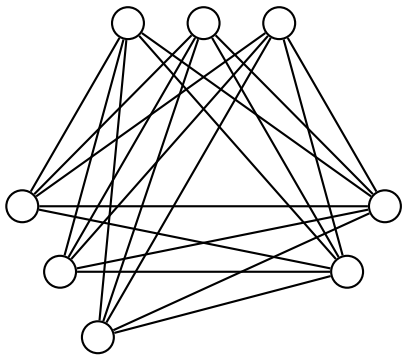


Source: prepared by the author.

The author also shows the maximum number of cliques in graphs with other restrictions, such as both number of edges and maximum degree fixed, fixed number of edges and d-degenerate, planar and planar with a fixed number of edges.

If many maximal cliques are desired, Miller and Müller [49] and Moon and Moser [50] independently show that any graph with n vertices has at most $\mathcal{O}(3^{n/3})$ maximal cliques and the latter authors prove that graphs that attain this value are complete multipartite graphs where each part has size 2 or 3 and as many parts as possible have size 3. Figure 3 gives an example.

Figure 3 – A graph with 8 vertices and $2 \cdot 3^{(8-2)/3} = 18$ maximal cliques, the maximum possible for any graph with this many vertices, as described by Moon and Moser [50].



Source: prepared by the author.

These graphs are a special case of the more general Turán graphs $T(n, k)$ due to Turán [51], which have the maximum number of edges among graphs on n vertices without cliques of size $k + 1$. Besides that, Zykov [52] shows that they also have the most number of cliques of size k among any n vertex graph without cliques of size $k + 1$.

These are graphs that combine a high number of cliques with other desired properties (fixed edge density or planarity, for example), but to the best of our knowledge have not been benchmarked with standard algorithms.

As a final note, we mention the family of graphs due to Lavnikevich [26]. Its difficulty is supported both by a formal proof on a lower bound for the solving time regarding a specific type of algorithms and by computational tests using an implementation of such algorithms. This construction will be discussed in more detail in Chapter 6 due to its unique nature — being theoretically *and* empirically hard.

4.2 Practical instances

The DIMACS benchmark set [23] contains 66 instances that arise from many contexts. They are split into nine families and were proposed in the Second DIMACS Implementation Challenge in 1992, but remain to this day as one of the main sources of hard inputs to clique algorithms. Random graphs in the $\mathcal{G}(n, p)$ model, usually with a constant p, are also widely used. These are, however, much easier instances in practice (and in theory, as we have seen in Chapter 3) and tend to serve only as a control group to be tested against some other graph construction with similar order and size.

The first family of instances, denoted by “CFat”, consists of 7 graphs originated by fault diagnosis problems on distributed systems [5]. The second and third, denoted by “Johnson” and “Hamming”, contain 4 and 6 graphs, respectively, that come from problems in coding theory. The fourth one, with 3 instances, is denoted by “Kel” and is based on Keller’s conjecture on tiling using hypercubes [53]. The fifth, denoted by “San”, has 11 graphs and is originated by problems concerning vertex covers [54]. The sixth, denoted by “SanR”, contains 4 random graphs with sizes similar to those in the fifth family. The seventh, denoted by “Brock”, consists of 12 instances that attempt to hide large cliques in quasi-random graphs, where the expected clique size is much smaller [25]. The eigth, with 15 graphs, is denoted by “PHat” and is given by a generalization of random graphs that has more parameters, a wider node degree spread and larger clique sizes [55]. The ninth and last, denoted by “Stein”, contains 4 graphs which correspond to instances of a clique translation of the set covering formulation of a problem concerning Steiner Triples [56].

The idea of hiding cliques in quasi-random graphs, in particular, is widely adopted. The “Brock” family of graphs, for example, focus on defeating a greedy heuristic that excels on instances generated by a similar model due to Kučera [57]. The graphs are generated by using two probabilities functions to decide if an edge should be added or not in the final graph, which enables them to control the hardness for these greedy algorithms to detect a clique while still being able to keep some desired average edge density.

A somewhat similar strategy is also used in the BHOSLIB [24], which is another repository of hard instances for Maximum Clique (but also for vertex cover and coloring problems) which is frequently used, although not as much as DIMACS. The main idea is to generate k disjoint independent sets with size k^ε, for some integer k and constant $\varepsilon > 0$, and all edges between vertices in different independent sets; select two of these independent sets uniformly at random and remove $pk^{2\varepsilon}$ edges between them, for some constant $p \in (0, 1)$, and repeat this process $rk \ln k - 1$ times, where $r > 0$ is another constant. When this is done, any clique in the resulting graph has size at most k. The key step is then to choose k vertices, one in each independent set, and add back any missing edges to form a clique of size exactly k. The expected hardness of these instances comes from choosing specific values of ε, p and r following principles of hardness of phase transitions, i.e., the problem is expected to be easy for some choices of these parameters, but they can be tuned to make it challenging. This method makes use of a reduction from a satisfiability problem to a clique one and hardness results due to Xu and Li [58].

In the next chapter we propose a new family of hard instances, based on a particular reduction from Minimum Vertex Coloring to Maximum Clique in the literature.

5 USING VERTEX COLORINGS TO BUILD CLIQUES

Chapter 3 stablished the fact that $\mathcal{G}(n, p)$ random graphs for constant p, cannot be too hard for MAXIMUM CLIQUE as they simply do not have enough cliques to be enumerated. In Chapter 4, we introduced a natural way of obtaining harder instances, namely to find graphs that at least have many more cliques. In this chapter, we describe a reduction from MINIMUM VERTEX COLORING to MAXIMUM CLIQUE in the literature and how to adapt it into an algorithm to build graphs with many cliques.

5.1 The Representatives Model

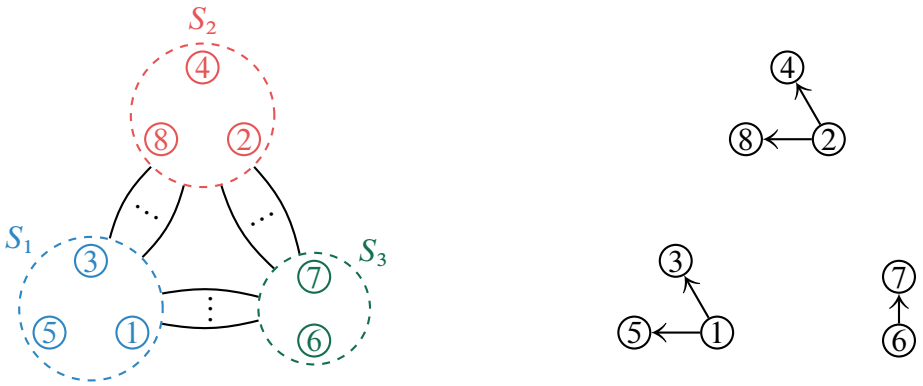
Campêlo, Campos and Corrêa [59] introduce a linear integer programming formulation for MINIMUM VERTEX COLORING. This model, called the "Asymmetric Representatives Model" establishes a connection between colorings and independent sets (and, consequently, between cliques as well).

Given a graph G and a linear order $\prec$ over its vertices, a k-coloring can be expressed by *representatives*, one for each color class S_i, $i \in \{1, 2, \ldots, k\}$. The set of representatives is a transversal to $\{S_1, S_2, \ldots, S_k\}$, i.e., there is exactly one $v_i \in S_i$ that is a representative for each $i \in \{1, \ldots, k\}$, which is the minimum vertex with respect to $\prec$ in the class, i.e., $v_i \prec u, \forall u \in S_i \setminus \{v_i\}$ and every vertex in the graph is either a representative (and represents itself) or is represented by exactly one other vertex. A coloring of G defines uniquely the set of its representatives and who they represent, and the converse is also true (up to color relabeling). Figure 4 shows the relation between a coloring and its representatives.

Figure 4 – Modeling a coloring through its representatives. The smallest vertex of each color class is a representative and represents vertices in the same class.

(a) Coloring of a graph G. Each dashed circle denotes a color class.

(b) Representatives of the coloring. An arc from a vertex u to a vertex v indicates that the former represents the latter.



Source: prepared by the author.

This model was explored by Cornaz and Jost [60], who describe a construction that takes a graph G and an acyclic orientation D of its complement (i.e., an orientation of $\overline{G}$ that has no oriented cycle) as input and outputs a specific construction, which they call $\tilde{G}$. The transformation is presented in Algorithm 5.1.

Algorithm 5.1. CornazJostReduction(G, D)

1 Let $L(\overline{G})$ be the line graph of $\overline{G}$
2 $\tilde{G} \leftarrow L(\overline{G})$
3 **foreach** $xy \in E(\tilde{G})$ **do**
4 | Let $x = uv$ and $y = uz$, as x and y are adjacent edges of $\overline{G}$
5 | **if** $(u, v), (u, z) \in E(D)$ **and** $vz \in E(\overline{G})$ **then**
6 | | $E(\tilde{G}) \leftarrow E(\tilde{G}) \setminus xy$
7 **return** $\tilde{G}$

The authors then show that there is a bijection between colorings of G and independent sets in $\tilde{G}$, suggesting that graphs with many colorings could be transformed into graphs with many independent sets (their complement having many cliques).

Theorem 5.1 (Cornaz and Jost [60])**.** *For any graph G and any acyclic orientation of its complementary graph, there is a one-to-one correspondence between the set of all colorings of G and the set of all stable sets of $\tilde{G}$. Moreover, for any coloring $V_1, \ldots, V_k$ and its corresponding stable set S in $\tilde{G}$, we have: $|S| + k = |V(G)|$. In particular, $\alpha(\tilde{G}) + \chi(G) = |V(G)|$.*

Given a graph G, if no particular acyclic orientation is known, we can define an arbitrary linear order $\prec$ and orient each edge in $\overline{G}$ from the smallest endpoint to the largest (according to $\prec$), and the orientation will be acyclic, as $\prec$ is linear. The conditional on line 5 can then be rewritten as "**if** $u \prec v, u \prec z$ **and** $vz \in E(\overline{G})$ **then**".

As this work is primarily concerned with cliques, we also define G^* as the complement of $\tilde{G}$ to avoid speaking in the language of independent sets. This adapted construction is presented in Algorithm 5.2, assuming that instead of an acyclic ordering, the algorithm receives as input a linear order over the vertex set of G.

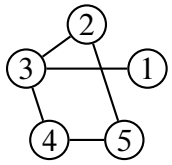
Algorithm 5.2. CliqueCornazJostReduction$(G, \prec)$

1 Let $L(\overline{G})$ be the line graph of $\overline{G}$
2 $\tilde{G} \leftarrow L(\overline{G})$
3 **foreach** $xy \in E(\tilde{G})$ **do**
4 | Let $x = uv$ and $y = uz$, as x and y are adjacent edges of $\overline{G}$
5 | **if** $u \prec v$, $u \prec z$ **and** $vz \in E(\overline{G})$ **then**
6 | | $E(\tilde{G}) \leftarrow E(\tilde{G}) \setminus xy$
7 Let G^* be the complement of $\tilde{G}$
8 **return** G^*

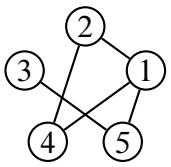
An example of this process is given in Figure 5. Even though the final steps of defining and returning G^* are not described by Cornaz and Jost [60], we hereon call this process the *Cornaz–Jost reduction*.

Figure 5 – Using the Cornaz–Jost reduction to obtain the graph G^* given a graph G and a linear order $\prec$.

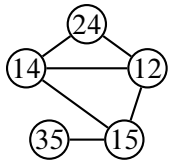
(a) A graph G and a linear order defined over its vertices.



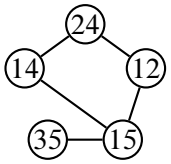
(b) The complementary graph $\overline{G}$.



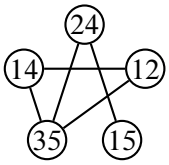
(c) The line graph $L(\overline{G})$ of the complementary graph $\overline{G}$.



(d) The graph $\tilde{G}$, in which the edge between vertices 12 and 14 was removed due to 2 and 4 being neighbors in $\overline{G}$.



(e) The final graph G^*, the complement of $\tilde{G}$.



Source: prepared by the author.

Theorem 5.2. *If G^* is the output of Algorithm 5.2 with input G, then the number of cliques in G^* is equal to the number of colorings of G.*

Proof. There is a natural bijection between the set of cliques in G^* and the set of independent sets in $\tilde{G}$. As each clique in G^* corresponds to an independent set in $\tilde{G}$ and each of these corresponds to a coloring of G by Theorem 5.1, the number of cliques in G^* is precisely the same as the number of colorings of G. □

The structure of G^* depends on the input graph G, so it is natural to focus on a specific family of inputs. Moreover, in order to determine the number of cliques in G^*, we need to know the number of colorings in G. We focus on the case where G is a random graph.

5.2 Counting colorings in random graphs

Following the observations made by Johnson and Trick [31] about the disparity of the results concerning MAXIMUM CLIQUE and MINIMUM VERTEX COLORING during the DIMACS Second Implementation Challenge, Campos, Carmo and N. [61][1] study the average number of colorings on random graphs, in a similar fashion to what was done to cliques.

Theorem 5.3 (Campos, Carmo and N. [61])**.** *For any $\alpha \in (0, 1/3)$, if $G \sim \mathcal{G}(n, p)$ for $p \leq 1 - n^{-\alpha}$, then the expected number of colorings of G is $n^{\Theta(n)}$.*[2]

Note that Theorem 5.3 is stronger the closer α is to $1/3$, as p can be chosen from a wider interval. This result enables counting the expected number of cliques in the Cornaz–Jost reduction with such an input. It also suggests that it should be harder to enumerate colorings than cliques, as random graphs are expected to have much more of the latter than of the former (In the $\mathcal{G}(n, p)$ model with constant p, for example, for a sufficiently large n we have $p \leq 1 - n^{-\alpha}$ for any constant p and any $\alpha \in (0, 1/3)$).

Theorem 5.3 (and the rest of this work) counts colorings as partitions of the vertex set in which each part is an independent set, i.e., relabeling the color classes in a given partition yields the same coloring. In particular, a complete graph has only one coloring.

We now improve Theorem 5.3 by relaxing the condition on α and giving a concentration inequality. To do so, we need the following result.

Theorem 5.4 (Johansson, Kahn and Vu [62])**.** *Given a strictly balanced graph H with k vertices and m edges, if $G \sim \mathcal{G}(n, p)$ for $p = \omega(n^{-(k-1)/m}(\ln n)^{1/m})$ and $n \equiv 0 \pmod{k}$, then the number of H-factors in G is at least*

$$(n^{k-1} p^m)^{n/k} \exp(-\mathcal{O}(n))$$

with probability at least $1 - 1/n^{\omega(1)}$.

Johansson, Kahn and Vu [62] also argue that if we set $m = p\binom{n}{2}$ instead and choose $G \sim \mathcal{G}(n, m)$, the conclusion of Theorem 5.4 is still valid. We are now ready to prove the first main result of this work.

Theorem 5.5. *Let $\alpha \in [0, 1)$ and $c \in (0, 1]$. If $G \sim \mathcal{G}(n, p)$ for $p \leq 1 - cn^{-\alpha}$, then the number of colorings of G is $n^{\Theta(n)}$ with probability $1 - 1/n^{\omega(1)}$.*

Proof. Let C be the random variable that counts the number of colorings of G. We search for two positive constants ε and δ such that $n^{\varepsilon n} \leq C \leq n^{\delta n}$ with probability at least $1 - 1/n^{-\omega(1)}$.

[1] This extended abstract published in 2022 contains a preliminary discussion of some of the results presented in this work.

[2] This theorem is stated slightly different from the original source. Originally, α could be 0, but this lead to a divison by 0 in the proof; moreover, G belonged to the $\mathcal{G}(n, m)$ model for $m \leq \binom{n}{2}(1 - n^{-\alpha})$, but the first step of the proof changed the model back to $\mathcal{G}(n, p)$ (this was mainly due to page constraints).

For the upper bound, as any coloring can use at most n colors, there are at most n^n colorings of G with probability 1.

Now, for the lower bound, we count the number C_2 of colorings where each color class has size exactly 2, as $C \geq C_2$. We assume that n is even, otherwise let G' be an induced subgraph of G with $n-1$ vertices and note that any coloring of G' can be extended to a coloring of G, so the number of colorings of G is at least the number of colorings of G'. Moreover, $G' \sim \mathcal{G}(n-1, p)$ and

$$\left(\frac{n-1}{n}\right)^{-\alpha} = \left(\frac{n}{n-1}\right)^{\alpha} < \frac{n}{n-1} \leq 2,$$

so we can take $c' = c/2$ to get

$$1 - cn^{-\alpha} = 1 - 2c'n^{-\alpha} \leq 1 - c'\left(\frac{n-1}{n}\right)^{-\alpha} n^{-\alpha} = 1 - c'(n-1)^{-\alpha},$$

hence $p \leq 1 - cn^{-\alpha}$ implies $p \leq 1 - c'(n-1)^{-\alpha}$, and we can work with G' instead.

If $H = K_2$, then C_2 is the number of H-factors in $\overline{G}$. Letting $q = 1 - p$, we have $\overline{G} \sim \mathcal{G}(n, q)$ and $q \geq cn^{-\alpha}$, but, in order to apply Theorem 5.4, we need $q = \omega(\ln n/n)$ and, indeed, $n^{-\alpha} = \omega(\ln n/n)$, because $\ln n = o(n^{\beta})$ for any $\beta > 0$, thus, we can take $\beta = 1 - \alpha$. This means that with probability at least $1 - 1/n^{\omega(1)}$

$$\begin{aligned} C_2 &\geq (nq)^{n/2} \exp(-\mathcal{O}(n)) \\ &\geq n^{n/2}(cn^{-\alpha})^{n/2} \exp(-\mathcal{O}(n)) \\ &= n^{n(1-\alpha)/2 - \mathcal{O}(n/\lg n)} \\ &\geq n^{\varepsilon n}, \end{aligned}$$

for some $\varepsilon > 0$. □

In Theorem 5.5, the parameter α is now only required to be less than 1 instead of $1/3$ and the result is now valid w.h.p. (to say with probability $1 - 1/n^{\omega(1)}$ is in fact stronger than to say w.h.p.). Hence, this result is much stronger than Theorem 5.3. To see this, consider the following corollary regarding the expected value, which is still stronger than the old result.

Corollary 5.6. *Let $c \in (0, 1]$ and $\alpha \in [0, 1)$. If $G \sim \mathcal{G}(n, p)$ for $p \leq 1 - cn^{-\alpha}$, then the expected number of colorings of G is $n^{\Theta(n)}$.*

Proof. Again, as $C \leq n^n$, it follows that $\mathbb{E}[C] \leq n^n$. Now, by Markov's inequality, $\mathbb{P}[C \geq t] \leq \mathbb{E}[C]/t$ for any $t > 0$ and, by Theorem 5.5, there exists $\varepsilon > 0$ such that $C \geq n^{\varepsilon n}$ with probability at least $1 - 1/n^{\omega(1)}$. Setting $t = n^{\varepsilon n}$ we have

$$1 - \frac{1}{n^{\omega(1)}} \leq \mathbb{P}[C \geq n^{\varepsilon n}] \leq \frac{\mathbb{E}[C]}{n^{\varepsilon n}},$$

hence $\mathbb{E}[C] \geq n^{\varepsilon n}(1 - 1/n^{\omega(1)}) \geq n^{\varepsilon' n}$ for some $\varepsilon' > 0$. □

Theorem 5.5 thus strengthens the intuition that enumerating colorings should be harder than enumerating cliques, as there are $n^{\Theta(\lg n)}$ cliques w.h.p. versus $n^{\Theta(n)}$ colorings w.h.p. on graphs in the $\mathcal{G}(n, p)$ model for constant p (Just take $\alpha = 0$). Nevertheless, it is possible to decide in polynomial time if a graph has a clique of size at least k for any fixed k (just examine all $\binom{n}{k}$ subsets of size k searching for such a clique in time $\mathcal{O}(n^k)$), while deciding if a graph has a proper coloring using at most k colors is still a NP-complete problem for any $k \geq 3$ [13]. So in a parameterized complexity point of view, MINIMUM VERTEX COLORING *is* harder than MAXIMUM CLIQUE, but our explanation differs in its essence, appealing to enumeration complexity, which we find practical as almost all state-of-the-art algorithms are enumerative (and already were when the second DIMACS challenge took place).

As a final remark, we note that Theorem 5.5 can be adapted to work in the $\mathcal{G}(n, m)$ model, which is useful to fix the number of edges in the input graph, as this affects the number of vertices in the transformed graph.

Corollary 5.7. *Let $\alpha \in [0, 1)$ and $c \in (0, 1]$. If $G \sim \mathcal{G}(n, m)$ for $m \leq \binom{n}{2}(1 - cn^{-\alpha})$, then the number of colorings of G is $n^{\Theta(n)}$ with probability $1 - 1/n^{\omega(1)}$.*

Proof. The n^n upper bound on the number of colorings with probability 1 is still valid. As Johansson, Kahn and Vu [62] state that Theorem 5.4 also works for $G \sim \mathcal{G}(n, m)$ if $m = p\binom{n}{2}$, where $p = \omega(n^{-(k-1)/m}(\ln n)^{1/m})$, the lower bound given in the proof of Theorem 5.5 can be used analogously. □

We are now ready to move on to the Cornaz–Jost reduction analysis.

5.3 Instances with more cliques than average

Campos, Carmo and N. [61] use Theorem 5.3 to analyze the Cornaz–Jost reduction when the inputs to MINIMUM VERTEX COLORING are random graphs.

Theorem 5.8 (Campos, Carmo and N. [61])**.** *For any $n \in \mathbb{N}$ and $\varepsilon \in (0, 1/10]$, the Cornaz–Jost reduction can build graphs with n vertices in which the expected number of cliques is $n^{\Theta(n^{3/5-\varepsilon})}$.*

The algorithm behind this result is as simple as choosing a value of α as a function of the given ε, sampling a random $\mathcal{G}(N, m)$ graph for a particular choice of N and m and applying the Cornaz–Jost reduction on it using a random linear ordering on $V(G)$. Theorem 5.8 can be strengthened using Theorem 5.5 by repeating essentially the same idea but choosing a "better" value of α. Algorithm 5.3 illustrates this method and Theorem 5.9 provides an analysis.

Algorithm 5.3. CORNAZJOSTINSTANCES(n, ε)

1 Let $c \in (0, 1]$ and $N \in \mathbb{N}$ be such that $1 + (2n)^{1-\varepsilon} \leq N \leq (2n/c)^{1-\varepsilon}$
2 $m \leftarrow \binom{N}{2} - n$
3 Sample $G \sim \mathcal{G}(N, m)$ and a linear order $\prec$ over $V(G)$ uniformly at random
4 **return** CLIQUECORNAZJOSTREDUCTION$(G, \prec)$

Theorem 5.9. *For any $n \in \mathbb{N}$ and $\varepsilon \in (0, 1/2]$, the output of Algorithm 5.3 with inputs n and ε has n vertices and $n^{\Theta(n^{1-\varepsilon})}$ cliques with probability at least $1 - 1/n^{\omega(1)}$.*

Proof. Let G^* be the output of Algorithm 5.3 and $\alpha = (1-2\varepsilon)/(1-\varepsilon)$. As $0 < \varepsilon \leq 1/2$,

$$\frac{1}{2-\alpha} = 1 - \varepsilon \quad \text{and} \quad 0 \leq \alpha < 1. \tag{5.1}$$

For any $\varepsilon \in (0, 1/2]$ and $n \in \mathbb{N}$, there exists a small enough constant $c_\varepsilon > 0$ depending only on ε such that

$$[1 + (2n)^{1-\varepsilon}, (2n/c)^{1-\varepsilon}] \cap \mathbb{N} \neq \emptyset, \qquad \forall c \in (0, c_\varepsilon),$$

which means there is always some choice of c and N in line 1. Note that $N \geq 1 + (2n)^{1-\varepsilon}$ implies $(N-1)^{2-\alpha}/2 \geq n$ by (5.1) and this means m is well defined, as

$$m = \binom{N}{2} - n \geq \frac{(N-1)^{2-\alpha}}{2} - n \geq 0.$$

Moreover,

$$m = \binom{N}{2} - n \leq (1 - cN^{-\alpha})\binom{N}{2},$$

so $G \sim \mathcal{G}(N, m)$ has $N^{\Theta(N)}$ colorings with probability at least $1 - 1/N^{\omega(1)}$ by Corollary 5.7. Furthermore, by the choice of N, we have

$$N = \Theta(n^{1-\varepsilon}), \tag{5.2}$$

hence, as the number of cliques in G^* is equal to the number of colorings of G by Theorem 5.2 and $N^{\Theta(N)} = n^{\Theta(n^{1-\varepsilon})}$ by (5.2), G^* has $N^{\Theta(N)} = n^{\Theta(n^{1-\varepsilon})}$ cliques with probability at least $1 - 1/N^{\omega(1)}$, which is $1 - 1/n^{\omega(1)}$. □

In Theorem 5.9, the number of cliques is now $n^{\Theta(n^{1-\varepsilon})}$ instead of $n^{\Theta(n^{3/5-\varepsilon})}$ and this is valid now w.h.p. The least number of cliques remains $n^{\Theta(\sqrt{n})}$, this was achieved before when $\varepsilon = 1/10$ and is now when $\varepsilon = 1/2$. Hence, this result is much stronger than Theorem 5.8. To see this, consider the following corollary regarding the expected number of cliques, which is still stronger than the old result.

Corollary 5.10. *For any $n \in \mathbb{N}$ and $\varepsilon \in (0, 1/2]$, the output of Algorithm 5.3 with inputs n and ε has n vertices and its expected number of cliques is $n^{\Theta(n^{1-\varepsilon})}$.*

Proof. Using a similar argument to that on Theorem 5.9's proof, the expected number of cliques in the output of Algorithm 5.3 is $N^{\Theta(N)}$ by Corollary 5.6. As $N^{\Theta(N)} = n^{\Theta(n^{1-\varepsilon})}$ by (5.2), the result follows. □

These instances *at least* have asymptotically many more cliques than graphs in the $\mathcal{G}(n, p)$ model for constant p, putting them as *candidates* for being harder to solve. They are, however, much denser.

Proposition 5.11. *For any $n \in \mathbb{N}$ and $\varepsilon \in (0, 1/2]$, the output of Algorithm 5.3 with inputs n and ε has expected density $1 - \Theta(n^{1-\varepsilon})$.*

Proof. Let G^* be the output of Algorithm 5.3. First, we count the expected number of edges in $\tilde{G}$, the complement of G^*. If in line 3 of Algorithm 5.3 the sampled graph is $G \sim \mathcal{G}(N, m)$ for some choice of N and m made in line 1, then vertices of $\tilde{G}$ are non-edges of G and edges of $\tilde{G}$ are forbidden pairs of non-edges in G, i.e., pairs that do not define a valid set of representatives and who they represent.

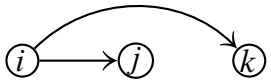
Defining again D as the acyclic orientation of $\overline{G}$ induced by $\prec$, there are four minimal configurations of vertices $i \prec j \prec k$ of G that create edges in $\tilde{G}$:

1. If $ij, ik \notin E(G)$ and $jk \in E(G)$, i.e., $(i, j), (i, k) \in E(D)$;
2. If $ik, jk \notin E(G)$ and $ij \in E(G)$, i.e., $(i, k), (j, k) \in E(D)$;
3. If $ij, jk \notin E(G)$ and $ik \in E(G)$, i.e., $(i, j), (j, k) \in E(D)$;
4. If $ij, jk, ik \notin E(G)$, i.e., $(i, j), (j, k), (i, k) \in E(D)$.

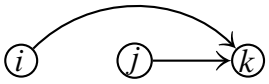
Figure 6 illustrates all cases.

Figure 6 – Vertex configurations in D that induce edges in $\tilde{G}$.

(a) Case 1.

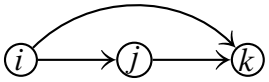


(b) Case 2.



(c) Case 3.

(d) Case 4.



Source: prepared by the author.

In case 1, if i represents k, then j cannot be represented by i, so (i, j) and (i, k) cannot be taken simultaneously in $\tilde{G}$, meaning that $(i, j)(j, k) \in E(\tilde{G})$. In case 2, if i represents k, then j cannot also represent k, so $(i, k)(j, k) \in E(\tilde{G})$ analogously. In case 3, if represents j, then k cannot be represented by j, so now $(i, j)(j, k) \in E(\tilde{G})$. Finally, in case 4, we have a superposition of all three other cases, so again $(i, k)(j, k)$ and $(i, j)(j, k)$ are edges in D, but now $(i, j)(i, k)$ is not because i can represent both j and k as they are not neighbors in G now.

The probability that case 1 occurs is the same as that of cases 2 and 3, being

$$\binom{\binom{N}{2} - 3}{m - 1} \Big/ \binom{\binom{N}{2}}{m},$$

as (in case 1) jk must be in G and both ij and ik are forbidden, so another $m-1$ edges must be sampled among the $\binom{N}{2}-3$ remaining choices. The probability that case 4 occurs is, analogously,

$$\binom{\binom{N}{2}-3}{m}\Big/\binom{\binom{N}{2}}{m}.$$

Cases 1, 2 and 3 all induce one edge in $\tilde{G}$, while case 4 induces two and all four cases are defined by triples of vertices in G, thus,

$$\begin{aligned}\mathbb{E}[e(\tilde{G})] &= \binom{N}{3}\left(3\binom{\binom{N}{2}-3}{m-1}+2\binom{\binom{N}{2}-3}{m}\right)\Big/\binom{\binom{N}{2}}{m}\\ &= \binom{N}{3}\left(\binom{\binom{N}{2}-3}{m-1}+2\binom{\binom{N}{2}-2}{m}\right)\Big/\binom{\binom{N}{2}}{m},\end{aligned}$$

by Pascal's rule. Now, symplifying, we get

$$\begin{aligned}\mathbb{E}[e(\tilde{G})] &= \binom{N}{3}\left(\frac{\left(\binom{N}{2}-3\right)!}{(m-1)!\left(\binom{N}{2}-m-2\right)!}+\frac{2\left(\binom{N}{2}\right)}{m!\left(\binom{N}{2}-m-2\right)!}\right)\Big/\binom{\binom{N}{2}}{m}\\ &= \binom{N}{3}\left(\frac{m\left(\binom{N}{2}-m\right)\left(\binom{N}{2}-m-1\right)}{\binom{N}{2}\left(\binom{N}{2}-1\right)\left(\binom{N}{2}-2\right)}+\frac{2\left(\binom{N}{2}-m\right)\left(\binom{N}{2}-m-1\right)}{\binom{N}{2}\left(\binom{N}{2}-1\right)}\right)\end{aligned}$$

and letting $p = m/\binom{N}{2}$, it follows that

$$\begin{aligned}\mathbb{E}[e(\tilde{G})] &= \binom{N}{3}\left(\frac{\binom{N}{2}p(1-p)\left(\binom{N}{2}(1-p)-1\right)}{\left(\binom{N}{2}-1\right)\left(\binom{N}{2}-2\right)}+\frac{2(1-p)\left(\binom{N}{2}(1-p)-1\right)}{\binom{N}{2}-1}\right)\\ &= \binom{N}{3}\frac{(1-p)\left(\binom{N}{2}(1-p)-1\right)}{\binom{N}{2}-1}\left(\frac{\binom{N}{2}p}{\binom{N}{2}-2}+2\right).\end{aligned}$$

Now, we can define $\tilde{G}$'s expected density to be $\mathbb{E}[d(\tilde{G})] = \mathbb{E}[e(\tilde{G})]/\binom{v(\tilde{G})}{2}$ and, since $v(\tilde{G}) = n = \binom{N}{2}-m = (1-p)\binom{N}{2}$, we have

$$\begin{aligned}\mathbb{E}[d(\tilde{G})] &= \binom{N}{3}\frac{(1-p)\left(\binom{N}{2}(1-p)-1\right)}{\binom{N}{2}-1}\left(\frac{\binom{N}{2}p}{\binom{N}{2}-2}+2\right)\cdot\\ &\qquad\qquad\frac{2}{(1-p)\binom{N}{2}\left((1-p)\binom{N}{2}-1\right)}\\ &= \binom{N}{3}\frac{2}{\binom{N}{2}\left(\binom{N}{2}-1\right)}\left(\frac{\binom{N}{2}p}{\binom{N}{2}-2}+2\right)\\ &= \Theta(N^3)\cdot\Theta(N^{-4})\cdot\Theta(p)\\ &= \Theta(p/N).\end{aligned}$$

By (5.2),

$$p = \frac{m}{\binom{N}{2}} = \frac{\Theta(N^2)-\Theta(N^{2-\alpha})}{\Theta(N^2)} = \Theta(1).$$

Therefore, as $\mathbb{E}[d(G^*)] = 1 - \mathbb{E}[d(\tilde{G})]$, we get

$$\mathbb{E}[d(G^*)] = 1 - \Theta(1/N) = 1 - \Theta(1/n^{1-\varepsilon}). \qquad \square$$

The actual hardness of these instances depends on the specific standard algorithm being used. We leave the theoretical analysis of the behavior of these graphs under different algorithm parameters, such as upper bound rules, for future work, but we test them in computational experiments in Chapter 7.

5.4 A final remark on random graphs

We have seen that $\mathcal{G}(n, p)$ random graphs for constant p have $n^{\Theta(\lg n)}$ cliques w.h.p. and their density is concentrated around p, as opposed to our instances given by the Cornaz–Jost reduction with parameters n and ε, which have expected density $1 - \Theta(1/n^{1-\varepsilon})$. A natural follow up question is to determine the number of cliques in a $\mathcal{G}(n, p)$ random graph when p is allowed to increase with n.

If $p = 1$, there are 2^n cliques, which is the most number of cliques for any graph. Bläsius, Katzmann and Stegehuis [63] show that if $p = 1 - \Theta(1/n)$ and n is large enough, then a graph in the $\mathcal{G}(n, p)$ model has $n^{\Theta(n/\lg n)} = 2^{\Theta(n)}$ cliques w.h.p. To the best of our knowledge, there is no analogous result for $p = 1 - \Theta(1/n^{1-\varepsilon})$, which would allow us to compare our instances to $\mathcal{G}(n, p)$ graphs with a matching density. We provide an answer regarding the expected number of clique of such graphs.

Theorem 5.12. *For any $\varepsilon \in (0, 1]$, if $G \sim \mathcal{G}(n, p)$ for $p = 1 - \Theta(1/n^{1-\varepsilon})$, then the expected number of cliques in G is $n^{\Theta(n^{1-\varepsilon} \lg n)}$.*

Proof. Take two positive constants c and c' such that

$$1 - \frac{c'}{n^{1-\varepsilon}} \leq p \leq 1 - \frac{c}{n^{1-\varepsilon}}.$$

Let X_k be the random variables that count the number of cliques of size k in G, for $k = 0, \ldots, n$ and X be the random variable that counts the total number of cliques in G. There are $\binom{n}{k}$ sets of k vertices and the probability that one such set is a clique is $p^{\binom{k}{2}}$, hence, by linearity of expectation, it follows that

$$\binom{n}{k}\Big(1 - \frac{c'}{n^{1-\varepsilon}}\Big)^{\binom{k}{2}} \leq \mathbb{E}[X_k] \leq \binom{n}{k}\Big(1 - \frac{c}{n^{1-\varepsilon}}\Big)^{\binom{k}{2}},$$

and

$$\sum_{k=0}^{n}\binom{n}{k}\Big(1 - \frac{c'}{n^{1-\varepsilon}}\Big)^{\binom{k}{2}} \leq \mathbb{E}[X] \leq \sum_{k=0}^{n}\binom{n}{k}\Big(1 - \frac{c}{n^{1-\varepsilon}}\Big)^{\binom{k}{2}}. \tag{5.3}$$

Now, by (2.5) and (2.6), we have

$$\binom{n}{k}\Big(1 - \frac{c}{n^{1-\varepsilon}}\Big)^{\binom{k}{2}} \leq \exp\Big(\frac{ck}{2n^{1-\varepsilon}} + k - \frac{ck^2}{2n^{1-\varepsilon}} + k\ln n - k\ln k\Big). \tag{5.4}$$

If we define $f : [0, n] \to \mathbb{R}$ as

$$f(x) = \frac{cx}{2n^{1-\varepsilon}} + x - \frac{cx^2}{2n^{1-\varepsilon}} + x \ln n - x \ln x,$$

we see that f is twice differentiable in $(0, n)$ and

$$f'(x) = \frac{c}{2n^{1-\varepsilon}} - \frac{cx}{n^{1-\varepsilon}} + \ln n - \ln x.$$

Note that $f'(x) = 0$ when

$$\frac{c}{2n^{1-\varepsilon}} - \frac{cx}{n^{1-\varepsilon}} + \ln n - \ln x = 0,$$

that is,

$$\frac{c}{2n^{1-\varepsilon}} + \ln n = \frac{cx}{n^{1-\varepsilon}} + \ln x. \tag{5.5}$$

Let $y = cx/n^{1-\varepsilon}$, so we can rewrite (5.5) as

$$\frac{c}{2n^{1-\varepsilon}} + \ln n = y + \ln y + (1-\varepsilon)\ln n - \ln c$$

and apply the exponential function in both sides, getting

$$\exp\Big(\frac{c}{2n^{1-\varepsilon}} + \varepsilon \ln n + \ln c\Big) = y \exp(y). \tag{5.6}$$

Equation (5.6) satisfies the functional equation of a Lambert function, as in (2.3). Its solution's asymptotics is given by (2.4), so

$$y = \frac{c}{2n^{1-\varepsilon}} + \varepsilon \ln n + \ln c - \Theta(\ln \ln n),$$

which means

$$x = \Big(\frac{\varepsilon}{c} \pm o(1)\Big) n^{1-\varepsilon} \ln n.$$

Furthermore, $f''(x) = -c/n^{1-\varepsilon} - 1/x$, which is always negative, so f is concave and its maximum happens at $(\varepsilon/c \pm o(1))n^{1-\varepsilon} \ln n$. Plugging this value for k and (5.4) into (5.3), we end up with

$$\begin{aligned}
\mathbb{E}[X] &\le n \exp\bigg(\Big(-\frac{\varepsilon^2}{2c} + \frac{\varepsilon}{c} - \frac{\varepsilon}{c}(1-\varepsilon) \pm o(1)\Big) n^{1-\varepsilon} \ln^2 n\bigg) \\
&= n \exp\bigg(\Big(\frac{\varepsilon^2}{2c} \pm o(1)\Big) n^{1-\varepsilon} \ln^2 n\bigg) \\
&= n^{\Theta(n^{1-\varepsilon} \lg n)}.
\end{aligned}$$

For the lower bound, we start by noting that $\mathbb{E}[X] \ge \mathbb{E}[X_k]$ for any k and, by (2.6),

$$1 - x = \frac{1}{1/(1-x)} = \frac{1}{1 + \frac{x}{1-x}} \ge \frac{1}{\exp\left(\frac{x}{1-x}\right)} = \exp\Big(-\frac{x}{1-x}\Big). \tag{5.7}$$

Plugging (5.7) and (2.7) into (5.3), it follows that

$$\begin{aligned}\mathbb{E}[X] &\geq \mathbb{E}[X_k] \\ &= \binom{n}{k}\left(1 - \frac{c'}{n^{1-\varepsilon}}\right)^{\binom{k}{2}} \\ &\geq \left(\frac{n}{k}\right)^k \exp\left(-\binom{k}{2}\frac{c'}{n^{1-\varepsilon}}\right) \\ &= \exp\left(-\binom{k}{2}\frac{c'}{n^{1-\varepsilon}} + k\ln n - k\ln k\right).\end{aligned}$$

For any $\varepsilon \in (0,1)$, there are only finitely many integer values of n for which $n^{1-\varepsilon}\ln n > n$. Hence, choosing a constant δ (depending only on ε) small enought such that

$$\delta n^{1-\varepsilon}\ln n \leq n, \qquad \forall n \in \mathbb{N}$$

we see that setting $k = \lfloor \delta n^{1-\varepsilon}\ln n \rfloor$ implies $k \leq n$. Thus,

$$\mathbb{E}[X] \geq \exp\left(\left(-\frac{c'\delta^2}{2} + \delta - \delta(1-\varepsilon) \pm o(1)\right)n^{1-\varepsilon}\ln^2 n\right),$$

but

$$-\frac{c'\delta^2}{2} + \delta - \delta(1-\varepsilon) = \delta\left(\varepsilon - \frac{c'\delta}{2}\right)$$

and if $\delta < 2\varepsilon/c'$ (again we can take a small δ depending only on ε and c', which are constants), this factor is positive. Therefore,

$$\begin{aligned}\mathbb{E}[X] &\geq \exp(\Theta(n^{1-\varepsilon}\lg^2 n)) \\ &= n^{\Theta(n^{1-\varepsilon}\lg n)}.\end{aligned}$$

□

This result tells us that the expected number of cliques in $\mathcal{G}(n, p)$ random graphs is even higher than that of our instances built from the Cornaz–Jost reduction. In Chapter 7 we compare them further using a standard algorithm with a strong upper bound in order to get an idea of their hardness difference in practice.

As a final remark, we note that if set $\varepsilon = 1$ in Theorem 5.12, we get a constant p and $n^{\Theta(\lg n)}$ cliques, agreeing with Lemma 3.4. If we were to set $\varepsilon = 0$, however, we would get $p = 1 - \Theta(1/n)$ but $n^{\Theta(n\lg n)}$ cliques, which does not agree with the result due to Bläsius, Katzmann and Stegehuis [63] (in particular this would not even be possible as the number of cliques in any graph is at most $n^{n/\lg n}$).

In the next chapter we analyze instances that are hard only for a subclass of standard algorithms, characterized by the usage of a particular (and efficient) upper bound.

6 CHROMATIC UPPER BOUNDS FOR STANDARD ALGORITHMS

The instances' analyses so far do not concern specific upper bound rules, which makes them valid for any standard algorithm. On the other hand, all state-of-the-art algorithms apply some sort of pruning rule to reduce the number of subinstances to be solved. When this is done, a high number of cliques on its own (and even ideas described in Section 4.1 of Chapter 4) may not be enough to ensure hardness, so we need more fine tuned arguments based on the upper bound being used. Hence, we now turn our attention to a more specific type of standard algorithms that have been vastly adopted and studied.

6.1 Introducing a bounding rule

Recall that standard algorithms are enumerative but need not implement any kind of upper bound in order to avoid branching when it is not necessary. Introducing such bounds, however, can lead to very efficient algorithms, provided the bound does not take up much time to be evaluated. A common rule to stop a node from branching is comparing the size of the largest clique already found to the least number of colors needed to color the graph induced by the current node's subinstance, where a graph G induced by the subinstance (Q, R) is $G[Q \cup R]$. We call this strategy a *chromatic upper bound*.

It is a well known fact that if a graph G has a clique Q, then any proper coloring of G uses at least $|Q|$ colors, as each vertex on Q must have its own color, hence,

$$\chi(G) \geq \omega(G). \tag{6.1}$$

Furthermore, if a subinstance (Q, R) is such that $\chi(G[Q \cup R]) \leq k$, then $\omega(G[Q \cup R]) \leq k$ and if a clique of size k has already been found, there is no real reason to keep branching after this node. This idea was first proposed by Babel and Tinhofer [16].

Note that computing the chromatic number of those subgraphs is not trivial in general, so, in order to keep the upper bound feasible time-wise, a possibly non optimal number of colors is computed by some heuristic instead, which still is an upper bound nonetheless. So, for each node (Q, R), the algorithm computes an upper bound, say $\Gamma(G[Q \cup R])$, on $\chi(G[Q \cup R])$ and checks if the largest clique found so far, say Q^*, satisfies $\Gamma(G[Q \cup R]) \leq |Q^*|$, and if so, the algorithm prunes this branch. An algorithm that implements this pruning rule is called a *χ-bounded* algorithm. In particular, a χ-bounded algorithm never discards a node (Q, R) in which $\chi(G[Q \cup R]) > \omega(G)$. We remark that the vast majority of state-of-the-art algorithms for MAXIMUM CLIQUE use some sort of chromatic upper bound, although newer results apply graph reductions (usually to satisfiability problems) before the coloring to obtain *infra*-chromatic upper bounds [21].

We now define a substructure of clique search trees.

Definition 6.1. Given a graph G and a clique search tree T of G, the *χ-pruned* subtree of T, denoted by T_χ, is the (unique) subtree of T such that:

1. The root of T_χ is $(\emptyset, V(G))$;
2. In every leaf (Q, R) of T_χ, we have $\chi(G[Q \cup R]) \leq \omega(G)$.
3. T_χ is minimal in size;

Intuitively, a χ-pruned tree is obtained by pruning several branches of a clique search tree. It can also be viewed as an "optimal" execution of a χ-bounded algorithm, as in each step the algorithm always colors the graph induced by the current subinstance with the least number of colors possible and, at any point, the largest clique found is always a maximum one. The following result describes the relation between χ-bounded algorithms and χ-pruned subtrees.

Proposition 6.2. *For any execution $\mathcal{E}$ of a χ-bounded algorithm on a graph G, there is a clique search tree T of G such that $\mathcal{E}$ is contained in T and the set of subinstances considered by $\mathcal{E}$ contain $V(T_\chi)$.*

Proof. By Proposition 3.3, there is a clique search tree T that contains $\mathcal{E}$, because any χ-bounded algorithm is a standard algorithm. Let $\Gamma(G)$ be the upper bound on the chromatic number evaluated by the χ-bounded algorithm on a graph G. Both T and T_χ share the same node as root, namely $(\emptyset, V(G))$, which is the first node considered in $\mathcal{E}$. If $(Q, R) \in V(T_\chi)$, then the parent (Q_P, R_P) of (Q, R) is not a leaf in T_χ and $\chi(G[Q_P \cup R_P]) > \omega(G)$, but

$$\Gamma(G[Q_P \cup R_P]) \geq \chi(G[Q_P \cup R_P]) > \omega(G),$$

so in $\mathcal{E}$, (Q_P, R_P) cannot be a leaf and both its children are considered too, hence $V(T_\chi)$ is contained in the set of instances considered by $\mathcal{E}$. □

So, essentially, the number of subinstances considered in an execution of a χ-bounded algorithm is at most the size of some clique search tree T and at least the size of the χ-pruned subtree T_χ.

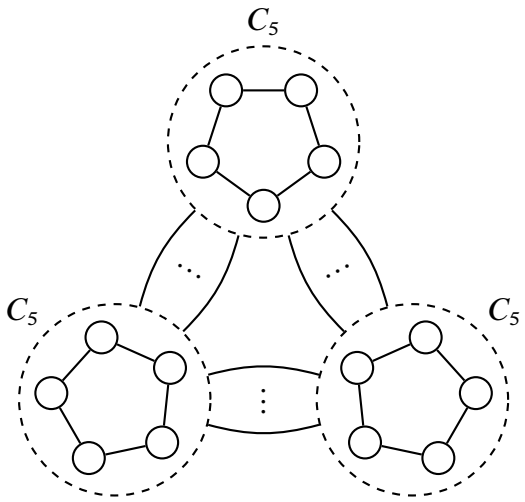
6.2 Exponential running time inducing graphs

We now define the class of Lavnikevich graphs. This notion was introduced by Lavnikevich [26] and Figure 7 provides an example.

Definition 6.3. For any $n \equiv 0 \pmod 5$, the Lavnikevich graph on n vertices, denoted by L_n, is obtained by the graph join between $n/5$ C_5's.

With a different notation, the author proves the following.

Figure 7 – Outline of the L_{15} graph, where each vertex in a C_5 is connected to all other vertices in the other two C_5's.



Source: prepared by the author.

Theorem 6.4 (Lavnikevich [26])**.** *The χ-pruned subtree of any clique search tree of the L_n graph has size $\Omega(2^{n/5})$.*

The number of instances considered in a χ-bounded algorithm is at least the size of a χ-pruned subtree contained in it by Proposition 6.2, hence, even if the algorithm takes $\Theta(1)$ time to process each node (which is too optimistic as the algorithm needs to color de graph induced by the node's intances), it still needs to process an exponential number of nodes. This establishes Lavnikevich graphs as exponential time instances for χ-bounded algorithms.

6.3 A preprocessing heuristic

A Lavnikevich graph is hard to solve via χ-bounded algorithms, but its structure allows the usage of a simple preprocessing step that drastically speeds up the solving process.

Proposition 6.5. *For any graph G, if $G = G_1 + G_2 + \cdots + G_k$, then*

$$\omega(G) = \max_{1 \leq i \leq k} \{\omega(G_i)\}.$$

Proof. As there are no edges between G_i and G_j for any $i \neq j$, any clique in G is contained in a single G_i for some i, so its largest clique is also the largest clique in some G_i. □

Proposition 6.6. *For any graph G, if $G = G_1 \vee G_2 \vee \cdots \vee G_k$, then*

$$\omega(G) = \sum_{i=1}^{k} \omega(G_i).$$

Proof. Given a clique Q_i of G_i, each vertex of Q_i is adjacent to every vertex in any clique Q_j of G_j. So $Q_1 \cup Q_2 \cup \cdots \cup Q_k$, where Q_i is a clique in G_i, is a clique of G and if we chose each

Q_i to be a maximum clique in G_i, we have a maximum clique in G, for if Q' was a larger clique in G, then $|Q' \cap V(G_i)| > \omega(G_i)$ for some $i \in \{1, 2, \ldots, k\}$ by the pigeonhole principle. □

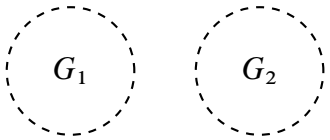
By Propositions 6.5 and 6.6, we can preprocess a graph by recursively splitting it into its components or into subgraphs induced by its complement's components. Algorithm 6.1 sums up this process, taking as input a graph G and a standard algorithm $\mathcal{A}$ and applying one of the two steps if possible. Figure 8 illustrates the strategy.

Algorithm 6.1. PREPROCESS$(G, \mathcal{A})$

```
1  Let C and C̄ be the sets of the components of G and Ḡ, respectively
2  if |C| > 1 then
3  |  Q* ← ∅
4  |  foreach C ∈ C do
5  |  |  Q ← PREPROCESS(C, A)
6  |  |  if |Q| > |Q*| then
7  |  |  |  Q* ← Q
8  |  return Q*
9  if |C̄| > 1 then
10 |  Q ← ∅
11 |  foreach C̄ ∈ C̄ do
12 |  |  Q ← Q ∪ PREPROCESS(C, A)
13 |  return Q
14 return A(G)
```

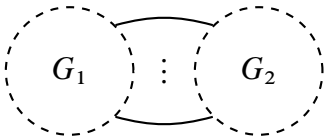
Figure 8 – The decomposition of a graph G using Algorithm 6.1. In each case, the algorithm is called recursively for G_1 and G_2.

(a) If $G = G_1 + G_2$.



$\omega(G_1 + G_2) = \max\{\omega(G_1), \omega(G_2)\}$

(b) If $G = G_1 \vee G_2$.



$\omega(G_1 \vee G_2) = \omega(G_1) + \omega(G_2)$

Source: prepared by the author.

This procedure is a weaker version of a Modular Decomposition (see [64] for a survey), but much easier to implement. For a graph on n vertices and m edges, it can be naïvely implemented in $\mathcal{O}(n^2)$ time (just run a depth-first search in G then another in $\overline{G}$), but Tedder et al. [65] show how to compute a Modular Decomposition in $\mathcal{O}(n + m)$ time, which indicates that our preprocessing could be computed faster.

Now, if the preprocessing is applied using a χ-bounded algorithm and L_n as the inputs, the problem is greatly reduced. This is because $L_n = C_5 \vee C_5 \vee \cdots \vee C_5$ where the join is done $n/5$ times and, thus, the algorithm has to solve MAXIMUM CLIQUE in the C_5 a linear number of times. The preprocess takes $\Theta(n^2)$ time (even in a potential linear implementation,

as $e(L_n) = \Theta(n^2)$) and the original problem is split in $\Theta(n)$ problems of $\Theta(1)$ size, that can be solved in linear time in n. The whole solving process takes quadratic time, which is a great improvement from the original exponential lower bound.

As the final step of this work, we focus our attention in graphs that maintain the exponential solving time requirement for any χ-bounded algorithm and resist the preprocessing described above.

6.4 Worst case instances resistant to the preprocessing

Essentially, we search for graphs on n vertices that when given as input, together with a χ-bounded algorithm $\mathcal{A}$, to Algorithm 6.1 ensure that at least one call to $\mathcal{A}$ has an input graph with $\Theta(n)$ vertices.

The first step to this intent is looking to L_n itself, searching for a way to shield it against the preprocessing. Campos, Carmo and N. [61] describe a way to obtain subgraphs of L_n that still exhibit exponential size χ-pruned subtrees. We present a proof of Lemma 6.7 as the authors omitted it from the original paper due to page constraints.

Lemma 6.7 (Campos, Carmo and N. [61])**.** *For any $n, d \in \mathbb{N}$ with $n \equiv 0 \pmod 5$, if G is a spanning subgraph of L_n where $\alpha(G) \leq 2$ and $\delta(G) \geq n - d - 1$, then the χ-pruned subtree of every clique search tree of G has $\Omega(2^{n/(5d)})$ nodes.*

Proof. Given a node (Q, R) of the χ-pruned subtree T_χ of a clique search tree in G and a pivot $v \in R$, a branching operation may discard one vertex from $G[Q \cup R]$ if it excludes v from the current clique and at most d if v is included as there are at most d vertices non adjacent to it. Therefore, if less than $n/(5d)$ branch operations happened in the path from the root to a node (Q, R), then less than $n/5$ vertices have been discarded in this subinstance and we have

$$\chi(G[Q \cup R]) \geq \frac{n(G[Q \cup R])}{\alpha(G[Q \cup R])} > \frac{n - n/5}{2} = \frac{2n}{5} = \omega(L_n) \geq \omega(G),$$

which means (Q, R) is not a leaf in T_χ. Therefore, the size of T_χ is $\Omega(2^{n/(5d)})$. □

The real matter is how to obtain such subgraphs that endure the preprocessing. If we want a spanning subgraph of L_n, we can only remove some of its edges (not vertices), which means we add edges to its complement. By asking the independent sets of this subgraph to have size at most two, we ask its complement to be triangle-free and by asking its minimum degree to be at least $n - d - 1$, we ask its complement's maximum degree to be at most d. From now on, then, we argue how to build the complement $\overline{G}$ of the subgraph G we want.

We build $\overline{G}$ by adding edges to $\overline{L_n}$ (removing edges from L_n) until it becomes connected, while no triangle is created, its maximum degree does not exceed d and G remains connected (initially $G = L_n$). We want d to be as small as possible so the lower bound on the χ-pruned subtrees' size is as large as possible (although it is always exponentially large in n

for any fixed d), but note that $\overline{L_n}$ is 2-regular, so setting $d = 1$ in Lemma 6.7 is of no use as any possible $\overline{G}$ has maximum degree at least 2. Asking for $d = 2$ does not help either, as the only possible choice for $\overline{G}$ here is $\overline{L_n}$ itself, which is disconnected. Hence, we need $d \geq 3$ and, indeed, we argue $d = 3$ is enough.

Now, if $d = 3$, then any vertex in $\overline{L_n}$ gains at most one edge. Even though it is possible to connect $\overline{L_n}$ adding only $n/5 - 1$ edges (just contract each C_5 into a vertex and build a tree), this would give $\overline{G}$ a tree-like structure that could be exploited in some other preprocessing heuristic. Once one edge is added, we already have $d = 3$ and it would do no harm (asymptotically) to add as many edges as possible, i.e., a maximum matching.

We use the Configuration Model due to Bollobás [66] to sample a random maximum matching to be added in $\overline{L_n}$. This is a tool for modeling random graphs with a prescribed degree sequence. Algorithm 6.2 illustrates the sampling of a graph on n vertices with degree distribution $\mathbf{d} = (d_1, d_2, \ldots, d_n)$, assuming $\sum_{i=1}^{n} d_i$ is even. It associates each vertex v_i with d_i *half-edges*, and then randomly pairs half-edges to form a *configuration*, which essentially describes the structure of the output graph, in which $d(v_i) = d_i$ for each $i \in \{1, 2, \ldots, n\}$.

Algorithm 6.2. CONFIGURATION$(n, \mathbf{d})$

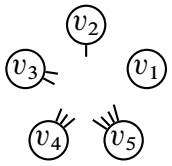
1 Let G be a graph such that $V(G) = \{v_1, v_2, \ldots, v_n\}$ and $E(G) = \{e_1, e_2, \ldots, e_m\}$, where $m = \frac{1}{2}\sum_{i=1}^{n} d_i$
2 Let $S = \{v_1^1, \ldots, v_1^{d_1}, v_2^1, \ldots, v_2^{d_2}, \ldots, v_n^1, \ldots, v_n^{d_n}\}$ be the set of half-edges
3 **for** $i \leftarrow 1, \ldots, m$ **do**
4 Pick x uniformly at random from S
5 $S \leftarrow S \setminus \{x\}$
6 Pick y uniformly at random from S
7 $S \leftarrow S \setminus \{y\}$
8 Map the edge e_i to the unordered pair $v'v''$, where x is a half-edge of v' and y is a half-edge of v''
9 **return** G

Algorithm 6.2 may output a graph that is not simple, as lines 4 and 6 may sample half-edges of the same vertex or half-edges of vertices already adjacent. Many configurations lead to the same output, and the probability of it being some specific graph H depends on the structure of H. If we condition the output to be simple, however, the distribution of the outputs becomes uniform. We refer the reader to the classic book due to Bollobás [37] and a survey due to Wormald [67] for a proof of this and many other results about this model. Figure 9 exemplifies an execution of this process.

We now present two results regarding the structure of the output of Algorithm 6.2 that we use as lemmas. In each of them, we denote the (nondeterministic) output graph of Algorithm 6.2 with inputs n and $\mathbf{d}$ by $G_{n,\mathbf{d}}$.

Figure 9 – An execution of Algorithm 6.2 with $n = 5$ and $\mathbf{d} = (0, 1, 2, 3, 4)$.

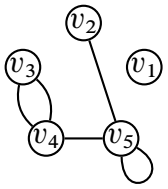
(a) Each vertex v_i is assigned d_i half-edges.



(b) A random pairing of the half-edges is sampled.

$$v_2^1v_5^3,\ v_4^2v_3^1,\ v_3^2v_4^3,$$
$$v_4^1v_5^2,\ v_5^1v_5^4$$

(c) The final configuration.



Source: prepared by the author.

Theorem 6.8 (Bollobás [66]). *If $d_i = \mathcal{O}(1)$ for each $i \in \{1, 2, \ldots, n\}$ and $\sum_{i=1}^{n} d_i = n + \omega(1)$, then the probability that $G_{n,\mathbf{d}}$ is loopless is*

$$(1 \pm o(1)) \exp\left(-\frac{1}{2m} \sum_{k=1}^{n} \binom{d_k}{2}\right).$$

Theorem 6.9 (Wormald [68]). *Fix $\delta, \Delta \in \mathbb{N}$ such that $3 \leq \delta \leq \Delta$. If $\delta \leq d_i \leq \Delta$ for each $i \in \{1, 2, \ldots, n\}$, then the probability that $G_{n,\mathbf{d}}$ is δ-connected is at least $1 - \mathcal{O}(1/n^{\delta-2})$.*

Bollobás [66] actually proves a stronger version of Theorem 6.8 on the distribution of cycles of length k for any $k \geq 1$. Algorithm 6.3 uses the spirit of the Configuration Model to take as input a (simple) graph and add a random maximum matching to it.

Algorithm 6.3. AddMatching(G)

```
Let S be a list containing the vertices of G
while |S| > 1 do
    Pick a vertex v ∈ S uniformly at random
    S ← S \ {v}
    Pick a vertex u ∈ S uniformly at random
    S ← S \ {u}
    E(G) ← E(G) ∪ {e}
    Map e to uv
return G
```

Naturally, we wish to apply Algorithm 6.3 on $\overline{L_n}$. The resulting graph need not be connected, triangle-free or even simple. If it is not triangle-free, then we cannot apply Lemma 6.7; if it is not connected, then the preprocessing step may break it into small parts and if it is not simple, then we cannot build a subgraph of L_n based on it, because adding an edge between vertices in $\overline{L_n}$ means removing an edge between vertices in L_n and if we add an edge between adjacent vertices then we would have to remove an edge between non-adjacent vertices, which is not a valid choice. We adress the probability that these events occur in the next lemma.

Lemma 6.10. *Let G_n be the output of Algorithm 6.3 with input $\overline{L_n}$. The probability that G_n is simple, triangle-free and connected is* $\exp(-2) \pm o(1)$.

Proof. We can view an execution of Algorithm 6.3 on $\overline{L_n}$ as an execution of Algorithm 6.2 on $n/5$ vertices (one for each C_5 in $\overline{L_n}$) with degree sequence being 5 for every vertex when n is even or 5 for $n-1$ vertices and 4 for the last vertex when n is odd; for each vertex, its half-edges are the vertices of its associated C_5 (except some vertex in some C_5 is ignored when n is odd). Whenever two vertices are sampled and an edge is added to $\overline{L_n}$, imagine that the two half-edges corresponding to these vertices are paired.

Let H be the output graph of this simulation of Algorithm 6.2. A loop in H corresponds to an edge being added between vertices of the same C_5 in $\overline{L_n}$. Note that if an edge is added between vertices of the same C_5 in $\overline{L_n}$, then either G_n is not simple or it is not triangle-free, as we can see in Figure 10.

Figure 10 – Problematic configurations when Algorithm 6.2 adds edges with both endpoints in the same C_5.

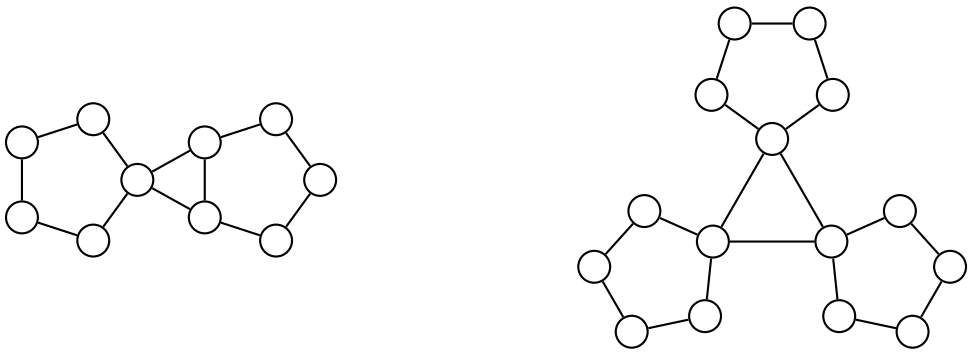
(a) In this configuration, G_n is not triangle-free. (b) In this configuration, G_n is not simple.

Source: prepared by the author.

Moreover, if no edge with both endpoints in the same C_5 is added, then no triangle is created, as vertices in different C_5s can only induce a triangle in G_n if at least one of them receives two new edges (see Figure 11), but we add only a matching.

Figure 11 – No triangle can be induced by vertices of different C_5s.

(a) Two C_5s. (b) Three C_5s.



Source: prepared by the author.

If no edge is added between vertices of the same C_5, then G must be simple as vertices in different C_5s are initally non-adjacent and receive at most one edge each, so no parallel edges exist. Hence, G_n is simple and triangle-free if, and only if, H has no loops.

Now, as G_n is connected if, and only if, H is connected, it follows that

$$\mathbb{P}[G_n \text{ is simple, triangle-free and connected}] = \mathbb{P}[H \text{ is loopless and connected}].$$

For an upper bound, consider the events $\{H \text{ is loopless}\}$ and $\{H \text{ is connected}\}$ and note that their probabilities are both at least the probability we want, as they contain the event we are interested. When $n \to \infty$, the lowest upper bound is $\mathbb{P}[H \text{ is loopless}]$, as

$$\begin{aligned}\mathbb{P}[H \text{ is loopless}] &= (1 \pm o(1)) \exp\left(-\frac{1}{2m}\sum_{k=1}^{n}\binom{d_k}{2}\right)\\ &= (1 \pm o(1)) \exp\left(-\frac{1}{n}\frac{10n}{5}\right)\\ &= \exp(-2) \pm o(1),\end{aligned}$$

for even n and

$$\begin{aligned}\mathbb{P}[H \text{ is loopless}] &= (1 \pm o(1)) \exp\left(-\frac{1}{n-1}\left(10\left(\frac{n}{5}-1\right)+6\right)\right)\\ &= (1 \pm o(1)) \exp\left(-\frac{2n-4}{n-1}\right)\\ &= \exp(-2) \pm o(1),\end{aligned}$$

for odd n by Theorem 6.8. For a lower bound, note that

$$\begin{aligned}\mathbb{P}[H \text{ is loopless}] = \mathbb{P}[H \text{ is loopless and disconnected}] &+\\ \mathbb{P}[H \text{ is loopless and connected}],&\end{aligned}$$

but

$$\mathbb{P}[H \text{ is loopless and disconnected}] \leq \mathbb{P}[H \text{ is disconnected}] = \mathcal{O}(1/n^2),$$

by Theorem 6.9, because each vertex has degree at least 4, hence,

$$\mathbb{P}[H \text{ is loopless and connected}] \geq \exp(-2) \pm o(1) - \mathcal{O}(1/n^2) = \exp(-2) \pm o(1).$$

Therefore, $\mathbb{P}[G_n \text{ is simple, triangle-free and connected}] = \exp(-2) \pm o(1)$. □

Altough not necessary for our proofs, it is worth noting the uniformness of the distribution of outputs of Algorithm 6.3.

Proposition 6.11. *The distribution of outputs of Algorithm 6.3 with input $\overline{L_n}$ is uniform even if we condition on the output being simple, triangle-free and connected.*

Proof. Let G_n be the output graph. As vertices are chosen uniformly at random, the edges are added uniformly at random. There are $(2m-1)!! = (2m-1)(2m-3)\cdots 3 \cdot 1$ possible matchings, where $2m = n$ if n is even and $2m = n-1$ if n is odd. This is because there are $2m$ endpoints and we can choose the m edges in $\binom{2m}{2}\binom{2m-2}{2}\cdots\binom{2}{2}$ ways, but as the order of the choices does not matter, each matching is counted $m!$ times, so the total number is

$$\frac{1}{m!}\binom{2m}{2}\binom{2m-2}{2}\cdots\binom{2}{2} = \frac{(2m)(2m-1)\cdots 2\cdot 1}{m!2^m} = (2m-1)!!.$$

Hence, each graph in the output's distribution happens with probability $1/(2m-1)!!$.

If we want to condition on G_n being simple, triangle-free and connected, let $\mathcal{G}$ be the family of such output graphs and note that

$$\mathbb{P}[G_n = G \mid G_n \in \mathcal{G}] = \frac{\mathbb{P}[\{G_n = G\} \cap \{G_n \in \mathcal{G}\}]}{\mathbb{P}[G_n \in \mathcal{G}]},$$

for any $G \in \mathcal{G}$. But $\{G_n = G\} \subseteq \{G_n \in \mathcal{G}\}$, thus,

$$\mathbb{P}[G_n = G \mid G_n \in \mathcal{G}] = \frac{\mathbb{P}[G_n = G]}{\mathbb{P}[G_n \in \mathcal{G}]} = \frac{1}{(2m-1)!!}\frac{1}{\mathbb{P}[G_n \in \mathcal{G}]},$$

which does not depend in G. □

We are now ready to build our resistant instances. We achieve this by repeating Algorithm 6.3 until the output is connected and none of its edges have endpoints in the same C_5. We also perform a small trick to create instances for values of n not multiple of 5. This is described in Algorithm 6.4.

Algorithm 6.4. ResistantChromaticInstance(n)

```
1 k ← n (mod 5)
2 do
3 |   G ← AddMatching(complement(L_{n-k}))
4 while G is not simple or is not triangle-free or is disconnected
5 if k = 0 then
6 |   return complement(G)
7 return complement(G) ∨ K_k
```

Algorithm 6.4 depends on repeating Algorithm 6.3 an unknown number of times and, thus, it need not terminate at all, but this event happens with probability 0.

Proposition 6.12. *Algorithm 6.4 runs in time $\Theta(n^2)$ with probability at least $1 - 1/n^{\mathcal{O}(n/\lg n)}$.*

Proof. We can build $\overline{L_{n-k}}$ once in $\Theta(n)$ and cache it into memory to only spend linear time per iteration of line 3, as Algorithm 6.3 also runs in linear time. Furthermore, we do need to pay $\Theta(n^2)$ to build the complementary graph $\overline{G}$, as $e(\overline{G}) = \Theta(n^2)$. For large enough n, the expected number of times Algorithm 6.3 is called is at most 8, as the number of calls is a geometric random variable X with parameter p and $1/p \to \exp(2) \leq 8$ when $n \to \infty$ by Lemma 6.10. If we let $C \geq \exp(-2)$ be some constant such that $\mathbb{E}[X] \leq C$, then for any $c > 0$ we have

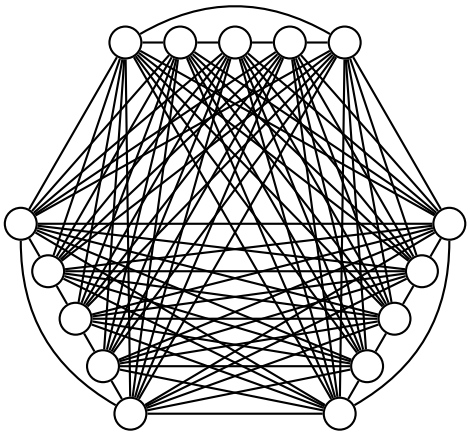
$$\begin{aligned}
\mathbb{P}[X > cn] &= 1 - \mathbb{P}[X \leq cn] \\
&= 1 - \sum_{i=1}^{\lfloor cn \rfloor} (1-p)^{i-1} p \\
&\leq (1-p)^{cn} \\
&\leq (1 - 1/C)^{cn} \\
&= n^{-\mathcal{O}(n/\lg n)},
\end{aligned}$$

so with probability $1 - 1/n^{\mathcal{O}(n/\lg n)}$ there are at most a linear number of calls to Algorithm 6.3, each costing linear time, giving us $\Theta(n^2)$ time. □

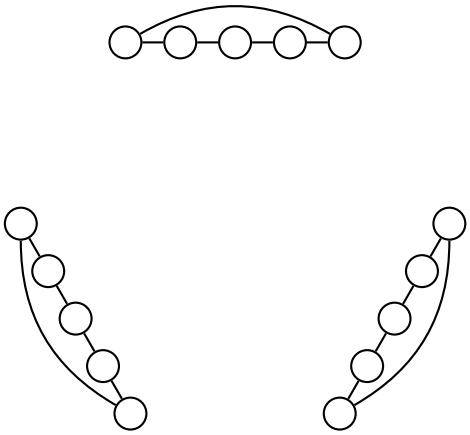
Figure 12 exemplifies an execution of this process. We finish this section showing that instances built by Algorithm 6.4 are indeed hard.

Figure 12 – An execution of Algorithm 6.4 with $n = 15$.

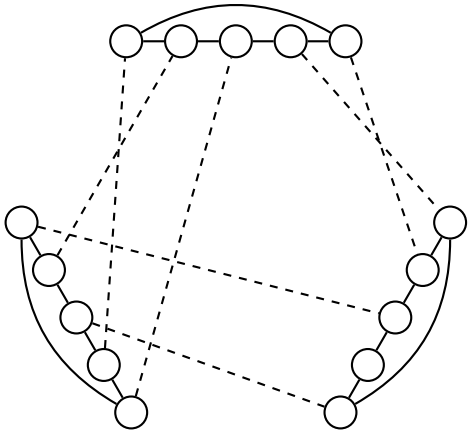
(a) The initial graph L_{15}.



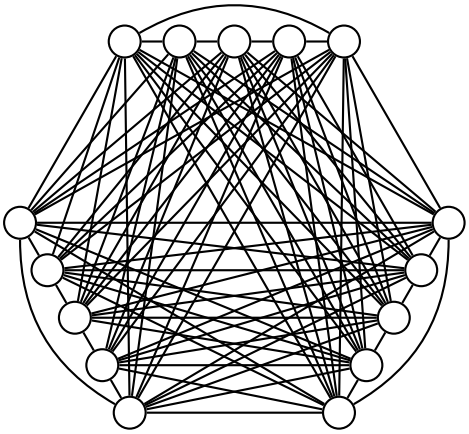
(b) The complementary graph $\overline{L_{15}}$.



(c) A maximum matching M is sampled uniformly at random until it has no edge internal to any C_5.



(d) The final graph $\overline{\overline{L_{15}} + M}$, which is immune to the preprocessing step.



Source: prepared by the author.

Theorem 6.13. *For any $n \in \mathbb{N}$, Algorithm 6.4 with input n outputs a graph on n vertices in which any execution of a χ-bounded algorithm needs $\Omega(2^{n/15})$ subinstances to terminate even if Algorithm 6.1 is used beforehand.*

Proof. Suppose $n \equiv 0 \pmod 5$ and let H_n be the graph given as input to Algorithm 6.1 (together with any χ-bounded algorithm). In this case, both H_n and its complement are connected. Moreover, $\delta(H_n) = n - 4$ and $\alpha(H_n) = 2$, so the χ-pruned subtree of any clique search tree of H_n has $\Omega(2^{n/15})$ nodes by Lemma 6.7, hence, the χ-bounded algorithm needs

to solve $\Omega(2^{n/15})$ subinstances until optimality. Now, if $n \equiv k \pmod 5$ for $1 \leq k \leq 4$, after Algorithm 6.1 is called on H_n the new instances are G_{n-k}, which is the output of Algorithm 6.3 with input $\overline{L_{n-k}}$, and at most 4 trivial graphs. The trivial graphs' χ-pruned subtrees have only one node, but G_{n-k} exhibits $\Omega(2^{(n-k)/15})$ nodes in any χ-pruned subtree by Lemma 6.7. Hence, there are $\Omega(2^{n/15})$ nodes in total, as k is constant.

Finally, if H_n is given as input directly to a χ-bounded algorithm which does not apply Algorithm 6.1, the result is similar and to show this we extend the proof of Lemma 6.7. For any k, we have $\delta(H_n) = n - 4$, $\alpha(H_n) = 2$ and

$$\omega(H_n) = \omega(G_{n-k}) + \omega(K_k) \leq \omega(L_{n-k}) + \omega(K_k) = \frac{2(n-k)}{5} + k = \frac{2n}{5} + \frac{3k}{5}.$$

Therefore, if less than $n/15 - 2$ branching operations have been made before a node (Q, R) is reached, then

$$\chi(H_n[Q \cup R]) > \frac{n - (n/15 - 2) \cdot 3}{2} = \frac{2n}{5} + 3 > \omega(H_n),$$

given that $3 > 3k/5$ for any $k \in \{1, 2, 3, 4\}$. Therefore, we have not reached a leaf yet, so there are $\Omega(2^{n/15})$ nodes in total. □

6.5 A fractional bounding rule

Building on the idea of using graph parameters as upper bounds in standard algorithms, the *fractional chromatic number* of a graph can be a particularly useful notion, as it is directly connected with that of the chromatic number. A classical definition of the fractional chromatic number is the optimal value of the linear relaxation of the MINIMUM VERTEX COLORING integer programming model given by (2.2), which is the dual of the linear relaxation of the MAXIMUM CLIQUE model given by (2.1). Particularly, for any graph G, we have $\chi_f(G) \leq \chi(G)$, because of linear duality.

Now, evaluating $\chi_f(G)$ for a graph G in the general case is still NP-hard [69] but there are ways to cope with the complexity of this evaluation, such as column generation-based methods for the linear program [70]. The benefit here is that the inequality

$$\omega(G) \leq \chi_f(G) \tag{6.2}$$

is tighter than (6.1), its usual discrete counterpart, and the gap between $\chi_f(G)$ and $\chi(G)$ can be arbitrarily large [71].

If we heuristically bound $\chi_f(G)$ by some value strictly smaller than $\chi(G)$, then this value is a better upper bound than any heuristic that approximates $\chi(G)$. Balas and Xue [72], for instance, propose a B&B algorithm which uses an estimate on the fractional chromatic number as an upper bound and remark that the size of the search three was cut almost in half compared to regular B&B methods that used estimates on the chromatic number.

In this sense, we define χ_f-*bounded* algorithms in an analogous way to χ-bounded algorithms, except now an upper bound on the fractional chromatic number is used, and χ_f-*pruned* subtrees in a similar fashion to that in Definition 6.1, except now in every leaf (Q, R) of a χ_f-pruned subtree of a graph G we have $\chi_f(G[Q \cup R]) \leq \omega(G)$. We now wish to prove analogous results to these types of algorithms and trees.

Lemma 6.14 (See e.g. Scheinerman and Ullman [73])**.** *For any graph G, $\chi_f(G) \geq v(G)/\alpha(G)$.*

This lemma is enough to adapt Lemma 6.7 to these trees.

Lemma 6.15. *For any $n, d \in \mathbb{N}$ with $n \equiv 0 \pmod 5$, if G is a spanning subgraph of L_n where $\alpha(G) \leq 2$ and $\delta(G) \geq n - d - 1$, then the χ_f-pruned subtree of every clique search tree of G has $\Omega(2^{n/(5d)})$ nodes.*

Proof. If less than $n/(5d)$ branching operations have been made in a path from the root to a node (Q, R), then less than $n/5$ vertices have been discarded and, by Lemma 6.14,

$$\chi_f(G[Q \cup R]) \geq \frac{n(G[Q \cup R])}{\alpha(G[Q \cup R])} > \frac{n - n/5}{2} = \frac{2n}{5} = \omega(L_n) \geq \omega(G),$$

so any χ_f-pruned subtree has $\Omega(2^{n/(5d)})$ nodes. □

We end this chapter by showing that instances given by Algorithm 6.4 also work for χ_f-bounded algorithms.

Theorem 6.16. *For any $n \in \mathbb{N}$, Algorithm 6.4 with input n outputs a graph on n vertices in which any execution of a χ_f-bounded algorithm needs $\Omega(2^{n/15})$ subinstances to terminate even if Algorithm 6.1 is used beforehand.*

Proof. If $n \equiv 0 \pmod 5$, then both the output H_n of Algorithm 6.4 and its complement are connected, $\delta(H_n) = n - 4$ and $\alpha(H_n) = 2$, so by Lemma 6.15 the χ_f-pruned subtree of any of its clique search trees has size $\Omega(2^{n/15})$. If $n \equiv k \pmod 5$ for $1 \leq k \leq 4$, then the χ_f-bounded algorithm needs to solve G_{n-k}, the output of Algorithm 6.3 with input $\overline{L_{n-k}}$, and at most 4 trivial graphs. The χ_f-pruned subtree of any clique search tree of G_{n-l} has $\Omega(2^{n/15})$ nodes as $k = \Theta(1)$. Finally, if H_n is given as input directly to a χ_f-bounded algorithm that does not apply Algorithm 6.1, note that $\omega(H_n) = 2n/5 + 3k/5$ and if less than $n/15 - 2$ branching operations have been made in a path from the root to a node (Q, R) in the χ_f-pruned subtree of some clique search tree of H_n, then

$$\chi_f(H_n[Q \cup R]) > \frac{n - (n/15 - 2) \cdot 3}{2} = \frac{2n}{5} + 3 > \omega(H_n),$$

where the first inequality also comes from Lemma 6.14. Hence, (Q, R) is not a leaf, so any χ_f-pruned subtree has $\Omega(2^{n/15})$ nodes. □

7 COMPUTATIONAL EXPERIMENTS

In this chapter we discuss empirical results gathered from tests with standard algorithms. The input instances were chosen to match the theoretical results approached throughout this text, namely Theorems 3.5, 5.9, 5.12, 6.4 and 6.13.

7.1 The setup

The computational environment consists of an Intel Core i7-12700 CPU with 24 GB of RAM running Debian Linux. All algorithms are implemented in `Python3` and transpiled to `C` via `Cython` [74]. They make use of `NetworkX` [75], a `Python3` package for storing and manipulating graphs. The implementation of the clique algorithms are due to Carmo and Züge [76] and these are available in a public code repository [77].

The algorithms return the size of the largest clique found, the number of subinstances solved and the solving time. Each instance is given at most 48 h (i.e., 172800 s) of CPU time before the algorithm halts the computation and returns the information gathered so far.

7.2 The instances

The first family of instances, denoted by `gnp`, consists of graphs in the $\mathcal{G}(n, p)$ model for $p = 1/2$. This choice for p is arbitrary, as any constant value of p, no matter how close to 1, would lead to $n^{\Theta(\lg n)}$ cliques w.h.p. in the graph, the main concern here is a rough idea in practice of the growth rate of the number of cliques. There are 210 graphs in this family, 10 for each $n \in \{50, 60, \ldots, 250\}$. As random graphs are non-deterministic by definition, we are interested on the average results of the 10 instances for each value of n, in order to bring the sample average closer to the actual average over all graphs in the model.

The second family of instances, denoted by `cj_0.5`, consists of graphs built by Algorithm 5.3, which uses the Cornaz–Jost reduction. The algorithm asks for n and ε, parameters that control the numbers of vertices and cliques in the output graph (i.e., after the reduction), respectively. For this family, we set $\varepsilon = 1/2$ and $n \in \{50, 60, \ldots, 250\}$. There are 210 instances, 10 graphs for each value of n, as they are also non-deterministic, and we average the values for each n. Algorithm 5.3 uses internally two other parameters: N and c, which control the order and size of the $\mathcal{G}(N, m)$ graph that is fed to the Cornaz–Jost reduction. These parameters do affect the number of cliques in the output graph, although their effect is hidden in the Θ notation. There are multiple choices for $c \in (0, 1]$ and $N \in \mathbb{N}$, so we choose specifically $c \in [1/2 - \delta, 1/2 + \delta]$, with δ initially set to 0, and the smallest possible N and we slowly increase the value of δ if there is no choice for N until the first feasible pair (c, N) is determined. Finally, when the base random graph is sampled, the linear ordering over its vertices is implicitly sampled by `NetworkX`, as it stores an integer label for each vertex and, hence, these are naturally ordered by their labels.

The third family of instances, denoted by `cj_0.1`, also consists of graphs built by Algorithm 5.3, but with $\varepsilon = 1/10$. We choose c, N and the linear order for the $\mathcal{G}(N, m)$ graph in the same way done for `cj_0.5`. Again, there are 210 instances, being 10 graphs for each value of $n \in \{50, 60, \ldots, 250\}$.

The fourth family of instances, denoted by `gnm_0.5`, consists of 210 $\mathcal{G}(n, m)$ graphs that match the order and size of those in the `cj_0.5` family, 10 for each value of $n \in \{50, 60, \ldots, 250\}$. These graphs are *not* the $\mathcal{G}(n, m)$ graphs fed to the Cornaz–Jost reduction; `gnm_0.5` instances are sampled *after* each `cj_0.5` instance is built, copying its number of vertices and edges. This family is related to Theorem 5.12, which gives the expected number of cliques in random graphs with a density similar to the instances built by Algorithm 5.3. We choose specifically the $\mathcal{G}(n, m)$ model in order to build graphs with order and size exatcly equal to those in the `cj_0.5` instances.

The fifth family of instances, denoted by `gnm_0.1`, also consists of $\mathcal{G}(n, m)$ graphs, but now matching the order and size of those in the `cj_0.1` family. There are again 210 instances, 10 for each $n \in \{50, 60, \ldots, 250\}$.

The sixth family, denoted by `lav`, consists of Lavnikevich graphs L_n for $n \in \{50, 60, \ldots, 100\}$. These are only 6 graphs in total, as they are deterministic and multiple executions of a standard algorithm yield the same results. No graph with over 100 vertices was generated, as they proved to be indeed challenging for standard algorithms that do not implement Algorithm 6.1 and their tests very time consuming.

The seventh family, denoted by `p_lav`, consists of the perturbed Lavnikevich graphs, built by Algorithm 6.4. There are 60 graphs in total for this family, 10 for each value of $n \in \{50, 60, \ldots, 100\}$, and we average their results, as they are non-deterministic by nature.

7.3 The algorithms

We perform tests using two standard algorithms. The first, denoted by `nb` (short for "no bound"), is the simplest method that implements the algorithmic framework of Chapter 3, namely one that does not use upper bounds and chooses the pivot as the first vertex in the array that represents the R set of an instance (Q, R).

The second, denoted by `kj` (short for "Konc and Janežič"), is a χ-bounded algorithm due to Konc and Janežič [29]. Besides the chromatic upper bound, it uses a coloring-based approach to choose the pivot vertices. We test the `kj` algorithm both when it is called after the preprocessing step described by Algorithm 6.1, denoted `pre`, and when it is called alone.

The choice of algorithms should illustrate the speed up due to the usage of chromatic upper bounds as well as the preprocessing step. Tests with the `nb` algorithm serve as the control group to analyze the efficiency of the upper bound, while those with the `kj` algorithm without the preprocess work analogously for the analysis of Algorithm 6.1.

7.4 The results

Tables 1 and 2 present results for the nb algorithm. Tests with cj_0.1 and gnm_0.1 graphs are lacking due to the fact that not a single instance in these families was solved within the time limit. This is due to the sheer size of the search tree when ε approaches 0, according to Theorems 5.9 and 5.12.

Table 1 – Average results for gnp using the nb algorithm

	gnp nb		
$v(G)$	$\omega(G)$	#Nodes	Time (s)
50	7.4	17950.4	0.01963
60	7.9	38857.4	0.03258
70	8.4	85607.6	0.06920
80	8.8	142630.2	0.11483
90	9.3	245339.4	0.19299
100	9.2	420843.6	0.32665

Source: prepared by the author.

Table 2 – Average results for cj_0.5 and gnm_0.5 instances using the nb algorithm.

	cj_0.5 nb			gnm_0.5 nb		
$v(G)$	$\omega(G)$	#Nodes	Time (s)	$\omega(G)$	#Nodes	Time (s)
50	9.8	1 150 123.6	0.81397	14.3	5 251 176.2	3.80183
60	10.6	6 245 612.6	4.46790	16.1	44 394 617.8	32.71266
70	12.0	36 514 493.8	27.16286	18.2	342 798 512.6	248.64842
80	12.7	187 596 209.8	142.04105	19.6	2 369 198 936.2	1 710.04047
90	13.7	901 798 549.2	670.48727	21.3	17 849 004 923.6	12 638.46593
100	14.9	5 829 034 274.8	4 212.48873	23.6	158 508 641 538.6	109 416.05243

Source: prepared by the author.

Tables 3, 4, 5 and 6 regard the kj algorithm with the preprocessing step, while Table 7 regards kj alone. Rows with a $\geq$ in Table 7 denote the fact that the algorithm could not solve the instance in the prescribed time limit and reported only a lower bound for some parameters.

7.5 Discussion

Results from Tables 1 through 7 mostly support the theoretical analyses provided throughout this work.

Tables 1 and 2 show the asymptotic difference between the number of cliques in gnp, cj_0.5 and gnm_0.5 graphs. The plot in Figure 13 illustrates that cj_0.5 instances have much more cliques than those in gnp, as they differ by a polynomial factor in the exponent, and have almost as many cliques as those in the gnm_0.5, as they differ only by a logarithmic factor in the

Table 3 – Average results for gnp using the kj algorithm with the preprocessing step.

	gnp kj pre		
$v(G)$	$\omega(G)$	#Nodes	Time (s)
50	7.4	163.4	0.00933
60	7.9	223.6	0.00783
70	8.4	339.2	0.01109
80	8.8	414.6	0.01610
90	9.3	607.4	0.01873
100	9.2	957.6	0.02800
110	9.4	1 351.6	0.03487
120	9.6	1 863.0	0.04367
130	9.8	2 429.2	0.05431
140	10.3	2 806.4	0.06761
150	10.2	4 237.0	0.09035
160	10.2	5 564.8	0.12153
170	10.7	5 798.4	0.13588
180	10.8	7 980.2	0.17873
190	11.0	9 332.0	0.20799
200	10.8	14 683.4	0.27582
210	11.1	15 958.0	0.31435
220	11.0	20 260.0	0.38288
230	11.5	23 935.8	0.44882
240	11.8	29 755.0	0.54050
250	11.6	40 237.2	0.66763

Source: prepared by the author.

Table 4 – Average results for cj_0.5 and gnm_0.5 instances using the kj algorithm with the preprocessing step.

	cj_0.5 kj pre			gnm_0.5 kj pre		
$v(G)$	$\omega(G)$	#Nodes	Time (s)	$\omega(G)$	#Nodes	Time (s)
50	9.8	670.0	0.01458	14.3	375.8	0.01334
60	10.6	1 445.2	0.02059	16.1	569.8	0.01522
70	12.0	2 329.4	0.03756	18.2	1 022.2	0.02838
80	12.7	4 191.6	0.06901	19.6	2 218.8	0.05887
90	13.7	11 275.2	0.16670	21.3	5 215.6	0.12860
100	14.9	17 221.2	0.29720	23.6	12 305.8	0.30345
110	15.9	36 810.8	0.62146	25.8	19 149.2	0.52159
120	16.0	137 320.2	2.16757	26.8	39 885.6	1.21182
130	17.0	207 645.8	3.65368	28.4	84 199.0	2.90597
140	18.0	356 706.6	8.12672	29.5	263 332.8	8.95589
150	18.6	499 376.4	12.04382	30.9	689 712.4	24.93412
160	19.3	1 564 050.0	39.84404	32.8	995 064.8	43.92872
170	20.0	5 007 625.0	116.81393	34.5	1 982 150.0	97.58425
180	20.4	7 407 345.6	171.09369	35.5	4 067 036.2	199.74870
190	21.0	16 601 933.2	465.76651	37.2	8 113 260.2	413.44183
200	22.1	35 721 362.6	866.64247	38.4	18 865 344.2	915.99224
210	22.0	62 609 205.6	1 664.24800	39.6	42 807 068.4	2 199.00176
220	23.1	118 553 865.2	3 288.66730	40.7	80 724 767.4	4 636.81364
230	23.9	166 787 242.0	5 304.88972	42.1	181 817 357.0	11 447.75480
240	24.0	443 961 083.0	14 504.52673	43.2	286 101 024.0	19 059.11599
250	24.7	1 067 281 104.4	34 384.80519	44.7	664 868 647.6	43 864.86124

Source: prepared by the author.

Table 5 – Average results for `cj_0.1` and `gnm_0.1` instances using the `kj` algorithm with the preprocessing step.

	`cj_0.1 kj pre`			`gnm_0.1 kj pre`		
$v(G)$	$\omega(G)$	#Nodes	Time (s)	$\omega(G)$	#Nodes	Time (s)
50	29.1	127.0	0.01287	32.3	142.8	0.01364
60	34.1	157.1	0.01210	37.2	197.2	0.01507
70	40.7	182.0	0.01721	44.8	234.7	0.02283
80	46.7	212.4	0.02306	51.0	263.4	0.02730
90	50.2	238.8	0.02834	55.3	324.8	0.03971
100	55.5	258.6	0.03535	61.9	353.5	0.04946
110	61.7	282.0	0.04299	67.6	349.1	0.05721
120	67.7	309.8	0.04765	74.8	411.9	0.07189
130	72.6	338.6	0.05818	79.7	495.1	0.09420
140	78.0	369.5	0.06779	87.1	580.4	0.10926
150	83.8	398.3	0.07402	94.1	730.6	0.13941
160	89.3	448.0	0.08621	97.3	700.5	0.15528
170	95.4	442.6	0.09867	105.4	735.2	0.17802
180	100.8	466.6	0.10863	112.1	874.2	0.20683
190	104.6	580.5	0.12964	115.0	994.9	0.27076
200	111.3	529.6	0.13360	123.9	961.2	0.28021
210	115.4	646.9	0.14618	130.2	1 021.2	0.32954
220	121.1	587.4	0.16016	132.8	1 967.1	0.60132
230	126.1	647.1	0.17250	140.0	1 314.4	0.45654
240	129.5	699.9	0.19051	144.7	1 491.0	0.58461
250	136.4	694.8	0.20550	153.1	1 711.5	0.71111

Source: prepared by the author.

Table 6 – Average results for `lav` and `p_lav` instances using the `kj` algorithm with the preprocessing step.

	`lav kj pre`			`p_lav kj pre`		
$v(G)$	$\omega(G)$	#Nodes	Time (s)	$\omega(G)$	#Nodes	Time (s)
50	20	90	0.00377	20	9 284.6	0.11365
60	24	108	0.00452	24	36 021.2	0.56496
70	28	126	0.00589	28	184 452.8	3.68521
80	32	144	0.00714	32	922 907.2	22.43834
90	36	162	0.00880	36	4 008 797.8	113.20505
100	40	180	0.01036	40	16 588 140.0	562.53972

Source: prepared by the author.

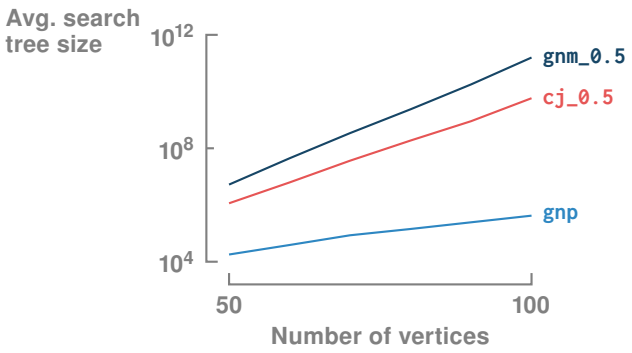
Table 7 – Average results for `lav` and `p_lav` instances using the `kj` algorithm alone.

	`lav kj`			`p_lav kj`		
$v(G)$	$\omega(G)$	#Nodes	Time (s)	$\omega(G)$	#Nodes	Time (s)
50	20	2 593 543	19.32607	20	9 284.6	0.11328
60	24	31 044 595	282.57747	24	36 021.2	0.58047
70	28	357 914 319	4 020.31492	28	184 452.8	3.78329
80	32	5 593 926 941	79 848.15856	32	922 907.2	22.14741
90	≥ 36	$\geq 11\,658\,023\,739$	172 800.00005	36	40 08 797.8	109.37013
100	≥ 40	$\geq 11\,687\,072\,031$	172 800.00020	40	16 588 140.0	546.67986

Source: prepared by the author.

exponent. We see that the graphs corresponding to `cj_0.5` and `gnm_0.5` instances seem to grow resembling a linear function, while the graph corresponding to gnp is slightly concave, which is expected when the exponent is only logarithmic.

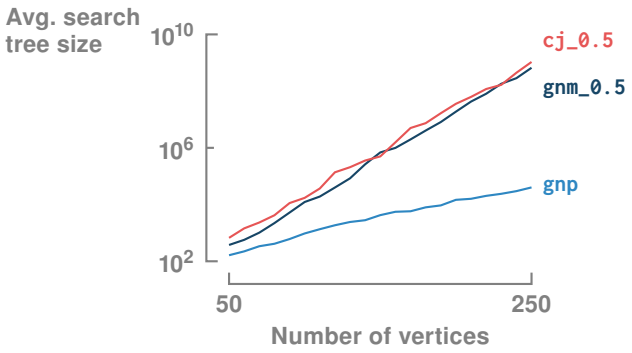
Figure 13 – Log-linear plot of the average search tree sizes when using the `nb` algorithm on gnp, `cj_0.5` and `gnm_0.5` instances vs. number of vertices.



Source: prepared by the author.

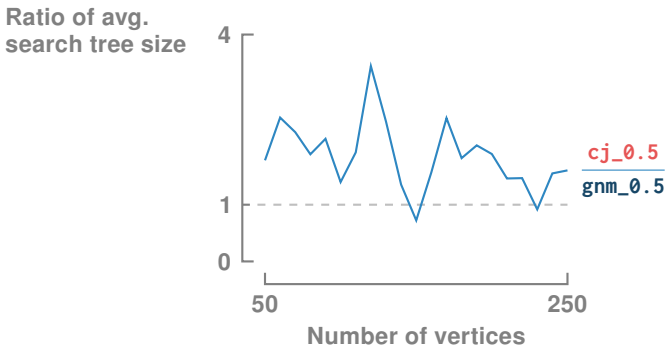
Tables 3 and 4 illustrate how strong the chromatic upper bound used by `kj` is, as the number of nodes in the search trees is several orders of magnitude smaller. They also indicate that most of the time the `cj_0.5` instances demand a larger search tree to be solved when compared to `gnm_0.5` (and even more when compared to gnp) if the `kj` algorithm is used, which implies that the chromatic upper bound adopted is more effective in random graphs for that specific density. Figure 14 shows that the average search trees' sizes grows in a similar fashion, while Figure 15 shows that although the `cj_0.5` family demands less nodes to be solved in two cases, overall its search trees are much larger than `gnm_0.5` ones, with the ratios peaking at 3.44285 and averaging at 1.84872, i.e., on average the `cj_0.5` instances demanded almost twice as many nodes to be solved. This supports the usage of such instances in benchmarks, as they seem to be more challenging, at least in the search tree size view.

Figure 14 – Log-linear plot of the average search tree sizes when using the `kj` algorithm with the preprocessing step on gnp, `cj_0.5` and `gnm_0.5` instances vs. number of vertices.



Source: prepared by the author.

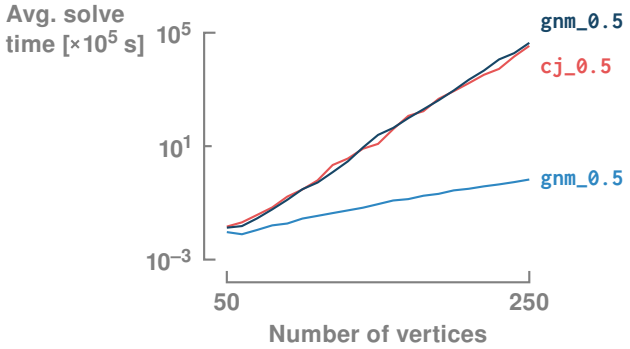
Figure 15 – Plot of the ratio between the average search tree sizes when using the kj algorithm with the preprocessing step on gnm_0.5 and cj_0.5 instances vs. number of vertices.



Source: prepared by the author.

Figure 16 shows that the average time to solve cj_0.5 and gnm_0.5 instances also grow very similarly. In Figure 17 we see few outliers where the gnm_0.5 graphs took more than twice or less than half as much time to solve when compared to cj_0.5 graphs of same order, which indicates that both families demand about the same time to be solved. In all other cases, the ratio between times is close to 1 with the average ratio being 1.09285 and gnm_0.5 taking more time in 11 out of the 21 averaged tests, which leaves both families almost tied in hardness time wise. Graphs in the cj_0.5 family need a larger search tree for kj, but gnm_0.5 graphs need slightly more time, hence the chromatic upper bound seems to run faster on the former instances, although it seems to prune better in the latter ones. This is an interesting behaviour that calls for tests using different upper bound rules and this is most definitely a priority in future works.

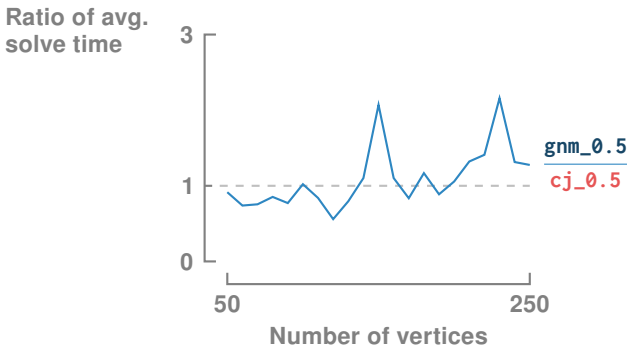
Figure 16 – Log-linear plot of the average solve times when using the kj algorithm with the preprocessing step on gnp, cj_0.5 and gnm_0.5 instances vs. number of vertices.



Source: prepared by the author.

Table 5 displays a very different situation for cj_0.1 and gnm_0.1 when compared to their _0.5 counterparts. These instances are now solved much faster and demand very few nodes to be solved by kj, even less than gnp. In Figure 18 we see that the cj_0.1 family consistently exhibits a smaller search tree size when compared to gnm_0.1 and they both loose by a fair amount to simple gnp instances. This is aligned to the fact that complete graphs, for example,

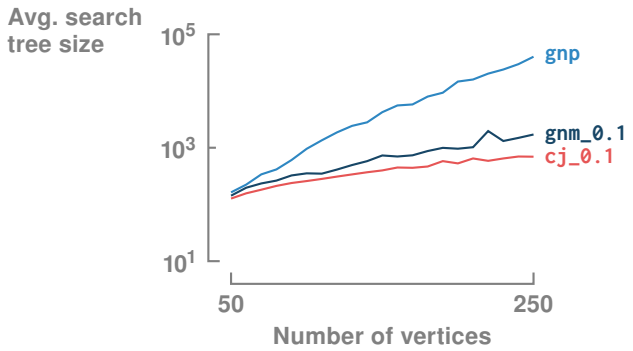
Figure 17 – Plot of the ratio between the average solve times when using the kj algorithm with the preprocessing step on gnm_0.5 and cj_0.5 instances vs. number of vertices.



Source: prepared by the author.

are far too easy to solve, as they are quickly handled by heuristics that find optimal colorings. Proposition 5.11 tells us that both cj_0.1 and gnm_0.1 instances have edge density close to 1 and this is likely what causes both of them to be easier to solve than the gnp family. Thus, denser graphs can be harder than random graphs with constant density, but not always.

Figure 18 – Log-linear plot of the average search tree sizes when using the kj algorithm with the preprocessing step on gnp, cj_0.1 and gnm_0.1 instances vs. number of vertices.

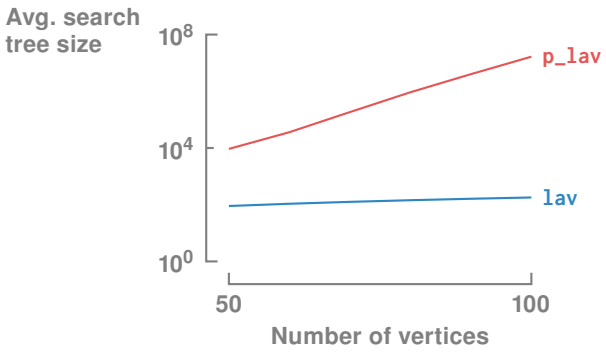


Source: prepared by the author.

Table 6 shows how the p_lav instances endure the preprocessing of Algorithm 6.1 as opposed to the lav ones. Indeed, the growth of the search tree sizes for lav is perfectly linear. In Figure 19 we see that the log-linearly lav graph looks almost constant, as it is actually a very slow-growing logarithm, while the p_lav graph grows resembling a linear function, indicating that the search trees' sizes for p_lav grow exponentially fast.

Table 7 displays how much larger the search trees for lav instances must be when the preprocessing step is not used. For $n = 90$ and $n = 100$, the kj is not even able to solve the lav instances to optimality in the time limit, proving that these instances are indeed challenging for this type of algorithm when lacking the preprocess. Figure 20 illustrates how the search trees for both the lav and p_lav instances grows exponentially, as their log-linearly graphs grow almost as a linear function. However, it is clear that when Algorithm 6.1 is not applied, the search

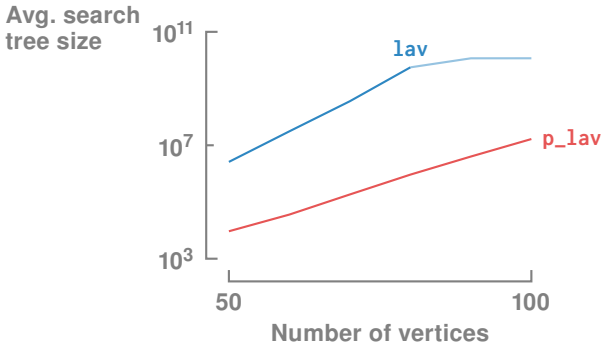
Figure 19 – Log-linear plot of the average search tree sizes when using the `kj` algorithm with the preprocessing step on `lav` and `p_lav` instances vs. number of vertices.



Source: prepared by the author.

tree size for the `p_lav` family looses by a fair amount to that of the `lav`, which is expected by the worst lower bound on its size given by Theorem 6.13. Even so, the theorem describing the exponential growth allied to the positive test results when the preprocessing step is used provide an use case in which `p_lav` instances are very demanding for state-of-the-art algorithms.

Figure 20 – Log-linear plot of the average search tree sizes when using the `kj` algorithm alone on `lav` and `p_lav` instances vs. number of vertices. Lighter line segments indicate the tests did not finish.



Source: prepared by the author.

8 CONCLUSIONS

Intuitively, generating hard instances to NP-hard problems should be trivial. For MAXIMUM CLIQUE this is not the case, as the worst case is far from the average one. The instances provided in this work are a step towards the goal of obtaining challenging inputs to this problem. Although it is common practice to test new algorithms with instances that are known to be hard simply because historically they take a long time to be solved, we present theoretical hardness analyses as a way to explain *why* such instance are expected to be hard.

For the instances with many cliques given by Algorithm 5.3, there is still potential for future work analyzing their behaviour under specific upper bound rules. Computational experiments empirically show that they are competitive when given as input to χ-bounded algorithms, as we have seen in Chapter 7. Even though the chromatic upper bound seems to run faster when applied to these instances, it also seems to prune less nodes in the search tree than it does when applied to random graphs of matching density. This motivates the study of the behaviour of these instances under the chromatic upper bound point of view, similarly to what is done in Theorems 6.4, 6.13 and 6.16, specially because their cliques are deeply connected with colorings of random graphs, so their colorings could also be related to some other structure.

There is also space for experiments with other upper bounds, as positive results could motivate a more in depth theoretical analysis. Another path to be explored is the study of different reductions from other problems to MAXIMUM CLIQUE and how the final graph instances stand against efficient clique algorithms, both in number of cliques and in resilience to upper bounds.

For the χ-bounded algorithms specialized instaces given by Algorithm 6.4, a natural follow up is the analysis of their hardness for other infra-chromatic upper bounds. Furthermore, it could be the case that the strategy used to make them immune to the preprocessing works for other objectives, such as taking a dense graph that is easy to solve and remove edges in a randomized way to prevent algorithms from closing the optimality gap quickly.

Another possibility is the study of different preprocessing steps for standard algorithms. The first objective would be to generalize Algorithm 6.1 into a Modular or even a Primeval Decomposition and analyse how our instances stand against them. Besides these two decompositions, there are also other ideas to reduce the input graph size, such as identifying universal vertices or false-twins to name a few.

Finally, we leave open for future work the exploration of both the enumeration complexity and the contrast of worst and average case of other graph problems, for different distributions of input graphs.

REFERENCES

1 SAN SEGUNDO, P.; RODRÍGUEZ-LOSADA, D.; MATÍA, F.; GALÁN, R. Fast exact feature based data correspondence search with an efficient bit-parallel mcp solver. *Applied Intelligence*, v. 32, n. 3, p. 311–329, Jun 2010. ISSN 1573-7497.

2 SAN SEGUNDO, P.; RODRÍGUEZ-LOSADA, D. Robust global feature based data association with a sparse bit optimized maximum clique algorithm. *Trans. Rob.*, IEEE Press, v. 29, n. 5, p. 1332–1339, oct 2013. ISSN 1552-3098.

3 HOTTA, K.; TOMITA, E.; TAKAHASHI, H. A view-invariant human face detection method based on maximum cliques. *Trans. IPSJ*, v. 44, p. 57–70, 01 2003.

4 STENTIFORD, F. Face recognition by detection of matching cliques of points. In: NIEL, K. S.; BINGHAM, P. R. (Ed.). *Image Processing: Machine Vision Applications VII*. [S.l.]: SPIE, 2014. v. 9024.

5 BERMAN, P.; PELC, A. Distributed probabilistic fault diagnosis for multiprocessor systems. In: *Digest of Papers. Fault-Tolerant Computing: 20th International Symposium*. Los Alamitos, CA, USA: IEEE Computer Society, 1990. p. 340–346.

6 DUARTE JR., E.; GARRETT, T.; BONA, L.; CARMO, R.; ZÜGE, A. Finding stable cliques of planetlab nodes. In: *2010 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. [S.l.]: IEEE, 2010. p. 317–322. ISBN 978-1-4244-7500-1.

7 BROUWER, A.; SHEARER, J.; SLOANE, N.; SMITH, W. A new table of constant weight codes. *IEEE Transactions on Information Theory*, v. 36, n. 6, p. 1334–1380, 1990.

8 SLOANE, N. J. A. On single-deletion-correcting codes. In: ARASU, K. T.; SERESS, A. (Ed.). *Codes and Designs: Proceedings of a conference honoring Professor Dijen K. Ray-Chaudhuri on the occasion of his 65th birthday. The Ohio State University May 18–21, 2000*. [S.l.]: De Gruyter, 2002. p. 273–292. ISBN 9783110198119.

9 BUTENKO, S.; PARDALOS, P.; SERGIENKO, I.; SHYLO, V.; STETSYUK, P. Estimating the size of correcting codes using extremal graph problems. In: ____. [S.l.]: Springer, 2009. p. 227–243. ISBN 978-0-387-98095-9.

10 BUTENKO, S.; WILHELM, W. Clique-detection models in computational biochemistry and genomics. *European Journal of Operational Research*, v. 173, n. 1, p. 1–17, 2006. ISSN 0377-2217.

11 FUKAGAWA, D.; TAMURA, T.; TAKASU, A.; TOMITA, E.; AKUTSU, T. A clique-based method for the edit distance between unordered trees and its application to analysis of glycan structures. *BMC Bioinformatics*, v. 12, n. 1, p. S13, Feb 2011. ISSN 1471-2105.

12 HARARY, F.; ROSS, I. C. A procedure for clique detection using the group matrix. *Sociometry*, American Sociological Association, Sage Publications, Inc., v. 20, n. 3, p. 205–215, 1957. ISSN 00380431, 23257938.

13 KARP, R. Reducibility among combinatorial problems. In: *Complexity of computer computations*. [S.l.]: Springer, 1972. p. 85–103.

14 NEMHAUSER, G. L.; TROTTER, L. E. Properties of vertex packing and independence system polyhedra. *Mathematical Programming*, v. 6, n. 1, p. 48–61, Dec 1974. ISSN 1436-4646.

15 CARRAGHAN, R.; PARDALOS, P. An exact algorithm for the maximum clique problem. *Operations Research Letters*, v. 9, n. 6, p. 375–382, 1990. ISSN 0167-6377.

16 BABEL, L.; TINHOFER, G. A branch and bound algorithm for the maximum clique problem. *Zeitschrift für Operations Research*, v. 34, n. 3, p. 207–217, May 1990. ISSN 1432-5217.

17 BALAS, E.; XUE, J. Minimum weighted coloring of triangulated graphs, with application to maximum weight vertex packing and clique finding in arbitrary graphs. *SIAM Journal on Computing*, v. 20, n. 2, p. 209–221, 1991.

18 BABEL, L. A fast algorithm for the maximum weight clique problem. *Computing*, v. 52, n. 1, p. 31–38, Mar 1994. ISSN 1436-5057.

19 ÖSTERGÅRD, P. R. A new algorithm for the maximum-weight clique problem. *Electronic Notes in Discrete Mathematics*, v. 3, p. 153–156, 1999. ISSN 1571-0653. 6th Twente Workshop on Graphs and Combinatorial Optimization.

20 LI, C.-M.; FANG, Z.; JIANG, H.; XU, K. Incremental upper bound for the maximum clique problem. *INFORMS Journal on Computing*, v. 30, n. 1, p. 137–153, 2018.

21 SAN SEGUNDO, P.; FURINI, F.; ÁLVAREZ, D.; PARDALOS, P. M. Clisat: A new exact algorithm for hard maximum clique problems. *European Journal of Operational Research*, v. 307, n. 3, p. 1008–1025, 2023. ISSN 0377-2217.

22 CARMO, R.; ZÜGE, A. On comparing algorithms for the maximum clique problem. *Discrete Applied Mathematics*, v. 247, 02 2018.

23 DIMACS. *The DIMACS Second Implementation Challenge Clique Benchmark*. 1992–1993. Available in <http://archive.dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique/>.

24 XU, K. *BHOSLIB: Benchmarks with Hidden Optimum Solutions for Graph Problems (Maximum Clique, Maximum Independent Set, Minimum Vertex Cover and Vertex Coloring)*. 2004. Available in <http://vlsicad.eecs.umich.edu/BK/Slots/cache/www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm>.

25 BROCKINGTON, M. G.; CULBERSON, J. C. Camouflaging independent sets in quasi-random graphs. In: *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*. [S.l.]: American Mathematical Society, 1996. v. 26.

26 LAVNIKEVICH, N. On the complexity of maximum clique algorithms: usage of coloring heuristics leads to the $\Omega(2^{n/5})$ running time lower bound. Preprint available in <http://arxiv.org/abs/1303.2546>. 2013.

27 ZUCKERMAN, D. Linear degree extractors and the inapproximability of max clique and chromatic number. In: *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*. New York, NY, USA: Association for Computing Machinery, 2006. (STOC '06), p. 681–690. ISBN 1595931341.

28 DOWNEY, R.; FELLOWS, M. Parameterized computational feasibility. In: CLOTE, P.; REMMEL, J. (Ed.). *Feasible Mathematics II*. Boston, MA: Birkhäuser Boston, 1995. p. 219–244.

29 KONC, J.; JANEŽIČ, D. An improved branch and bound algorithm for the maximum clique problem. *MATCH Communications in Mathematical and in Computer Chemistry*, jun 2007.

30 WALTEROS, J. L.; BUCHANAN, A. Why is maximum clique often easy in practice? *Operations Research*, v. 68, n. 6, p. 1866–1895, 2020.

31 JOHNSON, D.; TRICK, M. (Ed.). *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11–13, 1993*. Boston, MA, USA: American Mathematical Society, 1996. v. 26. ISBN 0821866095.

32 WEST, D. *Introduction to Graph Theory*. [S.l.]: Prentice Hall, 2001. (Featured Titles for Graph Theory). ISBN 9780130144003.

33 BONDY, A.; MURTY, U. *Graph Theory*. [S.l.]: Springer London, 2011. (Graduate Texts in Mathematics). ISBN 9781846289699.

34 AUSIELLO, G.; MARCHETTI-SPACCAMELA, A.; CRESCENZI, P.; GAMBOSI, G.; PROTASI, M.; KANN, V. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999. ISBN 978-3-642-58412-1.

35 VAZIRANI, A. U.; PAPADIMITRIOU, A. C. H.; DASGUPTA, A. S. *Algorithms*. [S.l.]: McGraw-Hill Education, 2006. ISBN 9780073523408.

36 WOLSEY, L. *Integer Programming*. [S.l.]: John Wiley & Sons, Ltd, 2020. ISBN 9781119606475.

37 BOLLOBÁS, B. *Random Graphs*. [S.l.]: Cambridge University Press, 2001. (Cambridge Studies in Advanced Mathematics). ISBN 9780521797221.

38 MITZENMACHER, M.; UPFAL, E. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. 2nd. ed. USA: Cambridge University Press, 2017. ISBN 110715488X.

39 CORLESS, R. M.; GONNET, G. H.; HARE, D. E. G.; JEFFREY, D. J.; KNUTH, D. E. On the lambert w function. *Advances in Computational Mathematics*, v. 5, n. 1, p. 329–359, Dec 1996. ISSN 1572-9044.

40 HOORFAR, A.; HASSANI, M. Inequalities on the lambert function and hyperpower function. *Journal of Inequalities in Pure & Applied Mathematics (electronic only)*, Victoria University, School of Communications and Informatics, v. 9, n. 2, 2008.

41 WU, Q.; HAO, J.-K. A review on algorithms for maximum clique problems. *European Journal of Operational Research*, v. 242, n. 3, p. 693–709, 2015. ISSN 0377-2217.

42 BRON, C.; KERBOSCH, J. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 16, n. 9, p. 575–577, set. 1973. ISSN 0001-0782.

43 FAHLE, T. Simple and fast: Improving a branch-and-bound algorithm for maximum clique. In: MÖHRING, R.; RAMAN, R. (Ed.). *Proceedings of the 10th Annual European Symposium on Algorithms (ESA 2002)*. Berlin, Heidelberg: Springer, 2002. (Lecture Notes in Computer Science, v. 2461), p. 485–498. ISBN 978-3-540-45749-7.

44 TOMITA, E.; TANAKA, A.; TAKAHASHI, H. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, v. 363, n. 1, p. 28–42, out. 2006. ISSN 03043975.

45 TOMITA, E.; KAMEDA, T. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global Optimization*, Springer, v. 37, n. 1, p. 95–111, jan. 2007. ISSN 0925-5001.

46 CHVÁTAL, V. Determining the stability number of a graph. *SIAM Journal on Computing*, v. 6, n. 4, p. 643–662, 1977.

47 PITTEL, B. On the probable behaviour of some algorithms for finding the stability number of a graph. *Mathematical Proceedings of the Cambridge Philosophical Society*, v. 92, p. 511–526, nov. 1982. ISSN 1469-8064.

48 WOOD, D. R. On the maximum number of cliques in a graph. *Graphs and Combinatorics*, v. 23, n. 3, p. 337–352, Jun 2007. ISSN 1435-5914.

49 MILLER, R. E.; MÜLLER, D. E. *A problem of maximum consistent subsets*. [S.l.], 1960.

50 MOON, J. W.; MOSER, L. On cliques in graphs. *Israel Journal of Mathematics*, v. 3, n. 1, p. 23–28, mar. 1965.

51 TURÁN, P. On an extremal problem in graph theory. *Matematikai és Fizikai Lapok*, v. 48, p. 436–452, 1941.

52 ZYKOV, A. A. On some properties of linear complexes. *Matematicheskii Sbornik*, v. 24(66), p. 163–188, 1949.

53 CORRÁDI, K.; SZABÓ, S. A combinatorial approach for keller's conjecture. *Periodica Mathematica Hungarica*, v. 21, n. 2, p. 95–100, Jun 1990. ISSN 1588-2829.

54 SANCHIS, L. A. Generating hard and diverse test sets for np-hard graph problems. *Discrete Applied Mathematics*, v. 58, n. 1, p. 35–66, 1995. ISSN 0166-218X.

55 GENDREAU, M.; SORIANO, P.; SALVAIL, L. Solving the maximum clique problem using a tabu search approach. *Annals of Operations Research*, v. 41, n. 4, p. 385–403, Dec 1993. ISSN 1572-9338.

56 MANNINO, C.; SASSANO, A. Solving hard set covering problems. *Operations Research Letters*, v. 18, n. 1, p. 1–5, 1995. ISSN 0167-6377.

57 KUČERA, L. A generalized encryption scheme based on random graphs. In: SCHMIDT, G.; BERGHAMMER, R. (Ed.). *Graph-Theoretic Concepts in Computer Science*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992. p. 180–186. ISBN 978-3-540-46735-9.

58 XU, K.; LI, W. Many hard examples in exact phase transitions. *Theoretical Computer Science*, v. 355, n. 3, p. 291–302, 2006. ISSN 0304-3975.

59 CAMPÊLO, M.; CAMPOS, V.; CORRÊA, R. On the asymmetric representatives formulation for the vertex coloring problem. *Discrete Applied Mathematics*, Elsevier, v. 156, n. 7, p. 1097–1111, 2008.

60 CORNAZ, D.; JOST, V. A one-to-one correspondence between colorings and stable sets. *Operations Research Letters*, Elsevier, v. 36, n. 6, p. 673–676, 2008.

61 CAMPOS, V.; CARMO, R.; NOGUEIRA, R. Instances for the maximum clique problem with hardness guarantees. In: *Anais do VII Encontro de Teoria da Computação*. Porto Alegre, RS, Brasil: SBC, 2022. p. 125–128. ISSN 2595-6116.

62 JOHANSSON, A.; KAHN, J.; VU, V. Factors in random graphs. *Random Structures & Algorithms*, v. 33, n. 1, p. 1–28, 2008.

63 BLÄSIUS, T.; KATZMANN, M.; STEGEHUIS, C. Maximal cliques in scale-free random graphs. Preprint available in <https://arxiv.org/abs/2309.02990>. 2023.

64 HABIB, M.; PAUL, C. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, v. 4, n. 1, p. 41–59, 2010. ISSN 1574-0137.

65 TEDDER, M.; CORNEIL, D.; HABIB, M.; PAUL, C. Simpler linear-time modular decomposition via recursive factorizing permutations. In: ACETO, L.; DAMGÅRD, I.; GOLDBERG, L. A.; HALLDÓRSSON, M. M.; INGÓLFSDÓTTIR, A.; WALUKIEWICZ, I. (Ed.). *Automata, Languages and Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 634–645.

66 BOLLOBÁS, B. A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. *European Journal of Combinatorics*, v. 1, n. 4, p. 311–316, 1980. ISSN 0195-6698.

67 WORMALD, N. C. Models of random regular graphs. In: ____. *Surveys in Combinatorics, 1999*. [S.l.]: Cambridge University Press, 1999. (London Mathematical Society Lecture Note Series), p. 239–298.

68 WORMALD, N. C. The asymptotic connectivity of labelled regular graphs. *Journal of Combinatorial Theory, Series B*, v. 31, n. 2, p. 156–167, 1981. ISSN 0095-8956.

69 GRÖTSCHEL, M.; LOVÁSZ, L.; SCHRIJVER, A. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, v. 1, n. 2, p. 169–197, Jun 1981. ISSN 1439-6912.

70 MEHROTRA, A.; TRICK, M. A. A column generation approach for graph coloring. *INFORMS Journal on Computing*, v. 8, p. 344–354, 1995.

71 GODSIL, C.; MEAGHER, K. *Erdős–Ko–Rado Theorems: Algebraic Approaches*. [S.l.]: Cambridge University Press, 2015. (Cambridge Studies in Advanced Mathematics).

72 BALAS, E.; XUE, J. Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring. *Algorithmica*, v. 15, n. 5, p. 397–412, May 1996. ISSN 1432-0541.

73 SCHEINERMAN, E.; ULLMAN, D. *Fractional Graph Theory: A Rational Approach to the Theory of Graphs*. [S.l.]: Dover Publications, 2011. (Dover books on mathematics). ISBN 9780486485935.

74 BEHNEL, S.; BRADSHAW, R.; CITRO, C.; DALCIN, L.; SELJEBOTN, D.; SMITH, K. Cython: The best of both worlds. *Computing in Science Engineering*, v. 13, n. 2, p. 31–39, 2011. ISSN 1521-9615.

75 HAGBERG, A. A.; SCHULT, D. A.; SWART, P. J. Exploring network structure, dynamics, and function using networkx. In: VAROQUAUX, G.; VAUGHT, T.; MILLMAN, J. (Ed.). *Proceedings of the 7th Python in Science Conference*. Pasadena, CA USA: [s.n.], 2008. p. 11–15.

76 CARMO, R.; ZÜGE, A. Branch and bound algorithms for the maximum clique problem under a unified framework. *Journal of the Brazilian Computer Society*, Springer, v. 18, n. 2, p. 137–151, 2012.

77 ZÜGE, A.; CARMO, R.; ANJOS, C. S.; CORRÊA, M. V. *MAXCLIQUEBB repository*. 2017. Available in <https://gitlab.c3sl.ufpr.br/apzuge/maxcliquebb>.