Spring 3-14-2012

Combinatorics Using Computational Methods

Derrick Stolee
University of Nebraska-Lincoln, s-dstolee1@math.unl.edu

Stolee, Derrick, "Combinatorics Using Computational Methods" (2012). *Dissertations, Theses, and Student Research Papers in Mathematics*. 30.
http://digitalcommons.unl.edu/mathstudent/30

COMBINATORICS USING COMPUTATIONAL METHODS

by

Derrick Stolee

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Doctor of Philosophy

Major: Mathematics and Computer Science

Under the Supervision of Professor Stephen G. Hartke and Professor N. V. Vinodchandran

Lincoln, Nebraska

May, 2012

COMBINATORICS USING COMPUTATIONAL METHODS

Derrick Stolee, Ph. D.

University of Nebraska, 2012

Advisers: S. G. Hartke and N. V. Vinodchandran

Computational combinatorics involves combining pure mathematics, algorithms, and computational resources to solve problems in pure combinatorics. This thesis provides a theoretical framework for combinatorial search, which is then applied to several problems in combinatorics.

Chain Counting: Linek asked which numbers can be represented as the number of chains in a width-two poset. By developing a method for counting chains in posets generated from small configurations, constructions are found to represent every number from five to 50 million, providing strong evidence that all numbers are representable.

Ramsey Theory on the Integers: Van der Waerden's Theorem states that for sufficiently large n the numbers $1, 2, \dots, n$ cannot be r-colored while avoiding monochromatic arithmetic progressions. Finding the minimum n with this property is an incredibly difficult problem. We develop methods to compute the minimum n as well as optimal colorings when trying to avoid two generalizations of arithmetic progressions.

p-Extremal Graphs: For an integer $p \geq 1$, a p-extremal graph is a graph with the maximum number of edges over all graphs of order n with p perfect matchings. We describe the structure of p-extremal graphs in terms of a finite number of fundamental graphs and then discover these fundamental graphs using a computational search.

Uniquely K_r-Saturated Graphs: A graph G is uniquely K_r-saturated if G contains no copy of K_r, but adding any missing edge to G creates exactly one copy of K_r as a subgraph. Very little was known about uniquely K_r-saturated graphs, but by adapting a technique from combinatorial optimization we found several new examples of these graphs. One of these graphs led to the discovery of two new infinite families of uniquely K_r-saturated graphs.

Some results in space-bounded computational complexity are also presented. First, two nondeterministic complexity classes defined by the number and structure of computation paths are shown to be equal. Second, a log-space algorithm is developed to solve reachability problems on directed graphs that are embedded in surfaces of low genus.

DEDICATION

To Katie, for everything.

Acknowledgements

They say it takes a village to raise a child. Imagine how many people it takes to train a Ph.D.! Over the past five years, I have benefited from the solid support of two departments, members of the research community, and my family and friends. I want to thank you all for helping me arrive here at this milestone along my academic journey.

First and foremost, I thank my amazing wife, Katie, for her constant love and support. It certainly was not easy to put up with me while I was stressed about finishing, obsessed over a proof[1], or constantly checked computations. Katie took all this in stride, feeding me when I forgot to eat, pulling me away from work when I was frustrated, and convincing me that it would all be OK in the end. I can only hope that I can reciprocate this support as she tackles her own Ph.D. thesis in the coming year.

My most sincere thanks go to my advisors Stephen and Vinod, for taking me in as an over-eager graduate student and putting up with me for the past few years. You have given me more opportunities than I can count and for that I am very grateful.

Thanks to my committee members Jamie Radcliffe, Stephen Scott, and Christina Falci. Jamie, thanks for always being available to answer a question whenever

---

[1] For example, the months of October, November, and December of 2011 were spent obsessing over the proof of Theorem 11.3, which fills 33 pages of this thesis.

I stopped by. It's a shame we didn't have the chance to work together more. Stephen (Scott), while you may not be aware of it, your description of the forward/backward algorithm for hidden Markov models during your Pattern Recognition course inspired much of the language used for the tabulation method in Chapter 5. Christina, I hope your experience on my committee has given you a nice view of what pure graph theory is all about.

Steve Goddard and Judy Walker both had significant impacts on my undergraduate career, helped guide me towards graduate school, and mentored me through my Ph.D. experience[2]. Steve, thanks for giving me my first research job, which got me hooked even though I had no idea what I was doing. I'll never forget how you patiently[3] mentored me for my undergraduate thesis, and how that changed my approach to completing this thesis. Judy, your combinatorics course gave me the first taste of what "real" math was like and convinced me to pursue graduate school in mathematics.

I also thank the faculty and staff at the Holland Computing Center, especially David Swanson, Brian Bockleman, and Derek Weitzel for their extremely helpful advice during the design, development, and execution of my software. Thanks to the Holland Computing Center and the Open Science Grid for providing access to significant computational resources, without which some of the results of this thesis would have less impact[4]. Further, I apologize for that time I crashed the Prairiefire supercomputer so badly the machines needed to be hard-rebooted[5].

Thanks to Douglas B. West for allowing me to participate in the Combinatorics Research Experience for Graduate Students (REGS) at the University of Illinois at

---

[2] I am also quite proud of being witness to both Steve and Judy's transitions from humble associate professors to full professors and chairs of their departments.

[3] I believe his first words after my defense were "I'm surprised you got this done."

[4] If I have seen further it is by standing on the shoulders of giant robots.

[5] At least you have my executable available to test against your safeguards.

Urbana-Champaign. The contents of Chapters 4 and 9 originated as collaborations at REGS. Further, our collaboration provided me with a critical awareness of quality writing and that my writing leaves much to be desired.

To Michael Ferrara, who not only provides good problems but provides excellent company[6]. Thanks for inviting me to Denver to work on some truly interesting problems. I was flattered that you asked for my computational expertise towards your "white whale" problem, and I was not surprised that you solved it without my help.

To Paul S. Wenger, whose interests overlap with mine in too many ways to count[7]. Thanks for letting me stay with you whenever I'm in town, and know that you're always welcome at my place. Also, thank you for telling me about unique saturation, where after months of work we know a lot more about the problem, but mostly we now know the problem is even more complicated than previously thought (see Chapter 11).

Thanks to Eric Allender, David Mix Barrington, and Lance Fortnow, three leaders of computational complexity, for treating a lowly, unproven graduate student as a colleague and friend. Sometimes, theoretical computer science can feel like a cutthroat and competetive research area, but you made me feel welcome. I plan to emulate your attitude towards young researchers in the future.

Thanks to my good friend, Joe Geisbauer, for always being available to listen to my problems, provide advice, and facilitate distraction (when appropriate). If anyone asks how to succeed in graduate school while maintaining sanity and focusing on living life in the present-tense, I don't have the answers but recommend

---

[6] Somehow, whenever Mike visited Lincoln, we ended up having a party at my house. Of course the rule was "do math first, eat wings later."

[7] If Paul and I are in the same town, we find a way to (1) go on a run, (2) do some math, and (3) go out for a beer (and not necessarily in that order).

they talk to Joe, who has the right idea. I must voice my enthusiasm for Joe's most recent work in applied duck shield research[8].

Almost every UNL math graduate student who attended from 2007 to 2011 should probably thank Zahava Wilstein, and I am no exception. In addition to being a very fun officemate, Zahava was a social catalyst in the department: hosting grad student parties, pursuing interactions with the quietest of grad students, and being an all-around pleasant person.

To my officemate and academic younger brother, James Carraher, whose calm and quiet demeanor reminds me that you don't need to be outspoken to have an impact. James is the kind of person that people pursue for help and advice, because he has all the answers, all the patience, and all the humility.

Thanks to all my comrades-at-arms (fellow Mathematics and Computer Science graduate students) that have made the graduate experience much more pleasant than it could have been.

Finally, thanks to all of my collaborators on projects finished and unfinished, written and unwritten, and published, submitted, in preparation, or to be finished another day: Pranav Anand, Chris Bourke, Jane Butterfield, Henry Escuadro, Brady Garvin, Ralucca Gera, Ellen Gethner, Adam S. Jobson, Travis Johnston, André Kézdy, Elizabeth Kupin, Timothy D. LeSaulnier, Jared Nishikawa, Kevin G. Milans, Andrew Ray, Ben Reiniger, Tyler Seacrest, Hannah (Kolb) Spinoza, Brendon Stanton, Raghunath Tewari, and Matthew Yancey.

---

[8] One of Joe's research problems involved the integral $\int_0^1 Du(x + she_i)ds$.

GRANT INFORMATION

This work was supported in part by the University of Nebraska Presidential Fellowship, the University of Nebraska Othmer Fellowship, a Nebraska EPSCoR First Award, and National Science Foundation grants DMS-0354008, DMS-0914815, and CCF-0916525.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 0

Introduction

Computational Combinatorics involves blending pure mathematics, algorithms, and computational resources to solve problems in in pure combinatorics by finding examples and counterexamples, discovering conjectures, and proving theorems. My main research goal is to fully investigate new interactions between the theoretical side of pure mathematics and the practical side of algorithms and computation in order to push the limits of knowledge in combinatorics.

The development and proliferation of computational methods will accelerate the field of combinatorics. This is because using computational methods at every step of the research process can help researchers quickly gain intuition on a problem. Further, thinking about problems algorithmically can lead to new theoretical developments. This can take the form of conjectures that are found experimentally and proven mathematically (for example, see Theorems 11.2 and 11.3 in Chapter 11). More interestingly, an algorithmic perspective creates new problems whose solutions may reveal something new and interesting for the original problem (for example, see Theorem 9.65 and Lemma 9.74 in Chapter 9).

Previous computational efforts in combinatorics focused on generating objects that appear frequently; the goal was to enumerate and examine many examples. I extend the current computational techniques to be more effective in the case of rare objects. These techniques address the fact that most combinatorial objects are unlabeled, but all computer representations are labeled. The techniques either attempt to reduce duplication of an isomorphism class or remove duplicates altogether. Knowing which technique to use for a given problem requires experience and experimentation. A black-box approach rarely suffices, so I further develop each technique for the current problem. Using the algorithmic perspective, I prove structural and extremal theorems of pure combinatorics which allows the algorithm to examine fewer examples.

Combinatorial problems demonstrate some of the most general (and most difficult) types of symmetry. By developing computational methods to handle the case of combinatorial objects, the techniques may also be effective in practical optimization problems where symmetry is present.

One part of this thesis presents results from space-bounded computational complexity. Space-bounded complexity has some common features with computational combinatorics. When restricting to logarithmic space, the entire memory can only contain a few pointers to vertices or edges of a larger graph. Thinking of combinatorial search as a walk on a large graph (where the vertices are combinatorial objects and edges correspond to augmentations), we see a connection. The similarity ends there, since space-bounded complexity is not concerned with time efficiency and algorithms frequently iterate over every possible vertex in order to perform simple operations. The two main results in this thesis have very different goals but have the common feature that the proofs greatly depend on techniques from pure graph theory.

Contents of this Thesis

This thesis is split into four parts:

Part 1: Fundamentals of Combinatorial Search.

Part 2: Isomorph-Free Generation.

Part 3: Orbital Branching.

Part 4: Reachability Problems in Space-Bounded Complexity.

The first three parts deal with computational combinatorics. Part 1 includes basic descriptions of graph theory, automorphisms of graphs, and a high-level description of combinatorial search. This part is finished by two chapters which describe two different combinatorial problems and the computational approach to solve them.

These first two problems largely avoid the issue of isomorphism among combinatorial objects. Parts 2 and 3 discuss two techniques to deal with combinatorial objects with large numbers of isomorphic duplicates.

Part 2 concerns isomorph-free generation, a technique to remove all but one representative of an isomorphism class. This technique is then extended to a specific case of building graphs by ear augmentations. These ear augmentations are used for two problems: verifying the Edge Reconstruction conjecture on 2-connected graphs and generating p-extremal graphs. The latter problem requires a significant portion of pure graph theory in order to show that a finite computation can solve the problem and then even more theory is developed to make the algorithm efficient.

Part 3 describes orbital branching, a generalization of the branch-and-bound technique from combinatorial optimization. This technique is then customized to tackle a relatively new problem in structural graph theory, resulting in several new graphs of a given type including two new infinite families.

Finally, Part 4 is an investigation into space-bounded computational complexity theory. After a short introduction to the area, two very different results are presented. One involves showing two complexity classes are equal. The other finds a new algorithm to solve reachability on a larger class of planar graphs than previously known.

A few appendices are included to further expand some details which support the main narrative. Appendix A contains a description of the symbols used in the algorithms of this work. Appendices B, C, D, and E document the software packages which were created as part of this work.

These parts and their included chapters are now described in further detail.

Part 1: Fundamentals of Combinatorial Search

Chapter 1 contains the basic definitions and notation from graph theory that I will use during the rest of the work.

In Chapter 2, I discuss a crucial issue of isomorphism of graphs and their automorphism groups. This includes a brief survey of some results dealing with automorphism groups of graphs as well as some recent results. Section 2.2 is based on joint work with Stephen G. Hartke, Hannah (Kolb) Spinoza, and Jared Nishikawa [60], while Theorem 2.21 is from [125].

Chapter 3 discusses the philosophy of combinatorial search. By using a toy example, I describe a mathematical framework that will be used for later computational experiments. I also introduce how the *TreeSearch* software library [122] abstracts the structure of a combinatorial search and allows for parallelism on a supercomputer.

Chapter 4 considers which numbers can be represented as the number of chains in a poset of width two. Instead of generating posets and counting the chains, I develop a method to generate posets by adding points to a small collection of *configurations* and from these configurations create formulas for counting the number of chains in the resulting posets. By evaluating these formulas on many inputs, I find that every number up to 50 million can be represented, providing significant evidence that every number is representable. This chapter is based on joint work with Elizabeth Kupin and Ben Reiniger [76].

Chapter 5 investigates Ramsey Theory on the integers, where the numbers from 1 to n are colored while attempting to avoid certain monochromatic patterns. The most famous type of pattern is an *arithmetic progression*, which is the subject of both van der Waerden's Theorem [140] and Szemerédi's Theorem [130, 131]. Arithmetic progressions are generalized in two different ways, to quasi-arithmetic progressions and pseudo-arithmetic progressions. Using a computational approach using constraint propagation, I extend the known bounds on extremal colorings, including some exact values. New theorems and conjectures result from the data. This is joint work with Adam Jobson and André Kézdy [71].

Part 2: Isomorph-Free Generation

Chapter 6 is an original description of a computational technique developed by Brendan McKay [92]. This technique guarantees exactly one representative of every isomorphism class is visited during a combinatorial search. The remaining chapters of this part customize this technique for the given problems.

Chapters 7 extends McKay's technique to work when the augmentation step is adding an *ear*: adding a path by attaching the endpoints to vertices in the current graph. This leads to a natural way to generate 2-connected graphs (the graphs which can be built from ear augmentations). Chapter 8 exploits this technique to verify the Edge Reconstruction conjecture on 2-connected graphs. These chapters are based on [124].

In Chapter 9, I investigate an extremal graph theory problem: which graphs have the maximum number of edges when the number of vertices and number of perfect matchings is fixed? Intuitively, more perfect matchings imply more edges are possible. By describing the structure of this infinite family of extremal graphs, I reduce the problem to a finite search to a set of fundamental graphs which are then combined to create the infinite family. The structure theorems allow the ear augmentation method to be exploited both theoretically and practically to greatly extend the knowledge on this problem. This chapter is based on joint work with Stephen G. Hartke, Douglas B. West, and Matthew Yancey [63] and [123].

Part 3: Orbital Branching

Chapter 10 is an original description of a computational technique developed by James Ostrowski, Jeff Linderoth, Fabrizio Rossi, and Stefano Smriglio [102]. This technique extends the branch-and-bound technique from combinatorial optimization. Orbital branching works to reduce the number of isomorphic duplicates, but does not remove them entirely. Instead, it utilizes the symmetries of partial solutions during the execution to place value on more than one variable at a time. This technique differs mostly from isomorph-free generation in that it immediately cooperates with constraint propagation.

In Chapter 11 I investigate uniquely K_r-saturated graphs. Unique saturation was recently defined by Cooper, Lenz, LeSaulnier, Wenger, and West [34] and then studied in great detail for the case of uniquely C_k-saturated graphs. By extending the technique of orbital branching to be more effective searching for uniquely K_r-saturated graphs, several new graphs are discovered. By investigating these graphs, a new algebraic construction is developed and used to find two new infinite families. This chapter is based on joint work with Stephen G. Hartke [62].

Part 4: Reachability Problems in Space-Bounded Complexity

This fourth part of the thesis has a very different flavor than the rest of the work. Part 4 contains my contributions to the area of computational complexity, specifically that in space-bounded complexity. Most space-bounded complexity problems reduce to a reachability problem (or path-finding problem) on a certain class of graphs.

Chapter 12 describes space-bounded complexity starting with the definition of Turing machines and space-bounded complexity classes. The chapter finishes with the major theorems of space-bounded complexity.

Chapter 13 is based on joint work with Brady Garvin, Raghunath Tewari, and N. V. Vinodchandran [48], where we collapse two complexity classes. These classes are defined by the type of graphs where reachability can be solved. By carefully combining reductions, hashing results, and oracle queries, we prove equality between the classes.

Chapter 14 is based on joint work with N. V. Vinodchandran [127], where I significantly increase the class of planar graphs that have deterministic log-space algorithms for reachability. This extends the previous-best result which is joint with Chris Bourke and N. V. Vinodchandran [126].

Techniques Used

This thesis encompasses the majority of my research productivity over the past few years. During this time, I learned several mathematical techniques. Below is a collection of these techniques and how they are used in this work.

1. *Computational.* This is the main technique. Chapters 4, 5, 8, 9, and 11 all contain results whose proofs use computation. Several other theorems use computation indirectly, as I use experimental computation during early stages of research to gain intuition for a problem.
2. *Structural.* Knowledge about the structure of a graph can be very useful when building a computational method. In Chapter 9, I begin by developing structural theorems about p-extremal graphs and then use that structure to develop a computational method.
3. *Extremal.* In Chapter 5, I prove exact values of an extremal function. In Chapter 9, I use the previously mentioned structural theorems to prove an extremal theorem which is used to significantly speed up the computational method.
4. *Probabilistic.* The probabilistic method uses random experiments to prove existence of a combinatorial object. I use the Lovász Local Lemma in Chapter 5 to prove a lower bound on an extremal function.

5. *Discharging.* This method involves defining a charge function and then passing around charge among elements while preserving the total charge sum. I use a two-stage discharging method to prove Theorem 11.3 in Chapter 11. This method provides a clean way to compute the clique number of a certain graph family.

6. *Face-Melting Case Analysis.* The previously mentioned discharging proof has a clean proof when the graph preserves its symmetry. Adding an edge removes this symmetry, and counting the number of maximum cliques in this new graph reduces to a long and detailed case analysis.

7. *Reductions.* In Chapter 13, I use reductions to prove that two complexity classes are equal. In Chapter 14, I use log-space reductions in a novel way (I *compress* the input) in order to lower the required resources for solving the problem.

List of Papers

Below is a list of the papers which share content with this thesis.

[48] B. Garvin, D. Stolee, R. Tewari, and N. V. Vinodchandran. ReachUL = ReachFewL. *17th Annual International Computing and Combinatorics Conference*, 2011.

[60] S. G. Hartke, H. Kolb, J. Nishikawa, and D. Stolee. Automorphism groups of a graph and a vertex-deleted subgraph. *Electron. J. Combin.*, 17(1):Research Paper 134, 8, 2010.

[62] S. G. Hartke and D. Stolee. Uniquely K_r-saturated graphs, 2012. preprint.

[63] S. G. Hartke, D. Stolee, D. B. West, and M. Yancey. On extremal graphs with a given number of perfect matchings, 2011, preprint.

[71] A. Jobson, A. Kézdy, and D. Stolee. A new variant of van der Waerden numbers, 2012. in preparation.

[76] E. Kupin, B. Reiniger, and D. Stolee. Counting chains in width-two posets with few cover edges, 2012. in preparation.

[123] D. Stolee. Generating p-extremal graphs, 2011. preprint.

[124] D. Stolee. Isomorph-free generation of 2-connected graphs with applications. Technical Report #120, *University of NebraskaÐLincoln, Computer Science and Engineering*, 2011.

[125] D. Stolee. Automorphism groups and adversarial vertex deletions, 2012. preprint.

[127] D. Stolee and N. V. Vinodchandran. Space-efficient algorithms for reachability in surface-embedded graphs. *27th Annual IEEE Conference on Computational Complexity*, 2012. to appear.

Part I

Fundamentals of Combinatorial Search

Chapter 1

Graph Theory

The main combinatorial object of this thesis is a *graph*. This chapter introduces the basic definitions and major theorems regarding some of the aspects of graph theory that will be used in later chapters. The major definitions and results can also be found in standard texts such as Bollobás [16, 18], Diestel [37], or West [146].

Definition 1.1. A graph G is a pair (V, E), where V is a set of *vertices* and E is a set of *edges*. When the pairs V, E are not specified in advance, the vertices of G are denoted $V(G)$ and the edges $E(G)$. The sizes of these sets are $n(G) = |V(G)|$ and $e(G) = |E(G)|$.

Typically, E is a subset of unordered pairs of V. In such a case, G is *simple* and *undirected*. A *directed graph* is a graph G where the edges E are a subset of ordered pairs.

There is a natural form of isomorphism between graphs.

Definition 1.2. Two graphs H and G are *isomorphic*, denoted $H \cong G$, if there is a bijection $\pi : V(H) \to V(G)$ so that for all pairs $u, v \in V(H)$ there is an edge $uv \in E(H)$ if and only if $\pi(u)\pi(v) \in E(G)$. Such a map π is called an *isomorphism*.

1.1 Substructures

Definition 1.3. For two graphs H and G, H is a *subgraph* of G, denoted $H \subseteq G$, if there is an injection $\pi : V(H) \to V(G)$ so that for all edges $uv \in E(H)$ there is an edge $\pi(u)\pi(v) \in E(G)$.

Definition 1.4. Let G be a graph and S a subset of $V(G)$. The subgraph $G[S]$ *induced by* S is the graph on vertex set S with an edge between vertices $u, v \in S$ if and only if $uv \in E(G)$. A graph H is an *induced subgraph* of G if there exists a set $S \subseteq V(G)$ so that $H \cong G[S]$.

Definition 1.5. A set of vertices $S \subseteq V(G)$ is *independent* if there are no edges between any two vertices in S. The maximum size of an independent set in G is denoted $\alpha(G)$.

Definition 1.6. A set of vertices $S \subseteq V(G)$ is a *clique* if there is an edge between every pair of vertices in S. The maximum size of a clique in G is denoted $\omega(G)$.

Essentially, an independent set of size r is an induced subgraph of $\overline{K_r}$ while a clique of size r is a copy of K_r as a subgraph[1].

1.2 Connectivity

Definition 1.7. A graph is *connected* if every pair $u, v \in V(G)$ admits a path between u and v. G is *disconnected* otherwise.

Definition 1.8. For an integer $k \geq 1$, G is *k-connected* when every subset $S \subseteq V(G)$ with $|S| < k \leq n(G) - 1$ has $G - S$ connected.

---

[1] Observe that a K_r subgraph is also an induced subgraph.

1.3 Matching Theory

Definition 1.9. A set of edges $M \subseteq E(G)$ is a *matching* if no two edges in M share an endpoint. The largest matching size is denoted $\alpha'(G)$. Since every edge requires two vertices, $\alpha'(G) \leq \lfloor \frac{n(G)}{2} \rfloor$.

Definition 1.10. If $n(G)$ is even, a matching M of size $n(G)/2$ is *perfect* since every vertex in G is incident to exactly one edge in M. If G has a perfect matching, then G is *matchable*. Let $\Phi(G)$ denote the number of perfect matchings in G.

The two most well-studied characterizations of matchable graphs are Hall's Theorem for bipartite graphs and Tutte's Theorem for general graphs.

Theorem 1.11 (Hall [58]). *A bipartite graph G with bipartition $V(G) = X \cup Y$ has a matching that saturates X if and only if for all $S \subseteq X$, $|S| \leq |N(S)|$.*

Theorem 1.12 (Tutte [138]). *A graph G is matchable if and only if for all sets $S \subseteq V(G)$, the number of odd components in $G - S$ is at most $|S|$.*

1.4 Extremal Graph Theory

Definition 1.13. Fix a graph H. A graph G is *H-saturated* if G does not contain H as a subgraph and for every nonedge $e \in E(\overline{G})$, $G + e$ contains at least one copy of H.

One of the most fundamental theorems in extremal graph theory is Turán's theorem.

Theorem 1.14 (Turán [137]). *Fix $r \geq 3$. The maximum number of edges in an n-vertex K_r-saturated graph is $\left(1 - \frac{1}{r-1}\right) \frac{n^2}{2}(1 - o(1))$.*

When considering the maximum number of edges in an H-saturated graph for an arbitrary non-bipartite H, what really matters is the chromatic number of H.

Theorem 1.15 (Erdős, Stone, Simonovits [42, 44])**.** *Fix H with $\chi(H) \geq 3$. The maximum number of edges in an n-vertex H-saturated graph is $\left(1 - \frac{1}{\chi(H)-1}\right)\frac{n^2}{2}(1 - o(1))$.*

This extremal question can be reversed by asking what is the *minimum* number of edges required to be H-saturated.

Theorem 1.16 (Erdős, Hajnal, Moon [43])**.** *Fix $r \geq 3$. The minimum number of edges in an n-vertex K_r-saturated graph is $\binom{r-2}{2} + (r-2)(n-r+2)$.*

Chapter 2

Automorphisms

In this chapter, we discuss the automorphisms of graphs.

Definition 2.1. Given a graph G, a bijection $\pi : V(G) \to V(G)$ is an *automorphism* if for every pair $u, v \in V(G)$, the pair uv is an edge in $E(G)$ if and only if the pair $\pi(u)\pi(v)$ is an edge of $E(G)$. The set of automorphisms of G forms a group under composition, denoted $\mathrm{Aut}(G)$.

We shall review some theorems about automorphisms of graphs. The reader should discover a feeling that the symmetries of graphs are very fragile and difficult to completely understand.

2.1 Fundamental Theorems for Automorphism Groups of Graphs

A graph is called *rigid* if it has a trivial automorphism group.

Theorem 2.2 (See Bollobás [17]). *Let* $G \sim G_{n,p}$ *for* $p = \frac{1}{2}$ *and let* $\varepsilon > 0$. *The graph* G *is rigid with probability*

$$\Pr[\operatorname{Aut}(G) \cong I] \geq 1 - 2ne^{-(n-1)p\varepsilon^2/2} - n^2 2^{1-(n-1)p(1-\varepsilon)},$$

which tends to 1 *as* n *tends to infinity.*

While this seems to imply that very few graphs have any symmetry at all, we can actually encode any type of symmetry into a graph.

Definition 2.3 (Frucht [47]). Given a group Γ generated by elements $S = \{\sigma_i\}_{i \in I}$, the *Cayley graph* $C(\Gamma, S) = (\Gamma, E)$ is the edge-labeled directed graph with vertex set Γ and an edge $x \to y$ with label σ if $\sigma \in S$ and $y = \sigma x$.

Theorem 2.4 (Sabidussi [115]). *Let* Γ *be a finite group generated by* S *with* $n = |\Gamma|$. *The Cayley graph* $C(\Gamma, S)$ *has automorphism group* Γ. *The labeled edges can be replaced with simple undirected gadgets of order* $\log |S|$ *to form a graph* $C'(\Gamma, S)$ *of order* $O(|\Gamma| \log |S|)$ *with automorphism group isomorphic to* Γ.

For a while, this stood as the best upper bound on the size of an undirected graph with given automorphism group. Then, Sabidussi presented in 1958 a complete characterization of the minimum-order graphs with a k-order cyclic automorphism group for each $k \geq 2$.

Definition 2.5. Let Γ be a finite group. We define the minimum graph order $\alpha(\Gamma)$ to be

$$\alpha(\Gamma) = \min\{n(G) : G = (V, E), \operatorname{Aut}(G) \cong \Gamma\},$$

the minimum order of a simple graph with automorphism group isomorphic to Γ.

Lemma 2.6 (Sabidussi [114]). *Let $m \geq 2$ be an integer.*

$$\alpha(\mathbb{Z}_m) = \begin{cases} 2 & \text{if } m = 2, \\ 3m & \text{if } m \in \{3,4,5\}, \\ 2m & \text{if } m = p^3 \geq 7, p \text{ prime}, \\ \sum_{i=1}^{t} \alpha(\mathbb{Z}_{p_i^{e_i}}) & \text{where } m = \prod_{i=1}^{t} p_i^{e_i} \text{ for } p_1, \dots, p_t \text{ distinct primes.} \end{cases}$$

It was not until 1974 when László Babai proved that those three cyclic groups were the only finite groups that required three vertices per element. All other finite groups with n elements are representable by a graph of order $2n$.

Theorem 2.7 (Babai [9]). *If Γ is a finite group not isomorphic to $\mathbb{Z}_3, \mathbb{Z}_4$, or $\mathbb{Z}_5$, then there exists a graph G with $\operatorname{Aut}(G) \cong \Gamma$ and $|V(G)| \leq 2|\Gamma|$.*

Proof. If Γ is cyclic, we are done by Sabidussi's theorem.

If $\Gamma \cong V_4$, we have $V_4 \cong \operatorname{Aut}(K_4 - e)$.

Now, assume $|\Gamma| > 6$. Let $S = \{\alpha_1, \dots, \alpha_t\}$ be a minimal generating set of Γ. Create two graphs $G_1 = (\Gamma, E_1), G_2 = (\Gamma, E_2)$.

In G_1, for each element $\gamma \in \Gamma$ and each $i \in \{1, \dots, t-1\}$, place an edge between $\alpha_i\gamma$ and $\alpha_{i+1}\gamma$. Note that each vertex set $\{\alpha_1\gamma, \dots, \alpha_t\gamma\}$ is a path in G_1. If there exists an edge between $\alpha_i\gamma$ and $\alpha_j\gamma$ with $j > i+1$, this contradicts minimality of S, since there exists $\gamma' \in \Gamma, \ell \in \{1, \dots, t-1\}$ so that

$$\alpha_i\gamma = \alpha_\ell\gamma', \quad \alpha_j\gamma = \alpha_{\ell+1}\gamma'.$$

This gives $\gamma' = \alpha_{\ell+1}^{-1}\alpha_j\gamma$ and hence $\alpha_i = \alpha_\ell\alpha_{\ell+1}^{-1}\alpha_j$.

In G_2, for each element $\gamma \in \Gamma$, place an edge between γ and $\alpha_1\gamma$.

Both G_s ($s \in \{1,2\}$) are regular with degree d_s. We have $d_2 = 2$. If $d_1 = d_2$, then these graphs have the same degree.

Define G_3 by case: if $d_1 \neq d_2$, then $G_3 = G_2$; if $d_1 = d_2$, then $G_3 = \overline{G_2}$. Note that G_3 is regular with degree $d_3 \neq d_1$, since if $d_1 = d_2$, then $d_3 = n - 1 - d_2 = n - 3 > 6 - 3 = 4 > d_2 = d_1$.

Define $G = (\Gamma \times \{1,3\}, E)$ where $E = E_1' \cup (E_3 \times \{3\}) \cup E'$, where

$$\begin{aligned} E_s' &= \{\{(\gamma, s), (\delta, s)\} : \{\gamma, \delta\} \in E_s\}, \\ E' &= \{\{(\gamma, 1), (\gamma, 3)\} : \gamma \in \Gamma\} \\ &\qquad \cup \{\{(\gamma, 3), (\alpha_i\gamma, 1)\} : \gamma \in \Gamma, i \in \{1, \dots, t\}\}. \end{aligned}$$

Claim 2.8. $\mathrm{Aut}(G) \cong \Gamma$.

First, note that Γ is isomorphic to a subgroup of $\mathrm{Aut}(G)$. Given $\delta \in \Gamma$, $\pi_\delta : V(G) \to V(G)$ is defined as

$$\pi_\delta(\gamma, s) = (\gamma\delta, s) \qquad\qquad \forall \gamma \in \Gamma, s \in \{1,3\}$$

Note that π_δ defines a bijection on each edge set E_1', E_3', E' as

$$\begin{aligned} \{(\alpha_i\gamma, 1), (\alpha_{i+1}\gamma, 1)\} &\xmapsto{\pi_\delta} \{(\alpha_i\gamma\delta, 1), (\alpha_{i+1}\gamma\delta, 1)\} && (E_1') \\ \{(\gamma, 3), (\alpha_1\gamma, 3)\} &\xmapsto{\pi_\delta} \{(\gamma\delta, 3), (\alpha_1\gamma\delta, 3)\} && (E_3' \text{ or } \overline{E_3'}) \\ \{(\gamma, 1), (\gamma, 3)\} &\xmapsto{\pi_\delta} \{(\gamma\delta, 1), (\gamma\delta, 3)\} && (E') \\ \{(\gamma, 3), (\alpha_i\gamma, 1)\} &\xmapsto{\pi_\delta} \{(\gamma\delta, 3), (\alpha_i\gamma\delta, 1)\} && (E') \end{aligned}$$

It remains to show any permutation in $\text{Aut}(G)$ is represented by π_δ for some $\delta \in \Gamma$.

Let $\gamma \in \Gamma$ be any element. Define the subgraph A_γ be the induced subgraph of G given by $(\gamma, 3), (\gamma, 1), (\alpha_1\gamma, 1), \ldots, (\alpha_t\gamma, 1)$. As mentioned previously, the vertices $(\alpha_1\gamma, 1), \ldots, (\alpha_t\gamma, 1)$ induce a path in G. It is also true that there is no edge from $(\gamma, 1)$ to $(\alpha_i\gamma, 1)$ for any $i \in \{1, \ldots, t\}$. If such an i existed, then there exists an $\ell \in \{1, \ldots, t-1\}$ and $\gamma' \in \Gamma$ ($\gamma' \neq \gamma$) so that

$$\gamma = \alpha_\ell \gamma', \quad \alpha_i\gamma = \alpha_{\ell+1}\gamma'.$$

However, this implies $\alpha_i = \alpha_{\ell+1}\alpha_\ell^{-1}$, which contradicts minimality of S.

Hence, $(\gamma, 1)$ is a leaf in A_γ.

Let $\pi \in \text{Aut}(G)$ be a permutation of $V(G)$. Consider an element $\gamma \in \Gamma$ and $\gamma' = \pi(\gamma)$. Since $\pi(A_\gamma) = A_{\gamma'}$, and $(\gamma, 1)$ is the only leaf in A_γ, $\pi(\gamma, 1) = \pi(\gamma', 1)$ since $(\gamma', 1)$ the only leaf in $A_{\gamma'}$.

So, π can be considered as a permutation of Γ that also acts on G. Let π be such a permutation given by a non-trivial automorphism of G.

Now, let γ be any element with $\pi(\gamma) \neq \gamma$ and define $\delta = \gamma^{-1}\pi(\gamma)$.

Claim 2.9. *For any element $\gamma' \in \Gamma$, $\pi(\gamma') = \gamma'\delta$.*

It is sufficient to prove that if $\pi(\gamma) = \gamma\delta$, then for all $i \in \{1, \ldots, t\}$ has $\pi(\alpha_i\gamma) = \alpha_i\gamma\delta$. If this is true, then for all $\gamma' \in \Gamma$, the sequence of generators $\alpha_{j_1} \cdots \alpha_{j_k} = \gamma'\gamma^{-1}$ gives $\gamma' = \alpha_{j_1} \cdots \alpha_{j_k}\gamma$ and iteration on the number of generators in the right-hand-side product gives $\pi(\gamma') = \gamma'\delta$.

Since the only vertex $(\alpha_i\gamma, 1)$ in A_γ that has $(\alpha_i\gamma, 3)$ adjacent to $(\gamma, 3)$ is $(\alpha_1\gamma, 1)$. Hence, $\pi(\alpha_1\gamma) = \gamma\delta$. Moreover, the path $(\alpha_1\gamma, 1)(\alpha_2\gamma, 1)\ldots(\alpha_t\gamma, 1)$ in A_γ is now embedded uniquely into $\pi(A_\gamma) = A_{\gamma\delta}$ as $(\alpha_1\gamma\delta, 1)(\alpha_2\gamma\delta, 1)\ldots(\alpha_t\gamma\delta, 1)$. This proves the claim. $\square$

Based on this construction of Babai, the worst-case order of a graph G with automorphism group Γ is $O(n)$ where $n = |\Gamma|$. Unfortunately, we cannot hope for better asymptotics than that (or much better constants, even), since there is a very close lower bound for the alternating group.

Theorem 2.10 (Liebeck [82]). *If $n \geq 23$, then the minimum order of a graph with automorphism group isomorphic to A_n is at least $\frac{1}{2}\binom{n}{\lfloor n/2 \rfloor}$.*

By Stirling's approximation, the above lower bound is approximately $\frac{2^n}{\sqrt{2\pi n}}$, while $|A_n| = \frac{n!}{2} = 2^{\theta(n \log n)}$.

Frequently, dropping the labels and directions from Cayley graphs provides a useful graph construction. It is important that this *unlabeled, undirected* Cayley graph $C(\Gamma, S)$ may have automorphism group larger than Γ.

A graph G is *vertex-transitive* if for any pair of vertices $u, v \in V(G)$ there is an automorphism $\sigma \in \operatorname{Aut}(G)$ so that $\sigma(u) = v$. These graphs are highly symmetric. While all Cayley graphs are vertex-transitive, the reverse is not always true. The following proposition gives a partial result to when a vertex-transitive graph can be guaranteed to be a Cayley graph.

Proposition 2.11 (Folklore). *Let p be a prime. A vertex-transitive graph of order p is isomorphic to the unlabeled, undirected Cayley graph $C(\mathbb{Z}_p, S)$ for some set $S \subseteq \mathbb{Z}_p$.*

Proof. Let G be a vertex-transitive graph with $n(G) = p$ a prime. Since G is vertex-transitive, the entire vertex set $V(G)$ is an orbit under the action of $\operatorname{Aut}(G)$. Since the length of an orbit is the index of the stabilizer, $p = |V(G)| = [\operatorname{Stab}_G(V(G)) : \operatorname{Aut}(G)]$, so p divides $|\operatorname{Aut}(G)|$. Thus, there is an automorphism $\sigma \in \operatorname{Aut}(G)$ of order p. Fix any vertex v of G and notice that $\sigma^{(i)}(v) = v$ if and only if p divides i. Therefore, we can label all vertices of G as $v_i = \sigma^{(i)}(v)$ for all $i \in \{0, \dots, p-1\}$. There is an edge between v_i and v_j if and only if there is an edge between v_0 and v_{j-i}. Therefore, let $S = \{i : v_0v_i \in E(G)\}$ and $G \cong C(\mathbb{Z}_p, S)$. □

2.2 Automorphism Groups of a Graph and a Vertex-Deleted Subgraph

The Reconstruction Conjecture of Ulam and Kelley famously states that the isomorphism class of all graphs on three or more vertices is determined by the isomorphism classes of its vertex-deleted subgraphs (see [55] for a survey of classic results on this problem). A frequent issue when attacking reconstruction problems is that automorphisms of the substructures lead to ambiguity when producing the larger structure.

This section considers the relation between the automorphism group of a graph and the automorphism groups of the vertex-deleted subgraphs and edge-deleted subgraphs. If a group Γ_1 is the automorphism group of a graph G, and another group Γ_2 is the automorphism group of $G - v$ for some vertex v, then we say Γ_1 *deletes to* Γ_2. This relation is denoted $\Gamma_1 \to \Gamma_2$. A corresponding definition for edge deletions is also developed. Our main result is that any two groups delete to each other, with vertices or edges.

These relations also appear in McKay's isomorph-free generation algorithm (see Chapter 6 and [92]), which is frequently used to enumerate all graph isomorphism classes. After generating a graph G of order n, graphs of order $n+1$ are created by adding vertices and considering each $G + v$. To prune the search tree, the canonical labeling of $G + v$ is computed, usually by `nauty`, McKay's canonical labeling algorithm [93, 61]. Finding a canonical labeling of a graph reveals its automorphism group. Since G was generated by this process, its automorphism group is known but is not used while computing the automorphism group of $G + v$. If some groups could not delete to the automorphism group of G, then they certainly cannot appear as the automorphism group of $G + v$ which may allow for some improvement to the canonical labeling algorithm. The current lack of such optimizations hints that no such restrictions exist, but this notion has not been formalized before this work.

One reason why this problem has not been answered is that the study of graph symmetry is very restricted, mostly to forms of symmetry requiring vertex transitivity. These forms of symmetry are useless in the study of the Reconstruction conjecture, as regular graphs are reconstructible. On the opposite end of the spectrum, almost all graphs are *rigid* (have trivial automorphism group) [17]. Graphs with non-trivial, but non-transitive, automorphisms have received less attention.

Graph reconstruction and automorphism concepts have been presented together before [10, 81]. However, there appears to be no results on which pairs of groups allow the deletion relation. While our result is perhaps unsurprising, it is not trivial. The reader is challenged to produce an example for $\mathbb{Z}_2 \to \mathbb{Z}_3$ before proceeding.

For notation, G always denotes a graph, while Γ refers to a group. The trivial group I consists of only the identity element, ε. All graphs in this chapter are finite, simple, and undirected, unless specified otherwise. All groups are finite. The automorphism group of G is denoted $\mathrm{Aut}(G)$ and the stabilizer of a vertex v in a graph G is denoted $\mathrm{Stab}_G(v)$.

2.2.1 Definitions and Basic Tools

We begin with a formal definition of the deletion relation.

Definition 2.12. Let Γ_1, Γ_2 be finite groups. If there exists a graph G with $|V(G)| \geq 3$ and vertex $v \in V(G)$ so that $\mathrm{Aut}(G) \cong \Gamma_1$ and $\mathrm{Aut}(G - v) \cong \Gamma_2$, then Γ_1 (*vertex*) *deletes to* Γ_2, denoted $\Gamma_1 \to \Gamma_2$. Similarly, the group Γ_1 *edge deletes to* Γ_2 if there exists a graph G and edge $e \in E(G)$ so that $\mathrm{Aut}(G) \cong \Gamma_1$ and $\mathrm{Aut}(G - e) \cong \Gamma_2$. If a specific graph G and subobject x give $\mathrm{Aut}(G) \cong \Gamma_1$ and $\mathrm{Aut}(G - x) \cong \Gamma_2$, the deletion relation may be presented as $\Gamma_1 \xrightarrow{G-x} \Gamma_2$.

To determine the automorphism structure of a graph, vertices that are not in the same orbit can be distinguished by means of neighboring structures. A useful gadget to make such distinctions is the rigid tree $T(n)$, where n is an integer at least 2. Build $T(n)$ by starting with a path $u_0, z_1, \dots, z_n$. For each $i, 1 \leq i \leq n$, add a path $z_i, x_{i,1}, x_{i,2}, \dots, x_{i,2i}, u_i$ of length $2i + 1$. This results in a tree with $n + 1$ leaves. Note that each leaf u_i is distance $2i + 1$ to a vertex of degree 3 (except for u_n, which is distance $2n + 2$). Thus, the leaves are in disjoint orbits and $T(n)$ is rigid. Also, if any leaf u_i is selected with $i \geq 1$, $T(n) - u_i$ is rigid. This gives an example of the deletion relation $I \to I$. For notation, let J be a set and $\{T_j\}_{j \in J}$ be disjoint copies of $T(n)$. Then $u_i(T_j)$ designates the copy of u_i in T_j. This is well-defined since there is a unique isomorphism between each T_j and $T(n)$.

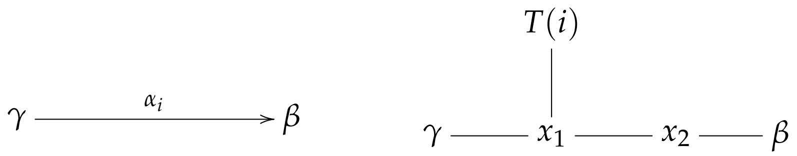
By Theorem 2.4, for every group Γ, there exists a simple, unlabeled, undirected graph G with $\mathrm{Aut}(G) \cong \Gamma$. The construction is derived from the well-known Cayley graph[1]. Define $C(\Gamma)$ to be a graph with vertex set Γ and complete directed edge set, where the edge (γ, β) is labeled with $\gamma^{-1}\beta$, the element whose right-

---

[1] In most uses of the Cayley graph, a generating set is specified. For simplicity, we use the entire group.

multiplication on γ results in β. The automorphism group of $C(\Gamma)$ is Γ, and the action on the vertices follows right multiplication by elements in Γ. That is, if $\gamma \in \Gamma$, the permutation σ_γ will take a vertex α to the vertex $\alpha\gamma$.

This directed graph with labeled edge sets is converted to an undirected and unlabeled graph by swapping the labeled edges with gadgets. Specifically, order the elements of $\Gamma = \{\alpha_1, \ldots, \alpha_n\}$ so that $\alpha_1 = \varepsilon$. For each edge (γ, β), subdivide the edge labeled $\alpha_i = \gamma^{-1}\beta$ with vertices x_1, x_2, and attach a copy $T_{\gamma,\beta}$ of $T(i)$ by identifying $u_0(T_{\gamma,\beta})$ with x_1. Note that $i \geq 2$ in these cases, since $\alpha_i \neq \varepsilon$. See Figure 2.1 for an example of this process.



A directed edge labeled α_i. An unlabeled undirected gadget.

Figure 2.1: Converting a labeled directed edge to an undirected unlabeled gadget.

Denote this modified graph $C'(\Gamma)$. We refer to it as *the* Cayley graph of Γ. Note that the automorphisms of $C'(\Gamma)$ are uniquely determined by the permutation of the group elements and preserve the original edge labels, since the trees $T(i)$ identify the label α_i and have a unique isomorphism between copies. Hence, $\mathrm{Aut}(C'(\Gamma)) \cong \mathrm{Aut}(C(\Gamma)) \cong \Gamma$.

Lemma 2.13. *Let Γ be a group and $G = C'(\Gamma)$. Then the stabilizer of the identity element ε (as a vertex in G) is trivial. That is, $\mathrm{Stab}_G(\varepsilon) \cong I$.*

Proof. Every automorphism of G is represented by right-multiplication of Γ. Hence, every automorphism except the identity map will displace ε. $\square$

2.2.2 Deletion Relations with the Trivial Group

Now that sufficient tools are available, we prove some basic properties.

Proposition 2.14. *(The Reflexive Property) For any group* Γ, $\Gamma \to \Gamma$.

Proof. Let Γ be non-trivial, as the trivial case has been handled by the rigid tree $T(n)$. Let G be the Cayley graph $C'(\Gamma)$. Create a supergraph G' by adding a dominating vertex v with a pendant vertex u. Now, u is the only vertex of degree 1, and v is the only vertex adjacent to u. Hence, these two vertices are distinguished in G' from the vertices of G. Removing v leaves G and the isolated vertex u. Thus, Γ is the automorphism group for both G' and $G' - v$. □

A key part of our final proof relies on the trivial group deleting to any group. An additional vertex is considered with a special property on its stabilizer in the deleted graph.

Lemma 2.15. *Let* Γ *be a finite group. There exists a graph* H *and two vertices* $x, y \in V(H)$ *so that*

1. $\text{Aut}(H) \cong I$.
2. $\text{Aut}(H - x) \cong \Gamma$.
3. $\text{Stab}_{H-x}(y) \cong I$.

Proof. Let $G = C'(\Gamma)$. Let $n = |\Gamma|$. Order the group elements of Γ as $\alpha_1, \dots, \alpha_n$. Create a supergraph, H, by adding vertices as follows: For each α_i, create a copy T_{α_i} of $T(2n)$ and identify $u_0(T_{\alpha_i})$ with the vertex α_i in G (Here, $2n$ is used to distinguish these copies from the edge gadgets), and add a vertex x that is adjacent to $u_i(T_{\alpha_i})$ for all i. For each α_i, the leaf of T_{α_i} adjacent to v distinguishes α_i. Hence, no non-trivial automorphisms exist in H. However, $H - x$ restores all automorphisms π

from $\mathrm{Aut}(G)$ by mapping T_{α_i} to $T_{\pi(\alpha_i)}$ through the unique isomorphism. Finally, let $y = \alpha_1$. Since all automorphisms of G are given by left multiplication of group elements, only the trivial automorphism stabilizes α_1, so $\mathrm{Stab}_{H-x}(y) \cong I$. □

Note that this proof uses a very special vertex that enforces all vertices to be distinguished. Before producing examples where deleting a vertex removes symmetry, it may be useful to remark that such a distinguished vertex cannot be used.

Lemma 2.16. *Let G be a graph and $v \in V(G)$. Then, automorphisms in G that stabilize v form a subgroup in the automorphism group of $G - v$. That is, $\mathrm{Stab}_G(v) \leq \mathrm{Aut}(G - v)$.*

Proof. Let $\pi \in \mathrm{Stab}_G(v)$. The restriction map $\pi|_{G-v}$ is an automorphism of $G - v$. □

The implication of this lemma is removing a vertex with a trivial orbit cannot remove automorphisms. However, we can remove all symmetry in a graph using a single vertex deletion.

Lemma 2.17. *For any group Γ, $\Gamma \to I$.*

Proof. Assume $\Gamma \ncong I$, since the reflexive property handles this case. Let $G = C'(\Gamma)$ and $n = |\Gamma|$.

Let G_1, G_2 be copies of G with isomorphisms $f_1 : G \to G_1$ and $f_2 : G \to G_2$. Create a graph G' from these two copies as follows. For all elements γ in Γ, create a copy T_γ of $T(n)$ and identify $u_0(T_\gamma)$ with $f_1(\gamma)$ and $u_n(T_\gamma)$ with $f_2(\gamma)$. Note that $\mathrm{Aut}(G') \cong \Gamma$, since no vertices from G_1 can map to G_2 from the asymmetry of the T_γ subgraphs, and any automorphism of G_1 extends to exactly one automorphism of G_2.

Any automorphism π of $G' - f_1(\varepsilon)$ must induce an automorphism $\pi|_{G_2}$ of G_2. But the vertices of G_1 must then permute similarly (by the definition $\pi(f_1(x)) =$

$f_1 f_2^{-1} \pi f_2(x))$. Since $f_1(\varepsilon)$ is not in the image of π, π stabilizes $f_2(\varepsilon)$. Lemma 2.13 implies π must be the identity map. Hence, $\mathrm{Aut}(G' - f_1(\varepsilon)) \cong I$. □

2.2.3 Deletion Relations Between Any Two Groups

We are sufficiently prepared to construct a graph to reveal the deletion relation for all pairs of groups.

Theorem 2.18. *If* Γ_1 *and* Γ_2 *are groups, then* $\Gamma_1 \to \Gamma_2$*.*

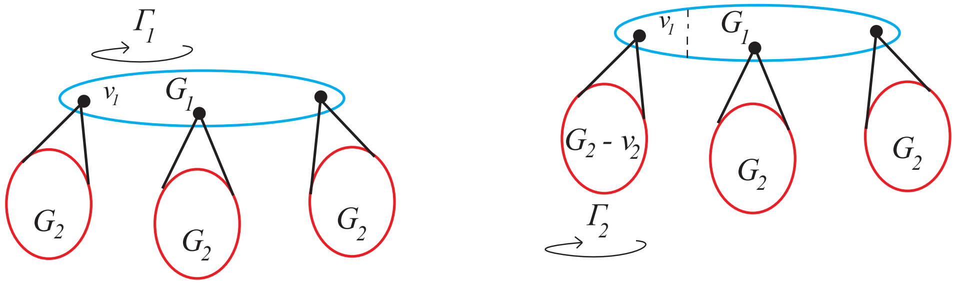
Proof. Assume both groups are non-trivial, since Lemmas 2.15 and 2.17 cover these cases. Let $G_1 = C'(\Gamma_1)$. Then identify $v_1 \in V(G_1)$ as the vertex corresponding to $\varepsilon \in \Gamma_1$. Note that $\mathrm{Stab}_{G_1}(v_1) \cong I$ as in Lemma 2.13. Also by Lemma 2.15, there exists a graph G_2 and vertex v_2 so that $I \overset{G_2 - v_2}{\longrightarrow} \Gamma_2$. Define $n_i = |\Gamma_i|$. Order the elements of Γ_1 as $\alpha_{1,1}, \alpha_{1,2}, \ldots, \alpha_{1,n_1}$ so that $\alpha_{1,1} = \varepsilon = v_1$.

We collect the necessary properties of G_1, G_2, v_1, v_2 before continuing. First, G_1 has automorphisms $\mathrm{Aut}(G_1) \cong \Gamma_1$ and v_1 is trivially stabilized ($\mathrm{Stab}_{G_1}(v_1) \cong I$). Second, G_2 is rigid ($\mathrm{Aut}(G_2) \cong I$) but $G_2 - v_2$ has automorphisms $\mathrm{Aut}(G_2 - v_2) \cong \Gamma_2$. The following construction only depends on these requirements.

Let $H_1, \ldots, H_{n_1}$ be copies of G_2. Construct a graph G by taking the disjoint union of G_1, H_1, …, H_{n_1}, and adding edges between $\alpha_{1,i}$ and every vertex of H_i, for $i = 1, \ldots, n_1$. Since $\mathrm{Aut}(H_i) \cong I$, the automorphism group of G cannot permute the vertices within each H_i. However, the vertices of G_1 can permute freely within $\mathrm{Aut}(G_1) \cong \Gamma_1$, since $H_i \cong H_j$ for all i, j. Hence, $\mathrm{Aut}(G) \cong \Gamma_1$.

When the copy of v_2 in H_1 is deleted from G, the automorphisms of $H_1 - v_2$ are Γ_2. However, the vertex v_1 of G_1 is now distinguished since it is adjacent to a copy of $G_2 - v_2$, unlike the other elements of Γ_1 in G_1 which are adjacent to a copy of G_2. This means the permutations of G_1 must stabilize v_1. Since $\mathrm{Stab}_{G_1}(v_1) = I$ by

Lemma 2.16, the only permutation allowed on G_1 is the identity. However, $H_1 - v_2$ has automorphism group Γ_2. Hence, $\operatorname{Aut}(G - v_2) \cong \Gamma_2$. $\square$



G with $\operatorname{Aut} G \cong \Gamma_1$. $G - v_2$ with $\operatorname{Aut} G - v_2 \cong \Gamma_2$.

Figure 2.2: The vertex deletion construction.

Figure 2.2 presents a visualization of the automorphisms in this construction before and after the deletion. A very similar construction produces this general result for the edge case.

Theorem 2.19. *If Γ_1 and Γ_2 are groups, then there exists a graph G and an edge $e \in E(G)$ so that $\Gamma_1 \xrightarrow{G-e} \Gamma_2$.*

Proof. Set $n_i = |\Gamma_i|$. Let $G_1 = C'(\Gamma_1)$ with v_1 corresponding to $\varepsilon \in \Gamma_1$ and order the elements of Γ_1 similarly to the proof of Theorem 2.18.

Form G_2 by starting with $C'(\Gamma_2)$ and making a copy T_γ of $T(2n_2)$ for each element $\gamma \in \Gamma_2$, identifying $\gamma \in V(C'(\Gamma_2))$ with $u_0(T_\gamma)$. Now, add an edge e between $u_{2n_2}(T_1)$ and $u_{2n_2-1}(T_1)$. This distinguishes the element ε as a vertex in $C'(\Gamma_2)$ and hence is stabilized. So, $\operatorname{Aut}(G_2) \cong I$ and if e is removed all group elements are symmetric again, so $\operatorname{Aut}(G_2 - e) \cong \Gamma_2$.

Notice that G_1, G_2, v_1, e satisfy the requirements of the construction of G in Theorem 2.18. Hence, the same construction (with e in place of v_2) provides an example of edge deletion from Γ_1 to Γ_2. $\square$

Note that the graph produced for Theorem 2.19 can be used for the proof of Theorem 2.18 by subdividing e and using the resulting vertex as the deletion point.

2.2.4 Generalizations

Theorem 2.18 can be extended to a sequence $\Gamma_0, \Gamma_1, \ldots, \Gamma_k$ of finite groups using two types of vertex deletions: single deletions or iterated deletions.

Question 2.20. Let $\Gamma_0, \Gamma_1, \ldots, \Gamma_k$ be finite groups. Do there exist a graph G and vertices $v_1, \ldots, v_k \in V(G)$ so that $\mathrm{Aut}(G) \cong \Gamma_0$ and for all $i \in \{1, \ldots, k\}$,

1. (Single Deletions) $\mathrm{Aut}(G - v_i) \cong \Gamma_i$?
2. (Iterated Deletions) $\mathrm{Aut}(G - v_1 - \cdots - v_i) \cong \Gamma_i$?

In fact, both of these types of deletions can be combined in an even more general situation. Suppose that an adversary selects finite groups $\Gamma_0, \Gamma_1, \ldots, \Gamma_k$ and a number $\ell \geq 1$. You then produce a graph G with $\mathrm{Aut}(G) \cong \Gamma_0$. The adversary then selects a map $\pi : \{1, \ldots, \ell\} \to \{1, \ldots, k\}$ and asks for ℓ vertices $v_1, \ldots, v_\ell$ so that for all $i \in \{1, \ldots, \ell\}$ the automorphism group of $G - v_1 - \cdots - v_i$ is isomorphic to $\Gamma_{\pi(i)}$. By carefully constructing G, you can be prepared for any such query from the adversary.

Theorem 2.21 (Adversarial Iterated Deletions)**.** *Let* $\Gamma_0, \Gamma_1, \ldots, \Gamma_k$ *be finite groups and fix* $\ell \geq 1$. *There exists a graph* G *with* $\mathrm{Aut}(G) \cong \Gamma_0$ *so that for all maps* $\pi : \{1, \ldots, \ell\} \to \{1, \ldots, k\}$, *there exist vertices* $v_1^\pi, \ldots, v_k^\pi \in V(G)$ *where* $\mathrm{Aut}(G - v_1^\pi - \cdots - v_i^\pi) \cong \Gamma_{\pi(i)}$ *for all* $i \in \{1, \ldots, k\}$.

Proof. Note that the case $k = 1$ holds by Theorem 2.18.

For every $i \in \{1, \ldots, k\}$, Lemma 2.15 implies there is a graph H_i with vertices $x_i, y_i \in V(H_i)$ so that $\mathrm{Aut}(H_i)$ is trivial, $\mathrm{Aut}(H_i - x_i) \cong \Gamma_i$, and $\mathrm{Stab}_{H_i - x_i}(y_i)$ is trivial. By Theorem 2.18, there exists a connected graph H_0 and vertex $v_0 \in V(H_0)$ so that $\mathrm{Aut}(H_0) \cong \Gamma_0$ and $\mathrm{Aut}(H_0 - v_0)$ is trivial. Observe that $\mathrm{Stab}_{H_0}(v_0)$ is trivial.

Construct the graph G starting from H_0 in ℓ iterations. Let $G_0 = H_0$ and we will build G_i from G_{i-1}.

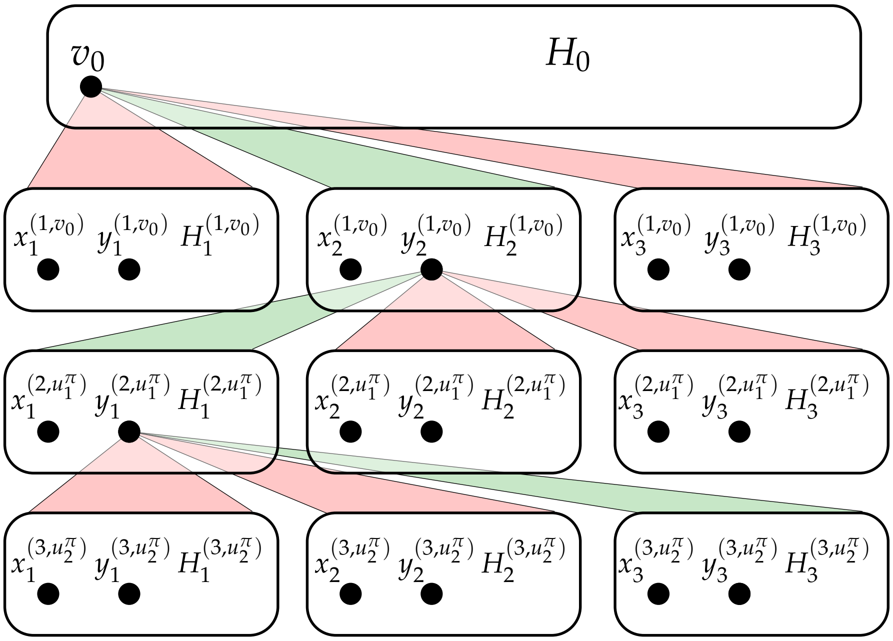
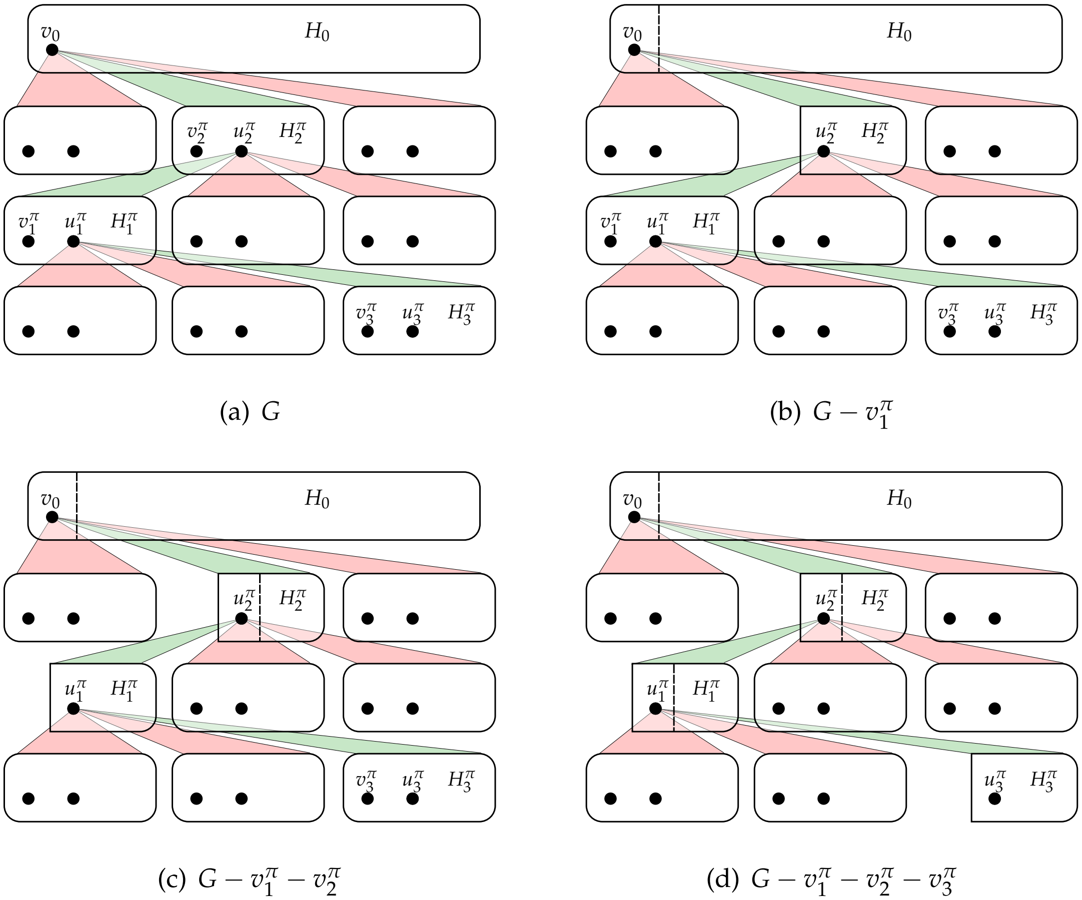


Figure 2.3: An example construction for Theorem 2.21 with $\pi = 213$.

Consider $i \geq 1$. For every vertex $v \in V(G_i)$ and $j \in \{1, \ldots, k\}$, create a copy $H_j^{(i,v)}$ of H_j and connect all vertices in $H_j^{(i,v)}$ to v. Let $x_j^{(i,v)}$ and $y_j^{(i,v)}$ denote the copies of x_j and y_j in $H_j^{(i,v)}$.

Now, let $G = G_\ell$. All automorphisms of G set-wise stabilize $V(G_0)$, so automorphisms of G induce automorphisms of $G_0 = H_0$. Since an isomorphic graph was attached to every vertex of G_0 and those graphs have trivial automorphism



(a) G (b) $G - v_1^\pi$

(c) $G - v_1^\pi - v_2^\pi$ (d) $G - v_1^\pi - v_2^\pi - v_3^\pi$

Figure 2.4: Deleting vertices in a construction for Theorem 2.21 with $\pi = 213$.

group, $\mathrm{Aut}(G) \cong \Gamma_0$.

Fix a map $\pi : \{1, \ldots, \ell\} \to \{1, \ldots, k\}$. Let $v_1^\pi = x_{\pi(1)}^{(1,v_0)}$ and $u_1^\pi = y_{\pi(1)}^{(1,v_0)}$, and for $i \in \{2, \ldots, k\}$ let $v_i^\pi = x_{\pi(i)}^{(i,u_{i-1}^\pi)}$ and $u_i^\pi = y_{\pi(i)}^{(i,u_{i-1}^\pi)}$. For all $i \in \{1, \ldots, k\}$, let H_i^π be the copy of $H_{\pi(i)}$ containing v_i^π.

Deleting v_1^π from G modifies the neighborhood of v_0, but not the neighborhood of any other vertex in $V(G_0)$. Therefore, all automorphisms of $G - v_1^\pi$ stabilize v_0. Since $\mathrm{Stab}_{H_0}(v_0)$ is trivial, all automorphisms of $G - v_1^\pi$ point-wise stabilize $V(G_0)$. However, the copy of $H_{\pi(1)}$ containing v_1^π now has automorphisms $\mathrm{Aut}(H_{\pi(1)} - v_1^\pi) \cong \mathrm{Aut}(H_{\pi(1)} - x_{\pi(1)}) \cong \Gamma_{\pi(1)}$.

Consider $i \in \{2, \ldots, \ell\}$. Deleting v_i^π from $G - v_1^\pi - \cdots - v_{i-1}^\pi$ modifies the neighborhood of u_{i-1}^π but not the neighborhood of any other vertex in that copy of $H_{\pi(i-1)} - v_{i-1}^\pi$. Therefore, u_{i-1}^π is stabilized, and so all automorphisms of $H_{i-1}^\pi - v_{i-1}^\pi \cong H_{\pi(i-1)} - x_{\pi(i-1)}$ stabilize that copy of $y_{\pi(i-1)}$ and so are trivial automorphisms. However, H_i^π lost its copy of v_i^π and now has automorphisms $\mathrm{Aut}(H_i^\pi - v_i^\pi) \cong \mathrm{Aut}(H_{\pi(i)} - x_{\pi(i)}) \cong \Gamma_{\pi(i)}$. □

2.2.5 Discussion

While we have constructions for almost any relationship between the automorphism groups of graphs and its vertex-deleted subgraphs, there remain open questions when restricted to special classes of graphs. For instance, the automorphism groups of trees are fully understood [118]. Let $\mathcal{G}_T$ be the class of groups that are represented by the automorphism groups of trees and $\mathcal{G}_F$ represented by automorphisms of forests[2]. The constructions in this chapter are not trees, so new methods will be required to answer the following questions. If we restrict to trees, can any group in $\mathcal{G}_T$ delete to any group in $\mathcal{G}_F$? Or, if we restrict to deleting leaves (and hence stay connected) can all pairs of groups in $\mathcal{G}_T$ delete to each other?

Another interesting aspect of our construction is that the resulting graphs are very large, with the order of the graphs cubic in the size of the groups. Which of these relations can be realized by small graphs? Can we restrict the groups that can appear based on the order of the graph? By Theorem 2.7, the current-best upper bound on the order of a graph G with automorphism groups isomorphic to a given group Γ is $|V(G)| \leq 2|\Gamma|$ and $\mathrm{Aut}(G) \cong \Gamma$. This has particular application to McKay's generation algorithm, where only "small" examples are usually computed (for example, all connected graphs up to 11 vertices were computed in [91]).

---

[2] An elementary proof shows that $\mathcal{G}_T = \mathcal{G}_F$.

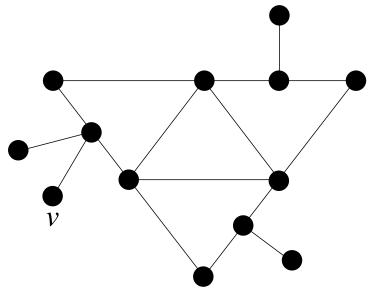


Figure 2.5: This graph G has $\mathrm{Aut}(G) \cong \mathbb{Z}_2$ and $\mathrm{Aut}(G - v) \cong \mathbb{Z}_3$.

To demonstrate that this is not trivial, see Figure 2.5 for a graph showing $\mathbb{Z}_2 \to \mathbb{Z}_3$.

While Theorem 2.18 shows that there exists a graph where *some* vertex can be deleted to demonstrate the deletion relations, our constructions have many other vertices that behave in very different ways when they are deleted. When relating to the Reconstruction conjecture, this raises questions regarding the combinations of automorphism groups that appear in the vertex-deleted subgraphs. For instance, if the multiset of vertex-deleted automorphism groups is provided, can one reconstruct the automorphism group? This question only gives the groups, but not the vertex-deleted subgraphs. An example is that n copies of S_{n-1} must reconstruct to S_n, but it is unknown whether the graph is K_n or nK_1. Since $\mathrm{Aut}(G) = \mathrm{Aut}(\overline{G})$, this ambiguity will always naturally arise. Can it arise in other contexts? Is the automorphism group recognizable from a vertex deck?

Chapter 3

Combinatorial Search

The goal of combinatorial search is to generate combinatorial objects that satisfy a given structural or extremal property. Combinatorial search techniques differ from local search techniques in that the method must be *exhaustive*: every object of a given order must be generated. This allows for a definitive result after executing the search.

Questions that have been answered using combinatorial search include:

1. Is there a projective plane of order 10? (Lam, Thiel, Swiercz [78].)
2. When do strongly regular graphs exist? (Many different works [121, 33, 94])
3. How many Steiner triple systems have order 19? (Kaski, Östergård [72])
4. What is the sixth van der Waerden number $W^2(6)$? (Kouril, Paul [75])
5. Does the Reconstruction Conjecture hold on small graphs? (McKay [91])

Throughout this thesis, I will always use a specific type of combinatorial search. Starting from a list of base objects, I will build objects piece-by-piece by performing some augmentations to the base objects. Typical augmentations include adding vertices or edges to a graph. In later chapters, I will use more complicated augmentations which are tied to the structure of the target objects. This chapter contains a high-level description of the search technique, the concerns that arise, and how those concerns are mitigated. Finally, a brief description of the *TreeSearch* library is given to show how combinatorial search can be parallelized using a common framework.

3.1 An Illustrated Guide to Combinatorial Search

In this section, we shall describe a way to visualize combinatorial search in a way that touches on most of the computational and mathematical concerns. Throughout the description, we shall refer to a common example of generating graphs using edge augmentations.

3.1.1 Labeled and Unlabeled Objects

Suppose a combinatorial search is defined by searching for combinatorial objects from a family $\mathcal{L}$ of *labeled objects*. Under the appropriate definition of isomorphism for those objects, let $\sim$ be the isomorphism relation and $\mathcal{U}$ be the family of *unlabeled objects*: the equivalence classes under $\sim$. Let $P : \mathcal{L} \to \{0,1\}$ be a *property*, and we wish to generate all objects X in $\mathcal{L}$ where $P(X) = 1$. We shall assume the property P is *invariant* under isomorphism ($\sim$): for all unlabeled objects $\mathcal{X} \in \mathcal{U}$ and labeled objects $X, X' \in \mathcal{X}$, $P(X) = P(X')$. In this case, we can define $P(\mathcal{X})$ for an unlabeled object $\mathcal{X}$ to be equal to $P(X)$ for any labeled object $X \in \mathcal{X}$.

Example. Let $\mathcal{L}$ be the set of graphs of order n. Then $\mathcal{U}$ is the family of unlabeled graphs where the standard relation of isomorphism ($\cong$) is used between graphs.

The property P could be $P(G) = 1$ if and only if G is 4-regular and G has chromatic number three.

3.1.2 Base Objects and Augmentations

A combinatorial search consists of a set $B \subset \mathcal{L}$ of *base objects* and an *augmentation*.

Let $B \subset \mathcal{L}$ be a set of labeled objects For a labeled object $X \in \mathcal{L}$, the augmentation defines a set $\mathcal{A}(X)$ of *augmented objects*. Let $\mathcal{D}(X)$ be the set of *deleted objects*, defined as

$$\mathcal{D}(X) = \{Y \in \mathcal{L} : Y \in \mathcal{A}(X)\}.$$

We shall consider our objects as being built *up*, so the set of augmented objects $\mathcal{A}(X)$ can be called the *above* objects while the deleted objects $\mathcal{D}(X)$ are the *downward* objects.

For isomorphism concerns, we shall assume that for an unlabeled object $\mathcal{X} \in \mathcal{U}$, any two labeled objects $X, X' \in \mathcal{X}$ have a bijection $\pi_{X,X'} : \mathcal{A}(X) \to \mathcal{A}(X')$ so that for all $Y \in \mathcal{A}(X)$, $Y \sim \pi_{X,X'}(Y)$. This allows us to define the augmented and deleted objects $\mathcal{A}(\mathcal{X})$ and $\mathcal{D}(\mathcal{X})$ for unlabeled objects $\mathcal{X} \in \mathcal{U}$ as well.

Example. For enumerating all graphs of order n, the empty graph of order n can serve as a base object. The augmentation step from a graph $G \in \mathcal{L}$ may be adding an edge to $E(G)$. Therefore, $\mathcal{A}(G) = \{G + e : e \in E(\overline{G})\}$ and $\mathcal{D}(G) = \{G - e : e \in E(G)\}$.

3.1.3 Search as a Poset

This relationship between augmented and deleted objects defines a partial order $\preceq$ on objects in $\mathcal{L}$. For $X, Y \in \mathcal{L}$, let $X \preceq Y$ be a cover relation if and only if $Y \in \mathcal{A}(X)$ (equivalently, $X \in \mathcal{D}(Y)$). Extending these cover relations by transitivity makes

$\preceq$ a partial order over $\mathcal{L}$. By our assumptions on unlabeled objects, $\preceq$ defines a partial order over $\mathcal{U}$.

The combinatorial search is *complete* if every unlabeled object $\mathcal{X} \in \mathcal{U}$ with $P(\mathcal{X}) = 1$, there is a base object $Y \in B$ so that $Y \preceq X$ for some $X \in \mathcal{X}$. A complete search ensures that every unlabeled object is visited at least once.

Figure 3.1 visualizes the poset $(\mathcal{U}, \preceq)$ and shows the unlabeled objects with property P as dots.

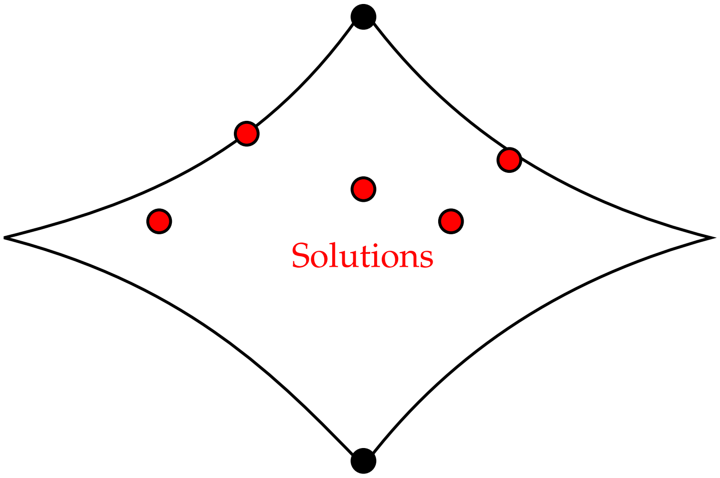


Figure 3.1: The search space as a poset.

Example. When generating graphs by edge augmentations, two unlabeled graphs G and H have $G \preceq H$ if and only if G is isomorphic to a subgraph of H. Since the empty graph is a subgraph of every graph, this search is complete.

3.1.4 Algorithm Structure

Consider the search space as a directed graph $\mathcal{H}$ with vertex set $V(\mathcal{H}) = \mathcal{L}$ (every vertex is a labeled object) and edge set $E(\mathcal{H})$ given by an edge from every $X \in \mathcal{L}$ to every $Y \in \mathcal{A}(X)$. This graph $\mathcal{H}$ is also the Hasse diagram of the poset $(\mathcal{L}, \preceq)$.

The combinatorial search algorithm is a depth-first search on $\mathcal{H}$ starting at every base object in B. This very basic view of the process is given as Algorithm 3.1.

Algorithm 3.1 CombinatorialSearch1(X)

```
if $P(X) \equiv 1$ then
  Output $X$
end if
for all $Y \in \mathcal{A}(X)$ do
  call CombinatorialSearch1($X$)
end for
```

Example. Algorithm 3.2 demonstrates a specific instance of Algorithm 3.1 where the objects are graphs, the augmentation involves adding edges, and the property $P(G) = 1$ if and only if G is 4-regular and has chromatic number three.

Algorithm 3.2 GraphSearch1(G)

```
if $\delta(G) \equiv \Delta(G) \equiv 4$ and $\chi(G) \equiv 3$ then
  Output $G$
end if
for all edges $e \in E(\overline{G})$ do
  call GraphSearch1($G + e$)
end for
```

Note that this process generates all labeled objects regardless of whether they can eventually lead to solutions. Further, the algorithm must operate on labeled graphs, so it currently does not remove multiple representatives of the same unlabeled object.

3.1.5 Sub-solutions and Pruning

A labeled object X (or unlabeled object $\mathcal{X}$) is a *sub-solution* if there exists a labeled object Y (or unlabeled object $\mathcal{Y}$) with $X \preceq Y$ and $P(Y) = 1$ (or $\mathcal{X} \preceq \mathcal{Y}$ and $P(\mathcal{Y}) = 1$). In the poset, the sub-solutions form a *down-set* generated by the objects with property P. Since these objects have some sequence of augmentations which lead to a solution, we must generate every unlabeled sub-solution at least once.

We also hope that we could immediately detect when our current object is not a sub-solution (there is no sequence of augmentations which leads to a solution). If we could immediately determine if the object is not a sub-solution, we could ignore these objects and generate only the sub-solutions. Unfortunately, the space of objects where we can efficiently detect that the object is not a sub-solution is not the complement of the sub-solution space. Figure 3.2 shows the regions of sub-solutions and those that are detectably not sub-solutions and there is a gap.

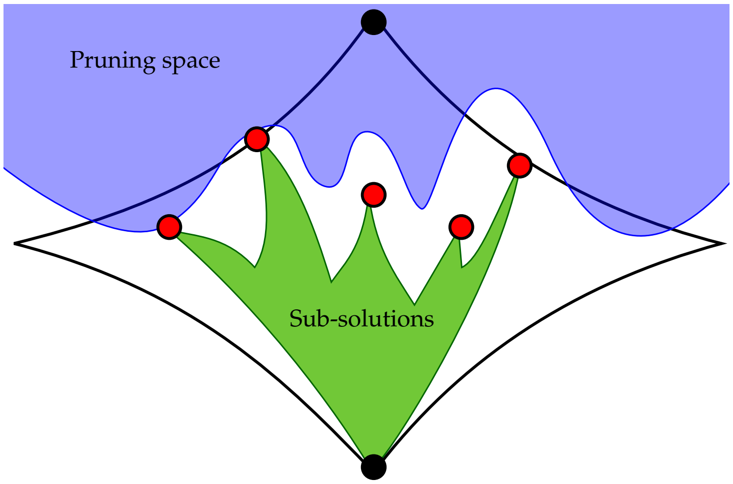


Figure 3.2: Sub-solutions and pruning space.

Suppose we create a procedure called Detect(X) which takes a labeled object $X \in \mathcal{L}$ and returns 1 only if X is not a sub-solution. We can modify the combinatorial search to utilize this procedure, as in Algorithm 3.3.

Example. For our example of generating 4-regular graphs with chromatic number three, a graph G with maximum degree $\Delta(G)$ at least five cannot extend to a 4-regular graph by adding edges. Similarly, a graph with chromatic number at least four cannot extend to a three-chromatic graph. Algorithm 3.4 demonstrates a

Algorithm 3.3 CombinatorialSearch2(X)

```
if $P(X) \equiv 1$ then
    Output $X$
end if
if Detect($X$) $\equiv 1$ then
    return
end if
for all $Y \in \mathcal{A}(X)$ do
    call CombinatorialSearch2($X$)
end for
```

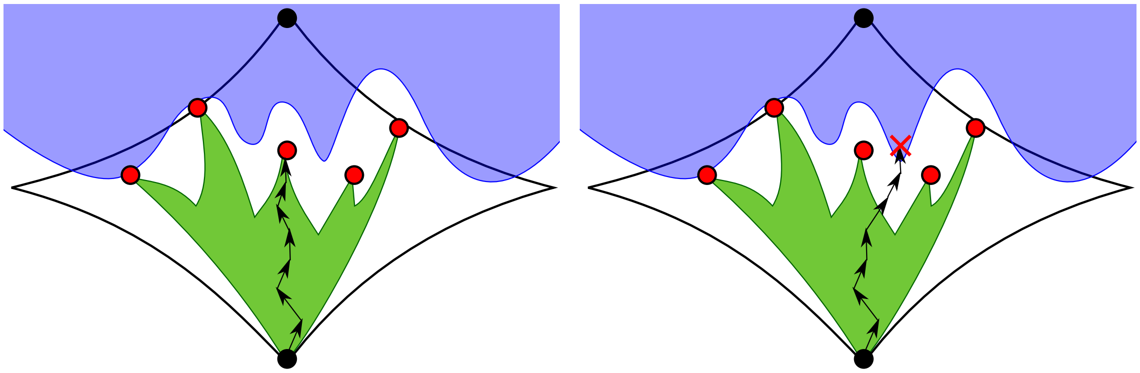
specific instance of Algorithm 3.3 by detecting non-sub-solutions using $\Delta(G)$ and $\chi(G)$.

Algorithm 3.4 GraphSearch2(G)

```
if $\delta(G) \equiv \Delta(G) \equiv 4$ and $\chi(G) \equiv 3$ then
    Output $G$
end if
if $\Delta(G) \geq 5$ or $\chi(G) \geq 4$ then
    return
end if
for all edges $e \in E(\overline{G})$ do
    call GraphSearch1($G + e$)
end for
```

An ideal path through the search space is to always remain within the sub-solutions and always hit a solution no matter what path is taken. However, this is not always possible. Sometime the augmentation will lead to an object which is not a sub-solution, but it takes a few more steps before reaching an object which is detectably not a sub-solution. Figure 3.3 demonstrates this issue.

In a non-ideal path, the number of steps between reaching a non-sub-solution and actually detecting that there is no solution greatly changes the run time of an algorithm. In this region, the algorithm is *thrashing*: augmenting in all possible ways for several steps before backtracking, with no hope of finding a solution.



(a) Ideal path.
(b) Non-ideal path.

Figure 3.3: Ideal and non-ideal paths in the search space.

To reduce the run time, the gap between sub-solutions and detectably non-sub-solutions must be reduced. This step requires modifying the augmentation step or proving a theorem.

3.1.6 Number of Paths to Each Unlabeled Object

Consider an unlabeled object $\mathcal{X}$ and a base object $Y \in B$. Let $\mathcal{Y}$ be the unlabeled object for Y. The interval between $\mathcal{Y}$ and $\mathcal{X}$ contains all unlabeled object $\mathcal{Z}$ so that $\mathcal{Y} \preceq \mathcal{Z} \preceq \mathcal{X}$ (Figure 3.4 shows such an interval). As stated before, every object $\mathcal{Z}$ in this interval must be generated at least once in order to guarantee that the search is complete.

However, if we are not careful, the Hasse diagram on this interval can be a tightly woven network. The number of times a labeled object $X \in \mathcal{X}$ is generated is equal to the number of paths in this graph.

Example. When generating a 4-regular graph G on n vertices by edge augmentations, there are a total of $2n$ total edges in G. If we assume that almost every subgraph of G is distinct up to isomorphism (which is not that much of an assumption, see Theorem 2.2), then there are 2^{2n} objects in the down-set generated

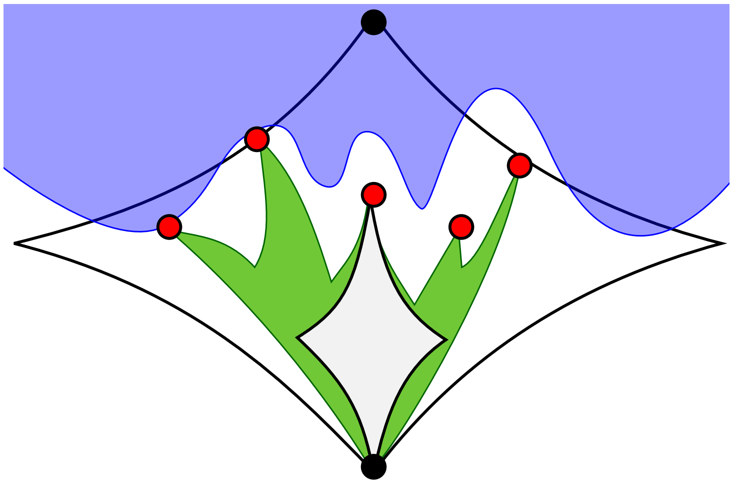


Figure 3.4: All nodes in the interval between a base example and a solution must be visited.

by G. Not only is that a large number of subobjects, but there are up to $(2n)!$ different ways to order the edges and build G by adding edges in that order. Since $(2n)! = 2^{\theta(n \log n)}$, this is asymptotically worse than just generating all subobjects. Further, this is based on the number of ways to build G as a *labeled* object. If G is a rigid graph, there are $n!$ different labeled graphs isomorphic to G, and hence $n! \cdot (2n)!$ possible ways to generate this isomorphism class. To avoid such drastic costs, something must be done to reduce isomorphic copies of G.

In an ideal world, every unlabeled object is generated at most once. Thus, there is at most one path in the Hasse diagram from any base object to any unlabeled object. This creates a tree structure to the poset, as seen in Figure 3.5. In Chapter 6, we describe a technique that performs this exact feat. In Chapter 10, we describe a different technique that reduces the number of paths, but cannot guarantee that every unlabeled object appears at most once.

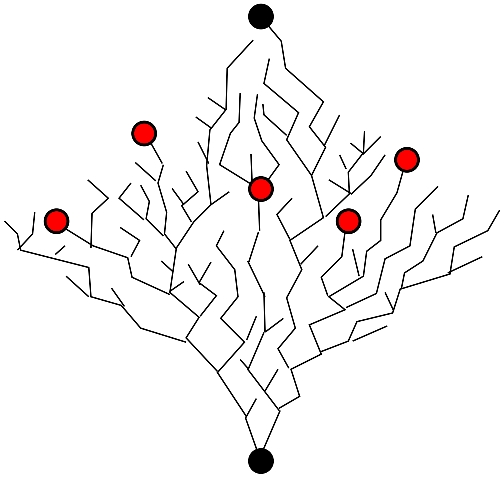


Figure 3.5: At most one path from a base example to any node creates a tree.

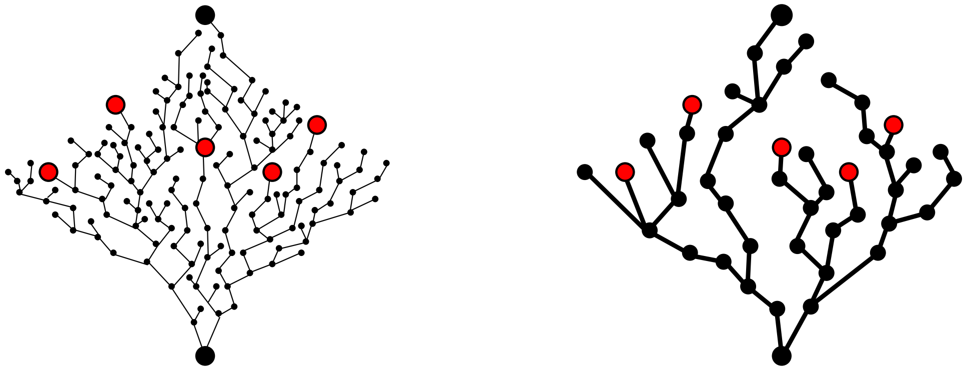
3.1.7 Count and Cost Tradeoff

There is an important tradeoff to consider when designing computational searches. The total computation time can be modeled as the amount of computation per object: $C(X)$. Thus,

$$\text{Total Time} = \sum_X C(X) = \text{Number of Objects} \times \text{Average Computation per Object}.$$

Depending on the problem considered, different augmentation steps can change either the number of objects in the space or the average computation per object. Figure 3.6 models computation time as the amount of black in the figure. In Figure 3.6(a), there are many nodes but they require very little cost per node. In contrast, Figure 3.6(b) has fewer nodes but each requires more cost. Which has less computation time? How could you tell before implementing and executing the search?

Similar tradeoffs occur with different methods to reduce isomorphic duplicates. One technique may remove all duplicates, but be overwhelmingly expensive to



(a) Many nodes, small cost per node.

(b) Fewer nodes, more cost per node.

Figure 3.6: Adjusting the search technique leads to different performance.

perform the computations. Another technique may be very quick per object, but suffers from combinatorial explosion in the number of isomorphic duplicates that appear. Finding the balance between these costs requires experience and experimentation.

Example. When generating graphs, it is almost always the case that generating isomorphic duplicates leads to combinatorial explosion. Using a technique such as canonical deletion (see Chapter 6) to remove isomorphic duplicates is a well-studied technique. However, this techniques works for augmenting by edges *or* augmenting by vertices. It is almost always the case that augmenting by edges requires at least as much computation per node as augmenting by vertices, except the number of objects that are visited is significantly more for edge augmentations. This is because edge augmentations visit every subgraph while vertex augmentations only visit *induced* subgraphs. Since induced subgraphs include knowledge about *non-edges*, more is known about the supergraphs that can be generated later, and non-sub-solutions may be detected earlier.

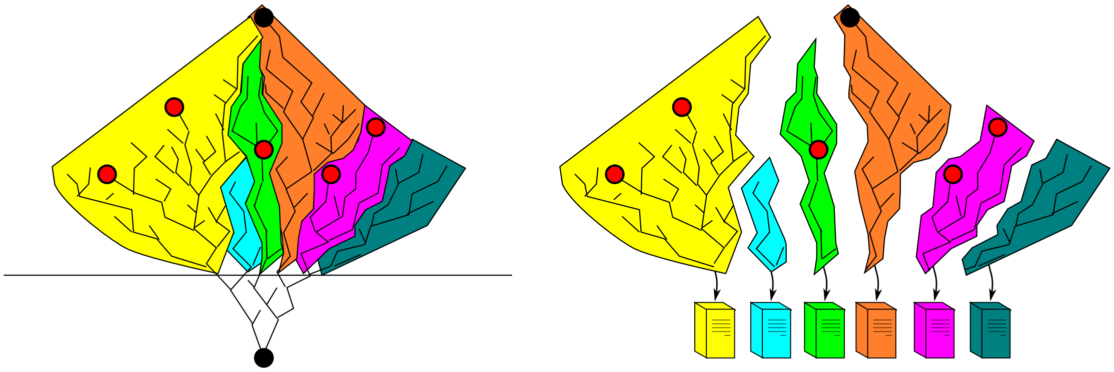
However, there are always counterexamples to the common theme: In Chapter 9 we will find that augmenting by *ears* is very similar to augmenting by edges, but it allows a certain invariant to be monotonic in these augmentations where it would not necessarily be for vertex augmentations. Further, in this example the ear augmentations are intimately tied to the structure of the target graphs and greatly assists the detection of non-sub-solutions.

In Chapter 11, we also go against the typical pattern by allowing isomorphic duplicates in favor of a shorter computation time per object. While this increases the number of labeled objects in total, the reduced amount of computation is significant and allows the search to expand to graphs of order 20 while other techniques became intractable around 14 vertices.

3.1.8 Partitioning and Parallelization

If we are given a tree-like search space, we can partition the search space by subtrees.

Given a base object $Y \in B$, the objects at the ith level (denoted $\mathcal{L}_i$) are those which are generated from Y by performing i augmentations. The objects in $\mathcal{L}_i$ then partition the space above the ith level: for $\mathcal{X} \in \mathcal{L}_i$, let $\mathcal{P}(\mathcal{X}) = \{\mathcal{Z} : \mathcal{X} \preceq \mathcal{Z}\}$. That is, the up-sets $\mathcal{P}(\mathcal{X})$ are disjoint and hence partition the space.



(a) Subtrees at a given level.

(b) Subtrees as independent jobs.

Figure 3.7: Partitioning the search space and parallelizing.

From this partition, since the recursive algorithm does not preserve information between recursive calls at the same level, these parts of the search space can be run independently. With appropriate communication protocols, every part $\mathcal{P}(\mathcal{X})$ can be run in a different process or even a different computation node. This allows for arbitrary parallelism by generating all objects in $\mathcal{L}_i$ for some i then running the search starting at every $\mathcal{X} \in \mathcal{L}_i$.

The following section discusses a software library called *TreeSearch* which enables this parallelization.

3.2 The *TreeSearch* Library

The previously described process of combinatorial search was purposely abstract. For all of the computational experiments described in this thesis, they are all implemented using a common library called *TreeSearch*. This library abstracts the recursive search, including the augmentation step, pruning, and outputting solutions. In addition, the library manages tracking statistics and distributing independent jobs to a supercomputer.

The distributed nature of *TreeSearch* is somewhat superficial. There is no actual communication or parallel programming occuring in the library itself. Instead, the library is built to manage the input and output files from parallel jobs in the Condor scheduler [134]. Condor is a scheduler that works for clusters and grids. In particular, the Open Science Grid [107], a collection of supercomputers around the country, has a running Condor scheduler. My access point is the UNL Campus Grid (designed by Weitzel [143]) as part of the Holland Computing Center.

3.2.1 Subtrees as Jobs

This tree structure allows for search nodes to be described via the list of children taken at each node. Typically, the breadth of the search will be small and these descriptions take very little space. This allows for a method of describing a search node independently of what data is actually stored by the specific search application. Moreover, the application may require visiting the ancestor search nodes in order to have consistent internal data. With the assumption that each subtree is computationally independent of other subtrees at the same level, one can run each subtree in a different process in order to achieve parallelization. These path descriptions make up the input for the specific processes in this scheme.

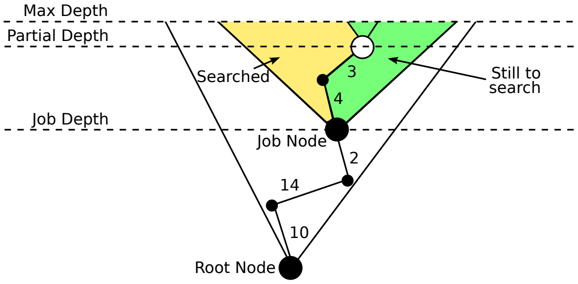


Figure 3.8: A partial job description.

Each path to a search node qualifies as a single job, where the goal is to expand the entire search tree below that node. A collection of nodes where no pair are in an ancestor-descendant relationship qualifies as a list of independent jobs. Recognizing that the amount of computation required to expand the subtree at a node is not always a uniform value, *TreeSearch* allows a maximum amount of time within a given job. In order to recover the state of the search when the execution times out, the concept of *partial jobs* was defined. A partial job describes the path from the root to the current search node. In addition, it describes which node in this path is the original job node. The goal of a partial job is to expand the remaining nodes in the subtree of the job node, without expanding any nodes to the left of the last node in this path. See Figure 3.8 to an example partial job and its position in the job subtree.

3.2.2 Job Descriptions

The descriptions of jobs and partial jobs are described using text files in order to minimize the I/O constraints on the distributed system. The first is the standard job, given by a line starting with the letter `J`. Following this letter are a sequence of numbers in hexadecimal. The first two should be the same, corresponding to the depth of the node. The remaining numbers correspond to the child values at each depth from the root to the job node.

A partial job is described by the letter `P`. Here, the format is the same as a standard job except the first number describes the depth of the job node and the second number corresponds to the depth of the current node. For example, the job and partial job given in Figure 3.8 are described by the strings below:

```
J 3 3 10 14 2
P 3 5 10 14 2 4 3
```

3.2.3 The *TreeSearch* Algorithm

Algorithm 3.5 details the full algorithm for *TreeSearch* execution. By running this algorithm after loading a job description (into an array called “job"), it will run the combinatorial search until either:

1. It completes and all objects are generated.
2. The amount of time spent goes beyond the specified KILLTIME.

3. The number of solutions is more than the maximum number of solutions: MAXNUMSOLS.

In the case of early termination, the position of the search is stored as a partial job to be run at a later date (or to be used as the start of a job generation mode).

The *TreeSearch* algorithm requires customization of the following methods:

1. pushNext() and pushTo(label): these methods encode the action of augmenting. The pushNext() method performs the next available augmentation, while the pushTo(label) method performs the augmentation encoded by the given label.

2. prune(): this method attempts to detect if the current object is not a sub-solution. Returns 1 if it detects a non-sub-solution, 0 otherwise.

3. isSolution(): this method attempts to detect if the current object satisfies the required property. Returns 1 if the object is a solution.

4. writeSolution(): this method writes the necessary data of a solution to standard output. This is typically in the form of a standard format of a graph output, such as an adjacency matrix or string encoding of an adjacency list. If the data is particularly long (and you want to avoid transmission over network), this method can be ignored during execution as the job description of that node is also output and the combinatorial object can be reconstructed from the augmentation steps.

5. pop(): This method reverses the previous augmentation from a pushNext() or pushTo() method.

For more information on compiling, integrating with, or running *TreeSearch*, the full software documentation is available as Appendix B.

Algorithm 3.5 DoSearch() — Recursive algorithm for *TreeSearch*.

```
Check if there are reasons to halt.
if mode $\equiv$ GENERATE and depth $\geq$ MAXDEPTH then
    call writeJob()
    return 0
end if
if runtime $\geq$ KILLTIME then
    call writePartialJob()
    Signal early termination.
    return -1
end if
if mode $\equiv$ LOADJOB then
    call pushTo(job[depth])
    result $\leftarrow$ doSearch()
    call pop()
    if result $< 0$ or depth $<$ JOBDEPTH then
        Do not continue with other augmentations.
        return result
    end if
    In this case, we are in a partial job and must continue augmenting.
end if
Attempt all possible augmentations.
while pushNext() $\not\equiv -1$ do
    if prune() $\equiv 0$ then
        if isSolution() $\equiv 1$ then
            numsols $\leftarrow$ numsols $+1$
            call writeSolutionJob()
            call writeSolution()
            if numsols $\geq$ MAXNUMSOLS then
                call writePartialJob()
                Signal early termination.
                return -1
            end if
        end if
        result $\leftarrow$ doSearch()
        if result $< 0$ then
            Early termination was signaled.
            call pop()
            return result
        end if
    end if
    call pop()
end while
return 0
```

Chapter 4

Chains of Width-2 Posets

In this chapter, we focus on finite posets as our combinatorial object.

Definition 4.1. A finite *partially ordered set* (or *poset*) is a pair $(X, \leq)$ where X is a finite set and $\leq$ is a relation between elements of X so that the following three properties hold for all $x, y, z \in X$:

1. (Reflexive) $x \leq x$.
2. (Antisymmetry) $x \leq y$ and $y \leq x$ implies $x = y$.
3. (Transitivity) $x \leq y$ and $y \leq z$ implies $x \leq z$.

Given a poset $P = (X, \leq)$, two elements $x, y \in X$ are *comparable* if $x \leq y$ or $y \leq x$. A *chain* is a subset $S \subseteq X$ so that all pairs $x, y \in S$ are comparable. In a chain S, the elements of S can be listed as $s_1, s_2, \ldots, s_k$ so that $s_i \leq s_j$ if and only if $i \leq j$.

We shall be concerned with the number of chains in a given poset.

Definition 4.2. Let $\mathrm{ch}(P)$ be the number of chains in P. This number includes the empty set.

Chains also define another invariant of a poset: width.

Definition 4.3. The *width* of a poset $P = (X, \leq)$ is the minimum number w of disjoint chains $S_1, S_2, \ldots, S_w$ so that $X = \cup_{i=1}^{w} S_i$.

If a poset has width one, then the poset is a single chain. The number of chains in these posets is simple to calculate: $\mathrm{ch}(P) = 2^{|P|}$. For larger width, the number of chains is not easy to compute.

Proposition 4.4. *For every integer N, there is at least one poset P_N with $\mathrm{ch}(P_N) = N$.*

Proof. Consider the binary representation of N: Suppose $N = \sum_{i=0}^{k} x_i 2^i$ where $x_i \in \{0, 1\}$ for each i. Let C_j denote a chain of j elements (for $j \geq 0$). Taking $P = \bigcup_{i=0}^{k} C_{ix_i}$ results in a poset with $N' = 1 + \sum_{i=0}^{k} x_i(2^i - 1) \leq N$ chains. By adding $N - N'$ incomparable elements to P, we find a poset with exactly N chains. $\square$

One problem with this construction is that it requires up to $2\log(n)$ disjoint chains, giving a non-constant width.

In this work, we ask: is there is a constant width w so that all postive integers are represented by the number of chains in a poset of width at most (or exactly) w? While we are not aware of any constant w that is sufficient, we ask if it holds for $w = 2$.

Definition 4.5. We say a number $k \in \mathbb{N}$ is *representable* if there is a poset P of width exactly two so that $\mathrm{ch}(P) = k$. If $k = \mathrm{ch}(P)$, we say k is *represented by P*.

Question 4.6 (Linek [83]). Is there an integer k_0 so that for all $k \geq k_0$ there is a poset of width two so that $\mathrm{ch}(P) = k$?

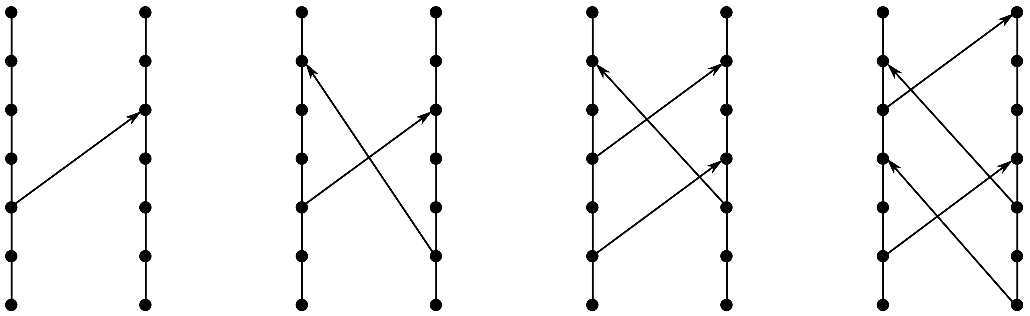
We shall provide strong evidence that the answer to this question is yes with $k_0 = 5$. By finding an automated method to count the number of chains in posets of width two, we find constructions that represent every number from 5 to around 7.3

million. These constructions further suggest that very few relations are required between elements from the two chains that partition the poset.

Suppose we have a poset P of width two, so there are two chains L and R that cover the entire poset. We will visualize L and R as vertical chains on the left and right side (respectively). Let $n = |L|$, $m = |R|$, $L = \{\ell_1, \ldots, \ell_n\}$, $R = \{r_1, \ldots, r_m\}$, where the elements ℓ_i and r_j are labeled so that $\ell_i \leq \ell_{i+1}$ and $r_j \leq r_{j+1}$.

Definition 4.7. Given $x < y$, we say *y covers x* if there is no z so that $x < z < y$. If y covers x and $|L \cap \{x, y\}| = |R \cap \{x, y\}| = 1$, we say the pair (x, y) is a *cover edge*.

By the above definition, the Hasse diagram of P can be drawn as two vertical lines for L and R, with ℓ_1, r_1 at the bottom and ℓ_n, r_m at the top, and left-to-right edges for cover edges (ℓ_i, r_j) and right-to-left edges for cover edges (r_j, ℓ_i). For example, see Figure 4.1 for posets with one, two, three, and four cover edges.



P_1
$\mathrm{ch}(P_1) =$

P_2
$\mathrm{ch}(P_2) =$

P_3
$\mathrm{ch}(P_3) =$

P_4
$\mathrm{ch}(P_4) = 364$

Figure 4.1: Examples of posets with few cover edges.

As an exercise, count the numbers of chains in the posets P_1, P_2, P_3, and P_4 of Figure 4.1. We count $\mathrm{ch}(P_1) = 344$, $\mathrm{ch}(P_2) = 353$, $\mathrm{ch}(P_3) = 357$, and $\mathrm{ch}(P_4) = 364$.

Definition 4.8. The *cover number* for an integer k, denoted $\mathrm{cov}(k)$, is the minimum number of cover edges in a poset of width two with k chains.

This work will develop several counting formulas for $\mathrm{ch}(P)$ when P has a small number of cover edges. Using this technique, formulas are generated for all configurations on two to five cover edges and the formulas are evaluated on a large variety of inputs corresponding to the number of points in L and R between the cover edges. We find that all numbers up to 10^6 (except 2 and 4) are representable by the number of chains in a width-two poset with at most six cover edges.

4.1 Products and Powers of Two

We begin by constructing representable numbers from products of representable numbers.

Definition 4.9. Given two posets P_1, P_2, the *join* $P_1 \vee P_2$ is the poset P on the base set $P_1 \cup P_2$ and relation $x \leq_P y$ if and only if $x, y \in P_1$ and $x \leq_{P_1} y$, $x, y \in P_2$ and $x \leq_{P_2} y$, or $x \in P_1$ and $y \in P_2$.

Proposition 4.10. *If P_1 is a poset of width two and P_2 is a poset of width at most two, then the poset $P = P_1 \vee P_2$ has width exactly two and* $\mathrm{ch}(P) = \mathrm{ch}(P_1) \cdot \mathrm{ch}(P_2)$.

Proof. Any selection of two chains C_1 and C_2 in P_1 and P_2 uniquely defines a chain in $P_1 \vee P_2$. $\square$

Corollary 4.11. *Suppose k is represented by a poset P. For all $i \geq 0$, $k2^i$ is representable and* $\mathrm{cov}(k2^i) \leq \mathrm{cov}(k) + 1$.

Proof. Let P' be a chain of i elements. The join $P \vee P'$ has $k2^i$ chains. If L, R is a partition of P with the smallest number of cover edges, then taking $L' = L \cup P'$ and $R' = R$ is a partition of $P \vee P'$ into two chains with at most one more cover edge than P. $\square$

Corollary 4.12. *If a number k factors as $k = 2^i k_1 k_2 \cdots k_\ell$, where each k_j is representable, then k is representable with* $\operatorname{cov}(k) \leq 1 + 2(\ell - 1) + \sum_{j=1}^{\ell} \operatorname{cov}(k_j)$.

Proof. Let $P_1, \ldots, P_\ell$ be width-two posets representing $k_1, \ldots, k_\ell$ and let P' be a chain of i elements. The poset $P = P_1 \vee P_2 \vee \cdots \vee P_\ell \vee P'$ represents k. If the poset P_j has a chain partition $L_j \cup R_j$ and $P' = L' \cup R'$, then we shall use the chain partition $L = \cup_{j=1}^{\ell} L_j \cup L'$ and $R = \cup_{j=1}^{\ell} R_j \cup R'$ for P. The number of cover edges in P with respect to L and R is at most the sum of the cover edges in the P_j, the two cover edges per join $P_j \vee P_{j+1}$ and one cover edge for the final join with P'. $\square$

4.2 An Even Number of Chains

In the previous section, we found that any representable number k provides a poset to represent all numbers $k2^i$ for any exponent i while adding at most one cover edge. We shall proceed to search for posets which represent odd numbers k, since all even products of such k will be representable. First, we shall consider which structures in width-two posets force an even number of chains.

Corollary 4.13. *Let P be a poset and $x \in P$. The number of chains contained in the down-set of x is even* ($\operatorname{ch}(D[x]) \equiv 0 \pmod 2$)*. The number of chains contained in the up-set of x is even* ($\operatorname{ch}(U[x]) \equiv 0 \pmod 2$)*.*

Proof. Note that $D[x] \cong (D[x] - x) \vee \{x\}$ and $U[x] \cong \{x\} \vee (U[x] - x)$. Hence, $\operatorname{ch}(D[x]) = 2\operatorname{ch}(D[x] - x)$ and $\operatorname{ch}(U[x]) = 2\operatorname{ch}(U[x] - x)$. $\square$

Definition 4.14. Let P be a poset of width two with chain partition L, R. A cover edge (ℓ_i, r_j) is *simple* if there does not exist a cover edge $(r_{j'}, \ell_{i'})$ with $i < i'$ and $j > j'$. Symmetrically, an edge (r_j, ℓ_i) is *simple* if there does not exist a cover edge $(\ell_{i'}, r_{j'})$ with $i > i'$ and $j < j'$.

A cover edge is simple if and only if the standard drawing of the Hasse diagram (with respect to L, R) has the cover edge uncrossed.

Theorem 4.15. *If P is a poset with a simple cover edge, then* $\mathrm{ch}(P)$ *is even.*

Proof. By symmetry, assume the simple cover edge is a left-to-right edge (ℓ_i, r_j). We shall consider two induced sub-posets P_1, P_2. The poset P_1 contains the elements $\{\ell_1, \ldots, \ell_i, r_1, \ldots, r_{j-1}\}$. The poset P_2 contains the elements $\{\ell_{i+1}, \ldots, \ell_n, r_j, \ldots, r_m\}$.

Let $A = \mathrm{ch}(P_1)$, $B = \mathrm{ch}(P_2)$. Also, let $D = \mathrm{ch}(D[\ell_i])$ and $U = \mathrm{ch}(U[r_j])$.

The number of chains in P is given by $BD + UA - UD$: Since ℓ_i is below all elements of P_2, any chain in $D[\ell_i]$ can be combined with any chain in P_2 to create a chain of P; Similarly, r_j is above all elements of P_1, so any chain in $U[r_j]$ can be combined with any chain in P_1 to create a chain of P. Since the (ℓ_i, r_j) is simple, $r_{j-1} \not\leq \ell_{i+1}$ and so we have counted every chain in P, but we have double-counted the chains which are a union of chains in $U[r_j]$ and $D[\ell_i]$.

Since D and U are even, every term of this count is even and hence $\mathrm{ch}(P)$ is even. □

Observe that Theorem 4.15 cannot become an "if and only if" condition since the poset P_4 in Figure 4.1 has 364 chains but no simple cover edges.

Now we know even numbers are representable when their odd factors are representable and posets with simple cover edges result in an even number of chains. In the following section, we will generate formulas for $\mathrm{ch}(P)$ when the set of cover edges is fixed and the number of points in L and R between the cover edges is varied. We shall focus on finding odd numbers, so we will ignore the configurations with simple cover edges.

4.3 Configurations and Parameterized Posets

In this section we define *configurations* to be the minimal width two posets with k “independent" split relations. From these configurations, we insert points between the split relations to generate larger posets with k split relations. By analyzing the structure of these posets, we create formulas to count the number of chains in the posets without needing to generate the actual poset.

Definition 4.16 (Configurations)**.** Let $k \geq 1$ be given. A *configuration* of order k is a width-two poset $C = (L \cup R, \leq)$ where

1. $k = |L| = |R|$,
2. every $\ell \in L$ has exactly one split relation containing ℓ, and
3. every $r \in R$ has exactly one split relation containing r.

Observe that the split relations between the left and right chains of a configuration C induce a perfect matching between L and R in the Hasse diagram of C. The edges of this matching can be directed according to the direction of the relation. Since these relations are cover relations, the edges from L to R must be *parallel* (i.e. for an edge $\ell_i \to r_j$ there is no edge $\ell_{i'} \to r_{j'}$ with $i' < i$ and $j < j'$). Similarly, the edges from R to L are parallel.

Therefore, all configurations with k relations can be generated by selecting a function $\sigma : \{1, \ldots, k\} \to \{-1, +1\}$ where

$$\sigma(i) = \begin{cases} +1 & \text{if the split relation containing } \ell_i \text{ is ordered } \ell_i \leq r_j, \\ -1 & \text{if the split relation containing } \ell_i \text{ is ordered } r_j \leq \ell_i. \end{cases}$$

and then selecting a perfect matching $M \subseteq E(K_{k,k})$ where the incoming and outgoing edges from L are parallel.

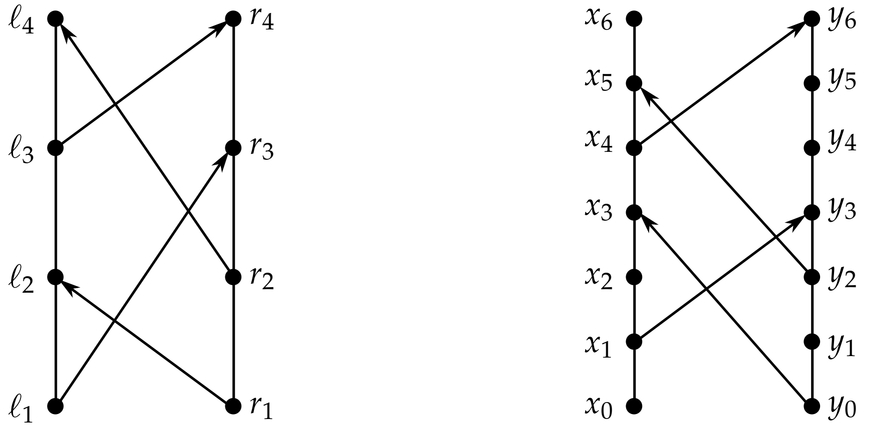
Definition 4.17 (Parameterized Posets)**.** Let $k \geq 1$ and $C = (L \cup R, \leq)$ be a configuration of order k. Fix integer vectors $\mathbf{a} = (a_0, a_1, \ldots, a_k)$ and $\mathbf{b} = (b_0, b_1, \ldots, b_k)$ where $a_i, b_j \in \mathbb{N} = \{0, 1, 2, \ldots\}$. Set $A_j = j + \sum_{i=0}^{j-1} a_i$, $B_j = j + \sum_{i=0}^{j-1} b_i$, $A = k + \sum_{i=0}^{k} a_i$ and $B = k + \sum_{i=0}^{k} b_i$. The *parameterized poset* generated by C, denoted $P_C(a_0, a_1, \ldots, a_k; b_0, b_1, \ldots, b_k)$, is the poset on elements

$$L' = \{x_0 \leq x_1 \leq \cdots \leq x_A\} \text{ and } R' = \{y_0 \leq y_1 \leq \cdots \leq y_B\},$$

for all split relations $\ell_j \leq r_{j'}$ in C, there is a split relation $x_{A_j} \leq y_{B_{j'}}$, and for all split relations $r_j \leq \ell_{j'}$ there is a split relation $y_{B_j} \leq x_{A_{j'}}$.

Observe that the parameterized poset $P_C(\mathbf{a}; \mathbf{b})$ can be built by inserting a_i points between ℓ_i and ℓ_{i+1} (for $i = 0$, insert a_0 points before ℓ_1; for $i = k$, insert a_k points after ℓ_k) and inserting b_j points between r_j and r_{j+1} (for $j = 0$, insert b_0 points before r_1; for $j = k$, insert b_k points after r_k). Figure 4.2 shows an example configuration and parameterized poset. Also, Figure 4.3 demonstrates how the posets in Figure 4.1 can be generated as parameterized posets by listing the parameters $\mathbf{a}$ and $\mathbf{b}$.

From the set of k-order configurations, we can generate all width-two posets with k independent split relations by selecting an appropriate set of parameters. Further, the parameterized posets have simple relations if and only if the configurations have simple relations. Therefore, to search for odd representable numbers, we only need to consider configurations with no simple relations. What is even more interesting is that we can count the number of chains in $P_C(\mathbf{a}; \mathbf{b})$ without generating the poset and using a chain-counting algorithm. First, we define this



(a) A configuration C_4 of order 4.

(b) $P_{C_4}(1,1,0,0,1;0,1,0,2,0)$.

Figure 4.2: A configuration of order four and a parameterized poset.

counting function.

Definition 4.18 (Counting Functions)**.** Let $k \geq 1$, C be a configuration of order k. The *counting function* generated by C is the function on domain $(\mathbf{a};\mathbf{b}) \in \mathbb{N}^{k+1} \times \mathbb{N}^{k+1}$ defined as $f_C(\mathbf{a};\mathbf{b}) = \mathrm{ch}(P_C(\mathbf{a};\mathbf{b}))$.

Given a configuration C, we can algorithmically write an algebraic formula to represent f_C. The strategy is to partition the chains of $P_C(\mathbf{a};\mathbf{b})$ by assigning every chain $S \subset P_C(\mathbf{a};\mathbf{b})$ a *canonical* maximal chain. Since the maximal chains of $P_C(\mathbf{a};\mathbf{b})$ correspond to maximal chains of C, we can write a formula for f_C by iterating over all maximal chains $S \subset C$ and counting the number of chains in $P_C(\mathbf{a};\mathbf{b})$ whose canonical maximal chains correspond to S.

4.3.1 Canonical Maximal Chains

We shall consider a configuration C, paramaters $\mathbf{a} = (a_0, a_1, \ldots, a_k)$ and $\mathbf{b} = (b_0, b_1, \ldots, b_k)$, and count the number of chains in $P_C(\mathbf{a};\mathbf{b})$. We first partition the chains of $P_C(\mathbf{a};\mathbf{b})$ by assigning each chain Z in $P_C(\mathbf{a};\mathbf{b}$ to a canonical maximal

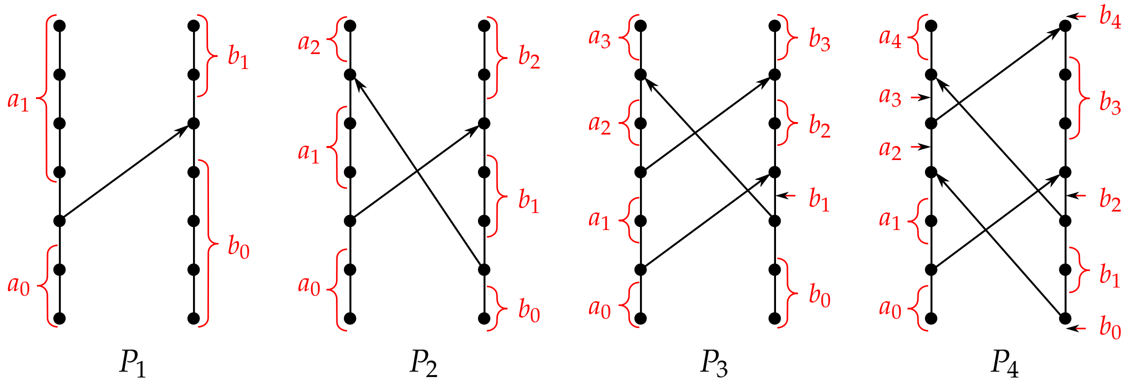


Figure 4.3: Posets from Figure 4.1 with parameters listed.

chain $m(Z)$ in C.

The selection of $m(Z)$ is given as a greedy algorithm. If $Z \subseteq L'$ or $Z \subseteq R'$, then $m(Z)$ is equal to L or R, respectively. Otherwise, Z has some alternating pattern between L' and R'. We select elements of C that induce the relations necessary for these alternations, and remove ambiguity by making a greedy decision: for a pair z_k, z_{k+1} where $z_k \in L$ and $z_{k+1} \in R$, we select a cover relation $\ell_i \leq r_j$ from C among all such relations where $z_k \leq x_{A_i} \leq y_{B_j} \leq z_{k+1}$ by taking the relation with y_{B_j} closest to z_{k+1} (a similar choice is made for the other direction). In the sense of tracing $m(Z)$ on the Hasse diagram on $P_C(\mathbf{a};\mathbf{b})$, we draw a path that hits all the points of Z while changing from L' to R' (or R' to L') at the *latest possible* cover relation.

Definition 4.19 (Canonical Maximal Chains)**.** Let Z be a non-empty chain in $P_C(\mathbf{a};\mathbf{b})$. Define a map $m(Z)$ from chains in $P_C(\mathbf{a};\mathbf{b})$ to chains in C as follows:

1. If $Z \subseteq L'$, then $m(X) = L$.
2. If $Z \subseteq R'$, then $m(X) = R$.
3. Otherwise, $Z \cap L' \neq \emptyset$ and $Z \cap R' \neq \emptyset$. List the elements of Z as $z_1, z_2, \ldots, z_m$ for $m = |Z|$. Let $i_1, i_2, \ldots, i_\ell$ and $j_1, j_2, \ldots, j_k$ be all indices so that $z_{i_t} \in L$ but

$z_{i_t+1} \in R$ for all $t \in \{1, \ldots, \ell\}$, and $z_{j_t} \in R$ but $z_{j_t+1} \in L$ for all $t \in \{1, \ldots, k\}$.

a) For all $t \in \{1, \ldots, \ell\}$, let $i_t^{(L)}$ be the smallest index so that there is an index $j_t^{(L)}$ where $z_{i_t} \leq x_{i_t^{(L)}} \leq y_{j_t^{(L)}} \leq z_{i_t+1}$. Define $L_t = \{x_{i_{t-1}^{(R)}}, \ldots, x_{i_t^{(L)}}\}$.

b) For all $t \in \{1, \ldots, k\}$, let $j_t^{(R)}$ be the smallest index so that there is an index $i_t^{(R)}$ where $z_{j_t} \leq y_{j_t^{(R)}} \leq x_{i_t^{(R)}} \leq z_{j_t+1}$.

c) If $i_1 < j_1$, then let $I = \{x_0, \ldots, x_{i_1^{(L)}}\}$, for even $t \in \{2, \ldots, \ell\}$ let $S_t = \{y_{j_{t-1}^{(L)}}, \ldots, y_{j_t^{(R)}}\}$, and for odd $t \in \{2, \ldots, k\}$ let $S_t = \{x_{i_{t-1}^{(R)}}, \ldots, x_{i_t^{(L)}}\}$.

d) If $j_1 < i_1$, then let $I = \{y_0, \ldots, y_{j_1^{(R)}}\}$ for even $t \in \{2, \ldots, k\}$ let $S_t = \{x_{i_{t-1}^{(R)}}, \ldots, x_{i_t^{(L)}}\}$, and for odd $t \in \{2, \ldots, \ell\}$ let $S_t = \{y_{j_{t-1}^{(L)}}, \ldots, y_{j_t^{(R)}}\}$.

e) If $i_\ell > j_k$, let $T = \{x_{i_k^{(R)}}, \ldots, x_A\}$.

f) If $j_k > i_\ell$, let $T = \{y_{j_\ell^{(L)}}, \ldots, y_B\}$.

Finally, set

$$m(Z) = I \cup \left(\cup_{t=2}^{k} S_t \right) \cup T.$$

Lemma 4.20. *Let $k \geq 1$ and $C = (L \cup R, \leq)$ be a configuration of order k. Let S be a maximal chain of C so that $S \cap L \neq \emptyset$ and $S \cap R \neq \emptyset$. Given $\mathbf{a}, \mathbf{b} \in \mathbb{N}^{k+1}$, set $A_j = j + \sum_{i=0}^{j-1} a_i$, $B_j = j + \sum_{i=0}^{j-1} b_i$, $A = k + \sum_{i=0}^{k} a_i$ and $B = k + \sum_{i=0}^{k} b_i$. Suppose the chain S contains t cover edges of C.*

- *There exist integers $i_1 < i_2 < \cdots < i_t$ and $j_1 < j_2 < \cdots < j_t$ so that x_{i_m} and y_{j_m} are the two elements of L' and R' involved in the mth split relation of S.*

- *For all $m \in \{1, \ldots, t\}$, let i'_m be the minimum i so that $i_m < i \leq i_{m+1}$ and the relation on ℓ_i has the form $\ell_i \geq r_j$ (if no such i exists let $i'_m = i_{m+1}$ when $m < t$ and $i'_m = k$ when $m = t$).*

- *Let* j'_m *be the minimum* j *so that* $j_m < j \leq j_{m+1}$ *and the relation on* r_j *has the form* $r_j \geq \ell_i$ *(if no such* j *exists let* $j'_m = j_{m+1}$ *when* $m < t$ *and* $j'_m = k$ *when* $m = t$*).*
- *If the minimum element of* S *is in* L*, let* $I_S = \left(2^{A_{i_1}} - 1\right)$ *and* $s = 0$*.*
- *If the minimum element of* S *is in* R*, let* $I_S = \left(2^{B_{j_1}} - 1\right)$ *and* $s = 1$*.*
- *If the maximum element of* S *is in* L*, let* $T_S = \left(2^{(A_{i'_t} - A_{i_t})} - 1\right) 2^{(A - A_{i'_t} + 1)}$*.*
- *If the maximum element of* S *is in* R*, let* $T_S = \left(2^{(B_{j'_t} - B_{j_t})} - 1\right) 2^{(B - B_{j'_t} + 1)}$*.*

Then, the number of chains in $P_C(\mathbf{a};\mathbf{b})$ *with canonical maximal chain induced by* S *is*

$$\begin{aligned}\mathrm{ch}_S(\mathbf{a};\mathbf{b}) = I_S \times &\prod_{\substack{m=1\\ m\equiv s \pmod 2}}^{t-1} \left[\left(2^{(A_{i'_m} - A_{i_m})} - 1\right) 2^{(A_{i_{m+1}} - A_{i'_m} + 1)}\right] \\ \times &\prod_{\substack{m=1\\ m\not\equiv s \pmod 2}}^{t-1} \left[\left(2^{(B_{j'_m} - B_{j_m})} - 1\right) 2^{(B_{j_{m+1}} - B_{j'_m} + 1)}\right] \times T_S.\end{aligned}$$

Proof. To count the number of chains $X \subseteq P_C(\mathbf{a};\mathbf{b})$ so that $m(X) = S$, we may select subsets of S by the segments of $S \cap L$ and $S \cap R$. For S to be the canonical maximal chain of a set X, every segment of $S \cap L$ or $S \cap R$ corresponds to a segment of L' or R' in $P_C(\mathbf{a};\mathbf{b})$ and these segments must contain at least one element of X. The indices $A_{i_1}, \ldots, A_{i_t}$ and $B_{j_1}, \ldots, B_{j_t}$ mark the end of these segments within $P_C(\mathbf{a};\mathbf{b})$.

For a segment $\{\ell_{i_m}, \ldots, \ell_{i'_m}, \ldots, \ell_{i_{m+1}}\}$ of $S \cap L$, X must contain a non-empty set within $\{x_{A_{i_m}}, \ldots, x_{A_{i'_m}} - 1\}$, or else the algorithm to create $m(X)$ would not have selected $r_{j_m} \leq \ell_{i_m}$ as a cover edge, since there is a cover relation ending at $\ell_{i'_m}$. There are $A_{i'_m} - A_{i_m}$ elements between the corresponding elements of $P_C(\mathbf{a};\mathbf{b})$ and $A_{i_{m+1}} - A_{i'_m} + 1$ remaining elements in the segment. There are

$$\left(2^{(A_{i'_m} - A_{i_m})} - 1\right) 2^{(A_{i_{m+1}} - A_{i'_m} + 1)}$$

possible ways to select a subset of this region so that the cover relation $r_{j_m} \leq \ell_{i_m}$ is selected.

By symmetry, there are

$$\left(2^{(B_{j'_m} - B_{j_m})} - 1\right) 2^{(B_{j_{m+1}} - B_{j'_m} + 1)}$$

possible ways to select a subset of a segment $\{y_{B_{j_m}}, \ldots, y_{B_{j'_m}}, \ldots, y_{B_{j_{m+1}}}\}$ corresponding to a segment $\{r_{j_m}, \ldots, r_{j'_m}, \ldots, r_{j_{m+1}}\}$ of $S \cap R$ so that the cover relation $\ell_{i_m} \leq r_{j_m}$ is selected.

The initial segments and terminal segments are counted by I_S and T_S using a similar formula. Note that I_S counts the number of non-empty sets in the initial segment, while T_S needs to check for a non-empty set before the next cover edge.

□

Theorem 4.21. *For a configuration C of order k,*

$$f_C(\mathbf{a}; \mathbf{b}) = \left(2^{k + \sum_{i=0}^{k} a_i} - 1\right) + \left(2^{k + \sum_{j=0}^{k} b_j} - 1\right) + 1 + \sum_{\substack{S \subset C,\ \textit{max'l chain} \\ S \cap L \neq \emptyset, S \cap R \neq \emptyset}} \mathrm{ch}_S(\mathbf{a}; \mathbf{b}).$$

Example 4.22. Consider the configuration C_4 from Figure 4.2(a). Theorem 4.21

states the function f_{C_4} can be computed by the formula

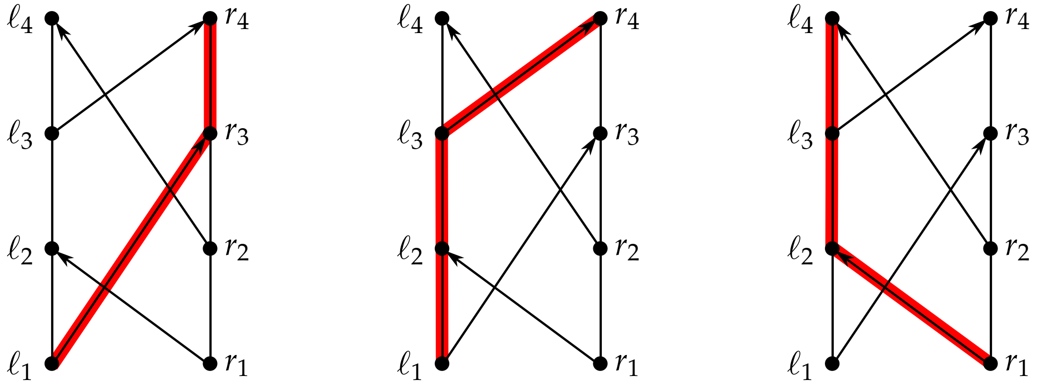
$$\begin{aligned}
f_{C_4}(\mathbf{a};\mathbf{b}) = {} & \left(2^{4+a_0+a_1+a_2+a_3+a_4}-1\right) && (L)\\
& + \left(2^{4+b_0+b_1+b_2+b_3+b_4}-1\right) && (R)\\
& + 1 && (\emptyset)\\
& + \left(2^{a_0+1}-1\right)\left(2^{b_3+1}-1\right)\left(2^{b_4+1}\right) && (S_1=\{\ell_1,r_3,r_4\})\\
& + \left(2^{a_0+a_1+a_2+3}-1\right)\left(2^{b_4+1}-1\right) && (S_2=\{\ell_1,\ell_2,\ell_3,r_4\})\\
& + \left(2^{b_0+1}-1\right)\left(2^{a_2+a_3+2}-1\right)\left(2^{a_4+1}\right) && (S_3=\{r_1,\ell_2,\ell_3,\ell_4\})\\
& + \left(2^{b_0+b_1+2}-1\right)\left(2^{a_4+1}-1\right) && (S_4=\{r_1,r_2,\ell_4\})\\
& + \left(2^{b_0+1}-1\right)\left(2^{a_2+2}-1\right)\left(2^{b_4+1}-1\right). && (S_5=\{r_1,\ell_2,\ell_3,r_4\})
\end{aligned}$$

The five maximal chains $S_1, \ldots, S_5$ other than L and R are shown in Figure 4.4.

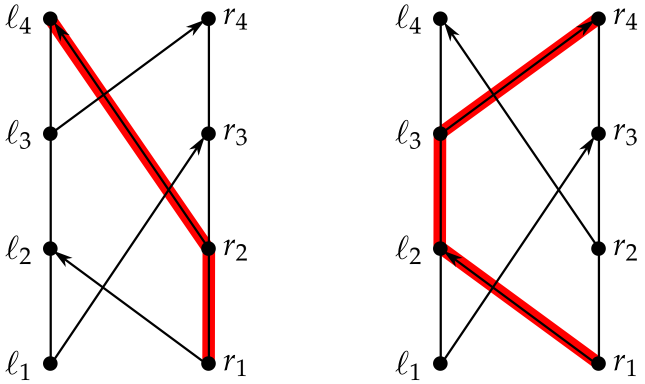
Proof of Theorem 4.21. Every non-empty chain T in $P_C(\mathbf{a};\mathbf{b})$ reduces to some canonical maximal chain in C. There are $2^{|L|}-1$ non-empty chains $T \subseteq L$ and $2^{|R|}-1$ non-empty chains $T \subseteq R$. (Observe $|L| = k + \sum_{i=0}^{k} a_i$ and $|R| = k + \sum_{j=0}^{k} b_j$.) All other non-empty chains intersect both L and R, and thus are counted by $\mathrm{ch}_S(\mathbf{a};\mathbf{b})$ for some maximal chain $S \subset C$ where $S \cap L \neq \emptyset$ and $S \cap R \neq \emptyset$. Finally, the empty set is a chain, which contributes exactly one to the total number. Summing these terms results in the specified formula. □

4.4 Generating Configurations and Formulas

Fix $k \geq 2$ to be a number of cover edges of a configuration. We can generate all configurations C with k cover edges by first selecting a vector $\mathbf{p} = (p_1, \ldots, p_k) \in$



(a) $S_1 = \{\ell_1, r_3, r_4\}$. (b) $S_2 = \{\ell_1, \ell_2, \ell_3, r_4\}$. (c) $S_3 = \{r_1, \ell_2, \ell_3, \ell_4\}$.



(d) $S_4 = \{r_1, r_2, \ell_4\}$. (e) $S_5 = \{r_1, \ell_2, \ell_3, r_4\}$.

Figure 4.4: The five maximal chains of C_4 other than L and R.

$\{-1, +1\}^k$ where p_i specifies the direction of the relation on ℓ_i:

$$p_i = \begin{cases} +1 & \text{the cover edge on } \ell_i \text{ is } \ell_i \geq r_j \text{ for some } j \\ -1 & \text{the cover edge on } \ell_i \text{ is } \ell_i \leq r_j \text{ for some } j \end{cases}.$$

For the elements $\ell_1, \ell_2, \ldots, \ell_k$ of L, let $r_{j_1}, r_{j_2}, \ldots, r_{j_k}$ be the elements of R so that ℓ_i is in a cover relation with r_{j_i} for all $i \in \{1, \ldots, k\}$. Then, since the cover edges correspond to cover relations, all relations ℓ_i, $\ell_{i'}$ with the same sign ($p_i = p_{i'}$) must be parallel ($r_{j_i} < r_{j_{i'}}$). By iterating over all feasible orderings of R, we can construct all possible configurations with k cover edges.

Since we are only concerned with representing odd numbers, we can immediately ignore configurations with simple cover edges, as Theorem 4.15 implies these configurations can only induce parameterized posets with an even number of chains. The rest of the configurations can be reduced to one representative per isomorphism class, by computing isomorphism among the Hasse diagrams using standard isomorphism tools (such as [93]). Table 4.1 shows how many configurations with k cover edges exist up to isomorphism, for $k \in \{2, \ldots, 6\}$.

k	2	3	4	5	6
N_k	1	1	3	5	17

Table 4.1: Number N_k of configurations with k cover edges, up to isomorphism.

Now, for every remaining configuration C, we can automatically generate the formula $f_C(\mathbf{a}; \mathbf{b})$ using Lemma 4.20 and Theorem 4.21. Since there are very few configurations for $k \leq 6$ and generating the formulas is a very quick operation, we do not describe this process in detail. However, what takes more time is to evaluate the formulas on many inputs and determine what numbers are representable by these parameterized posets.

4.5 Evaluating Formulas

After generating the formulas f_C for every configuration C with at most k cover edges, we must evaluate the formulas on all inputs $\mathbf{a}, \mathbf{b}$. However, there are an infinite number of formulas, so we must select a finite subset to check.

Observe that if $\mathbf{a}, \mathbf{b}$ and $\mathbf{a}', \mathbf{b}'$ are parameter sets so that $a_i \leq a_i'$ for all i and $b_j \leq b_j'$ for all j, then $f_C(\mathbf{a}; \mathbf{b}) \leq f_C(\mathbf{a}'; \mathbf{b}')$. Therefore, if we select values for $\mathbf{a}, \mathbf{b}$ in sequential order (i.e. set $a_1, a_2, \ldots, a_i$ for increasing i until $i = k$, then set

$b_1, b_2, \ldots, b_j$ for increasing j until $j = k$) then specifying the unassigned coordinates to be zero and evaluating f_C on those values provides a lower bound on $f_C(\mathbf{a}; \mathbf{b})$ for all later extensions. By specifying an integer N to be the maximum integer we wish to represent, we can fully determine which integers in $\{1, \ldots, N\}$ are representable as $f_C(\mathbf{a}; \mathbf{b})$. Algorithm 4.1 demonstrates this process.

Algorithm 4.1 Evaluate$_C(N, \mathbf{a}, \mathbf{b}, k, i, j)$

```
if $i < k$ then
    $a_{i+1} \leftarrow 0$
    while $f_C(\mathbf{a}; \mathbf{b}) \leq N$ do
        call Evaluate$_C(N, \mathbf{a}, \mathbf{b}, k, i+1, j)$
        $a_{i+1} \leftarrow a_{i+1} + 1$
    end while
else if $j < k$ then
    $b_{j+1} \leftarrow 0$
    while $f_C(\mathbf{a}; \mathbf{b}) \leq N$ do
        call Evaluate$_C(N, \mathbf{a}, \mathbf{b}, k, i, j+1)$
        $b_{j+1} \leftarrow b_{j+1} + 1$
    end while
else
    Mark $f_C(\mathbf{a}; \mathbf{b})$ as representable
end if
```

The Evaluate$_C(N, \mathbf{a}, \mathbf{b}, k, i, j)$ algorithm was implemented using *TreeSearch*. The **while** loops in Algorithm 4.1 makes a selection of the next parameter to fix, which makes a very natural selection for the labels for *TreeSearch*. Therefore, the labels of a job description can be exactly the values of $a_0, a_1, \ldots, a_k, b_0, b_1, \ldots, b_k$ in order, up to the number of fixed positions. Notice that there is a built-in pruning mechanism in the while loop, where we avoid parameters $\mathbf{a}, \mathbf{b}$ so that $f_C(\mathbf{a}; \mathbf{b}) > N$.

One problem is that marking integers as representable is not something that is immediately parallelizable. There are two ways to approach this issue:

1. Every process keeps a list of numbers found to be representable, these numbers are reported at the end of execution, and the lists are combined by taking

a union.

2. Every process keeps a list of numbers not found to be representable, these numbers are reported at the end of execution, and the lists are combined by taking an intersection.

Both of these approaches suffer some drawbacks. For instance, the list of representable numbers is expected to be large, but it could be that the list of non-representable numbers is large. Thus, simply reporting the list can take a significant amount of communication and storage. Further, performing the merging operation (union or intersection, respectively) is non-trivial for such a large data set.

There is a third option for parallelizing the approach that may be more suitable:

3. Every process is assigned an interval $I_\ell = \{N_{\ell-1}+1, \dots, N_\ell\}$ and reports the list of numbers in this range not found to be representable.

The benefit of this approach is that every process is responsible for the full list of all representable numbers in that interval, and we can find the first interval so that there are missed numbers. Each process is essentially calling Evaluate_C with a different upper bound N, except only marking numbers that are at least $N_{\ell-1}+1$. Thus, if ℓ is minimum so that I_ℓ contains a non-representable number, then the process checking this interval has the sharpest upper bound N_ℓ for the pruning operation while still finding the smallest non-representable numbers.

Selecting which interval I_ℓ to check can be integrated in the job descriptions for *TreeSearch*, but instead was implemented as part of the command line arguments and the intervals were split manually before being sent to parallel computation nodes.

$k \leq 2$	471	499	853	883	929
$k \leq 3$	1,003	1,515	1,771	1,899	1,963
$k \leq 4$	3,586,097	3,814,169	3,833,477	3,840,217	3,845,441
$k \leq 5$	95,731,511	97,882,839	97,949,367	97,978,827	98,205,771

Table 4.2: Smallest odd numbers not found to be representable by parameterized posets with k cover edges.

4.5.1 Results

After evaluating the formulas f_C for configurations C with at most 5 cover edges and for many inputs, we found many parameterized posets that represent a large number of odd integers. Table 4.2 lists the first five odd numbers that were not found to be representable by a parameterized poset with at most k cover edges, where $k \in \{2, 3, 4, 5\}$. The $k \geq 5$ case has a large gap from the third to fourth number due to the use of intervals and not completely searching the space in the time alloted.

These computations provide proof of the following theorems.

Theorem 4.23.

1. *If $n < 471$ is odd, then n is representable and* $\operatorname{cov}(n) \leq 2$.
2. *If $n < 942$ is even, then n is representable and* $\operatorname{cov}(n) \leq 3$.

Theorem 4.24.

1. *If $n < 1003$ is odd, then n is representable and* $\operatorname{cov}(n) \leq 3$.
2. *If $n < 2006$ is even, then n is representable and* $\operatorname{cov}(n) \leq 4$.

Theorem 4.25.

1. *If $n < 3,586,097$ is odd, then n is representable and* $\operatorname{cov}(n) \leq 4$.

2. *If* $n < 7,172,194$ *is even, then* n *is representable and* $\operatorname{cov}(n) \leq 5$.

Remark 4.26. The following theorem is based on a recent run of the algorithm with an upper bound of 50,000,000 where a representation using five cover edges was found for every odd number in range. Another search is being executed with an upper bound of 100,000,000.

Theorem 4.27.

1. *If* $n < 50,000,000$ *is odd, then* n *is representable and* $\operatorname{cov}(n) \leq 5$.
2. *If* $n < 100,000,000$ *is even, then* n *is representable and* $\operatorname{cov}(n) \leq 6$.

These results provide significant evidence that the answer to Linek's question is "yes." In addition, this suggests that the cover numbers $\operatorname{cov}(n)$ grow very slowly. To guide the investigation of this problem, we make the following conjecture.

Conjecture 4.28. *All* $n \geq 5$ *are representable. As* n *grows,* $\operatorname{cov}(n) = O(\log \log n)$.

Chapter 5

Ramsey Theory on the Integers

One of the earliest problems in combinatorics led to the development of Ramsey Theory. The general idea is that *absolute disorder is impossible*. For example, no matter how the edges of K_n are colored using r colors, there will be a monochromatic copy of K_ℓ (for n sufficiently large, given r and ℓ). Another way to consider this problem is that a sufficiently dense graph must contain a copy of K_ℓ.

One of the earliest problems in Ramsey Theory came about in a very number-theoretical setting. The idea is to color the numbers $\{1, 2, 3, \dots, n\}$ while trying to avoid giving certain structures (called *arithmetic progressions*) the same color. This chapter investigates this extremal coloring problem for two generalizations of arithmetic progressions. By considering the structure of these structures, we develop methods of constraint propagation which greatly reduces the time required to compute the extremal functions.

$r \backslash k$	3	4	5	6	7	8
2	9	35	178	1,132	$>$ 3,703	$>$ 11,495
3	27	$>$ 292	$>$ 2,173	$>$ 11,191	$>$ 48,811	$>$ 238,400
4	76	$>$ 1,048	$>$ 17,705	$>$ 91,331	$>$ 420,217	
5	$>$ 170	$>$ 2,254	$>$ 98,740	$>$ 540,025		
6	$>$ 223	$>$ 9,778	$>$ 98,748	$>$ 916,981		

Table 5.1: Known values and bounds for van der Waerden numbers, $W^r(k)$.

5.1 Arithmetic Progressions and van der Waerden Numbers

A *progression* is a set of integers $X = \{x_1 < x_2 < \cdots < x_k\}$. The *length* of the progression is $|X|$.

A progression $X = \{x_1 < x_2 < \cdots < x_k\}$ is a *k-term arithmetic progression* (*k-AP*) if there exists an ℓ with $\ell = x_{i+1} - x_i$ for each $i \in \{1, \ldots, k-1\}$.

An r-coloring of $\{1, \ldots, n\}$ is a function $c : \{1, \ldots, n\} \to \{0, \ldots, r-1\}$. A k-AP $x_1 < x_2 < \cdots < x_k$ is *monochromatic* if all colors $c(x_i)$ are the same. An r-coloring is *k-AP-avoiding* if there does not exist a monochromatic k-AP.

Theorem 5.1 (van der Waerden [140]). *Given integers $r \geq 2$ and $k \geq 3$, there exists an $N_{r,k}$ so that for all $n \geq N_{r,k}$ there is no k-AP-good r-coloring of $\{1, \ldots, n\}$.*

Given integers $r \geq 2$ and $k \geq 3$, the *van der Waerden number $W^r(k)$* is the minimum n so that there does not exist a k-AP-avoiding r-coloring of $\{1, \ldots, n\}$. Equivalently, $W^r(k)$ is one larger than the maximum n so that there exists a k-AP-good r-coloring of $\{1, \ldots, n\}$.

The van der Waerden numbers have been determined exactly for very few parameters r, k. Table 5.1 lists known values and bounds on $W^r(k)$ for small values of r and k. In fact, the value $W^2(5) = 1,132$ was determined recently by Kouril and Paul [75].

5.1.1 Lower Bounds on $W^r(k)$

As with all Ramsey-type problems, lower bounds are easier to prove than upper bounds since lower bounds only require existence.

The current-best lower bound is given as a concrete construction in the special case of $k = p + 1$ where p is a prime.

Theorem 5.2 (Berlekamp [14]). *If p is a prime, $W^2(p+1) \geq p2^p$.*

In general, we have an exponential lower bound. One of the first exponential lower bounds was proven using the Lovász Local Lemma [41] and is given as Theorem 5.3.

Theorem 5.3. *Fix $r \geq 2$. For all $k \geq 3$, $W^r(k) \geq \frac{r^{k-1}}{ek}(1 + o(1))$.*

We will see a proof of this lower bound in conjunction with more general lower bounds in Section 5.4. Szabó [128] proved Theorem 5.4 gives the current-best lower bound for the two-color case.

Theorem 5.4 (Szabó [128]). *Fix $\varepsilon > 0$. For k sufficiently large, $W^2(k) \geq \frac{2^k}{k^\varepsilon}$.*

5.1.2 Upper Bounds on $W^r(k)$

Proving non-existence of a k-AP-good coloring is harder than proving existence, so upper bounds are very difficult. Efforts to prove upper bounds on $W^r(k)$ have led to some of the most famous advancements in combinatorics.

The first use of Szemerédi's Regularity Lemma [132] appeared in the proof of Szemerédi's Theorem (Theorem 5.5), a more general version of van der Waerden's Theorem.

Theorem 5.5 (Szemerédi [130, 131]). *For every* $k \geq 3$ *and* $\varepsilon > 0$, *there exists an* $N(k,\varepsilon)$ *so that if* $N \geq N(k,\varepsilon)$ *and* $S \subseteq \{1,\ldots,n\}$ *with* $|S| \geq \varepsilon N$, *then* S *contains a k-AP.*

Szemerédi's Theorem implies van der Waerden's Theorem, since any r-coloring of $\{1,\ldots,n\}$ contains a color class S with $|S| \geq \frac{1}{r}n$. The infinite version of Szemerédi's Theorem is given as

Corollary 5.6. *If* $S \subseteq \mathbb{N}$ *has positive upper density* ($\limsup_{n\to\infty} \frac{|S\cap\{1,\ldots,n\}|}{n} > 0$)*, then* S *contains arbitrarily long arithmetic sequences.*

Since Szemerédi's Regularity Lemma was used, the bound on $N(k,\varepsilon)$ given is astronomical. However, Gowers [50] provided a new proof of Szemerédi's Theorem using combinatorics and functional analysis while proving a much lower bound on $N(k,\varepsilon)$.

Theorem 5.7 (Gowers [50]). *For* $k \geq 3$ *and* $\varepsilon > 0$, $N(k,\varepsilon) \leq 2^{2^{\varepsilon^{-2^{2^{k+9}}}}}$.

Corollary 5.8. *For* $k \geq 3$ *and* $r \geq 2$, $W^r(k) \leq 2^{2^{r^{2^{2^{k+9}}}}}$.

This tower of height five is the current-best upper bound on $W^r(k)$. While Gower's proof is a significant piece of mathematics, it is far from the upper bound conjectured by Graham:

Conjecture 5.9 (Graham [51]). $W^2(k) \leq 2^{k^2}$.

Szemerédi's Theorem is an important generalization of van der Waerden's Theorem, but it is only a special case of a conjecture of Erdős:

Conjecture 5.10 (Erdős [40]). *Let $S \subseteq \mathbb{N}$ have $\sum_{x \in S} \frac{1}{x}$ diverge. Then, S contains arbitrarily long arithmetic sequences.*

An important special case of this conjecture was proven by Green and Tao [53].

Theorem 5.11 (Green, Tao [53]). *The primes contain arbitrarily long arithmetic sequences.*

We now consider a different generalization of van der Waerden numbers by relaxing conditions on arithmetic progressions.

5.2 Quasi-Arithmetic Progressions

Fix a progression $X = \{x_1 < x_2 < \cdots < x_k\}$. The *low difference* of X is the minimum of consecutive differences: $\ell = \min\{x_{i+1} - x_i : i \in \{1, \ldots, k-1\}\}$. The *intermediate diameters* of X are the values $d_i = (x_{i+1} - x_i) - \ell$. The maximum intermediate diameter, $\max\{d_i : i \in \{1, \ldots, k-1\}\}$, is called the *diameter* of X.

A progression is a (k,d)*-quasi-arithmetic progression* (a (k,d)*-QAD*) if it has k terms and diameter at most d.

An r-coloring of $\{1, \ldots, n\}$ is (k,d)*-QAP-avoiding* if it does not contain a monochromatic (k,d)-QAP. The *quasi-arithmetic progression number* $Q_d^r(k)$ is the minimum n so that every r-coloring of $\{1, \ldots, n\}$ has a monochromatic (k,d)-QAP.

Quasi-arithmetic progressions were defined by Brown, Erdős, and Freedman [24] to generalize a structure that appeared in the original proof of Szemerédi's Theorem. Their main result is the equivalence of Erdős' conjecture (Conjecture 5.10) into a similar statement regarding quasi-arithmetic progressions.

Theorem 5.12 (Brown, Erdős, Freedman [24]). *Fix $d \geq 1$. The following are equivalent:*

1. *Every set $S \subseteq \mathbb{N}$ where $\sum_{x \in S} \frac{1}{x}$ diverges contains a k-AP for all $k \geq 3$.*
2. *Every set $S \subseteq \mathbb{N}$ where $\sum_{x \in S} \frac{1}{x}$ diverges contains a (k,d)-QAP for all $k \geq 3$.*

There are two distinct approaches to studying the numbers $Q_d^r(k)$. The first approach is to fix a small diameter d and attempt to show $Q_d^r(k)$ behaves similar to van der Waerden numbers. Vijay [141] demonstrated the only known exponential lower bound on $Q_d^r(k)$ is given exactly when $d = 1$.

Theorem 5.13 (Vijay [141])**.** $Q_1^2(k) \geq 1.08^k$.

The second approach is to fix a parameter i and let the diameter be $d = k - i$. Landman [79] first found exact values of $Q_{k-i}^2(k)$ for $i \in \{1,2\}$. Jobson, Kézdy, Snevily, and White [70] found many more values of $Q_{k-i}^2(k)$ when $i \leq k/2$.

Theorem 5.14 (Jobson, Kézdy, Snevily, White [70])**.** *Let $k \geq 3$, $i \geq 1$ with $2i \leq k$. If $k = mi + r$ with $0 \leq r < i$, then*

$$Q_{k-i}^2(k) \leq 2ik - 4i + 2r + 1$$

and equality holds when $1 \leq r < i/2$ and $r \leq m + 1$.

Table 5.2 lists all known values and bounds on $Q_{k-i}^2(k)$, including updated bounds computed by methods described in this chapter. Tables 5.3, 5.4, and 5.5 list values and bounds on $Q_{k-i}^r(k)$ for $r \in \{3,4,5\}$ as found by our methods.

The constraint $\ell \leq x_i - x_{i-1} \leq \ell + d$ for (k,d)-QAPs is very localized, but allows for accumulated flexibility when k is large. In the next section, we define a new variant of arithmetic progressions which place a more global constraint on the differences $x_i - x_{i-1}$.

$k\backslash i$	1	2	3	4	5	6	7
3	5	9	9				
4	7	11	19	35			
5	9	17	29	33	178		
6	11	19	27	49	67	1132	
7	13	25	37	65	73	$\underline{\mathbf{127}}$	> 3703
8	15	27	39	51	93	$\underline{\mathbf{119}}$	$\underline{\mathbf{> 262}}$
9	17	33	45	65	115	$\underline{\mathbf{127}}$	$\underline{\mathbf{> 210}}$
10	19	35	55	67	83	$\underline{\mathbf{155}}$	$\underline{\mathbf{> 182}}$
11	21	41	57	$\boxed{75}$	101	$\underline{\mathbf{184}}$	$\underline{\mathbf{> 196}}$
12	23	43	63	83	103	123	$\underline{\mathbf{> 223}}$
13	25	49	73	97	$\boxed{115}$	145	$\underline{\mathbf{> 255}}$
14	27	51	75	99	$\boxed{123}$	147	171
15	29	57	81	$\boxed{107}$	133	$\boxed{\leq 161}$	197
16	31	59	91	115	151	$\boxed{\leq 175}$	199
17	33	65	93	129	153	$\boxed{\leq 189}$	215
18	35	67	99	131	$\boxed{165}$	195	$\boxed{\leq 231}$
19	37	73	109	$\boxed{139}$	$\boxed{173}$	217	$\boxed{\leq 247}$
20	39	75	111	147	183	219	$\boxed{\leq 263}$

Bold underlined values and bounds were found in this work. Values below the jagged line are within the range of Theorem 5.14, while boxed numbers are those where the theorem cannot guarantee equality.

Table 5.2: Values and bounds on $Q^2_{k-i}(k)$.

$k \setminus i$	2	3	4	5
3	17	27		
4	38	64	> 292	
5	103	> 166	> 176	$> 2{,}173$
6	> 138	> 185		

Table 5.3: Values and bounds on $Q^3_{k-i}(k)$

$k \setminus i$	2	3	4	5
3	37	76		
4	> 102	> 128	$> 1{,}048$	
5	> 176	> 272	> 536	$> 17{,}705$
6	> 301	> 402		

Table 5.4: Values and bounds on $Q^4_{k-i}(k)$

$k \setminus i$	2	3	4	5
3	> 80	> 170		
4	> 119	> 165	$> 2{,}254$	
5	> 263	> 553	> 900	$> 98{,}740$
6	> 626			

Table 5.5: Values and bounds on $Q^5_{k-i}(k)$

5.3 Pseudo-Arithmetic Progressions

Recall the definition of quasi-arithmetic progression used the low difference and intermediate diameters of a progression.

A progression $X = \{x_1 < x_2 < \cdots < x_k\}$ is a (k,d)*-pseudo-arithmetic progression* (a (k,d)*-PAP*) if the intermediate diameters sum to at most d: $\sum_{i=1}^{k-1} d_i \leq d$.

An r-coloring of $\{1,\ldots,n\}$ is (k,d)*-PAP-avoiding* if it does not contain a monochromatic (k,d)-PAP. The *pseudo-arithmetic progression number* $P_d^r(k)$ is the minimum n so that every r-coloring of $\{1,\ldots,n\}$ has a monochromatic (k,d)-PAP.

The following inequalities are immediate from the definitions:

$$W^r(k) = Q_0^r(k) \geq Q_1^r(k) \geq Q_2^r(k) \geq Q_3^r(k) \geq \cdots \geq Q_{k-1}^r(k) > r(k-1).$$

$$W^r(k) = P_0^r(k) \geq P_1^r(k) \geq P_2^r(k) \geq P_3^r(k) \geq \cdots \geq P_{k-1}^r(k) > r(k-1).$$

$$P_1^r(k) \geq Q_1^r(k) \geq P_{k-1}^r(k).$$

Table 5.6 lists all known values and bounds on $P_{k-i}^2(k)$, including updated bounds computed by methods described here. Tables 5.7, 5.8, and 5.9 list values and bounds on $P_{k-i}^r(k)$ for $r \in \{3,4,5\}$ as found by our methods.

In Section 5.4, prove that for every fixed $d \geq 0$, the numbers $P_d^r(k)$ have an exponential lower bound that is similar to the lower bound on $W^r(k)$ given in Theorem 5.3.

5.4 Exponential Lower Bounds

Theorem 5.3, an exponential lower bound on $W^r(k)$, was one of the first applications of the Lovász Local Lemma.

$k \setminus i$	1	2	3	4	5	6	7
3	5	9	9				
4	7	11	19	35			
5	9	33	33	39	178		
6	11	27	51	61	99	1132	
7	13	73	73	84	146	> 254	> 3703
8	15	51	99	117	> 200	> 310	> 520
9	17	129	129	> 152	> 288	> 424	> 544
10	19	87	163	> 208	> 334		
11	21	201	> 200	> 260			
12	23	129	> 242	> 282			
13	25	289	> 292	> 302			
14	27	179	> 338	> 352			
15	29	393	> 392	> 398			
16	31	237	> 446	> 454			

Table 5.6: Values and bounds on $P^2_{k-i}(k)$.

$k \setminus i$	2	3	4	5
3	17	27		
4	41	74	> 292	
5	> 178	> 189	> 215	$> 2{,}173$
6	> 217	> 269		

Table 5.7: Values and bounds on $P^3_{k-i}(k)$

$k \setminus i$	2	3	4	5
3	37	76		
4	> 111	> 177	$> 1{,}048$	
5	> 285	> 309	> 651	$> 17{,}705$
6	> 292	> 626		

Table 5.8: Values and bounds on $P^4_{k-i}(k)$

$k \setminus i$	2	3	4	5
3	75	> 170		
4	> 128	> 142	$> 2{,}254$	
5	> 198	> 825	> 1300	$> 98{,}740$
6	$> 1,254$			

Table 5.9: Values and bounds on $P^5_{k-i}(k)$

Lemma 5.15 (Lovász Local Lemma [41]). *Let* $A_1, \ldots, A_m$ *be events so that* $\Pr[A_i] = p$ *and for any event* A_i*, there are at most* d *other events mutually dependent on* A_i*. If* $ep(d+1) < 1$*, then the probability that none of the events* $A_1, \ldots, A_m$ *occur is positive:* $\Pr[\cap_{i=1}^{m} \overline{A_i}] > 0$.

Proof sketch for Theorem 5.3. Randomly color (uniformly and independently) $[N]$ with colors $\{1, \ldots, r\}$. Given a k-arithmetic progression X, the event A_X where X is monochromatic has probability $p = r^{-(k-1)}$. The number of k-APs Y that intersect X in at least one element is bounded above by the number of intersection positions of X (k) times the number of positions in Y that element has, times the number of possible differences ℓ ($\frac{N}{k-1}$). This results in $\Delta = kN(1 - o(1))$. When the inequality $ep(\Delta + 1) < 1$ holds, there exists an r-coloring of $[N]$ which does not have a monochromatic k-AP. It suffices to take any $N < \frac{r^{k-1}}{ek}(1+o(1))$. □

This proof technique fails for bounding quasi-arithmetic numbers $Q_d^2(k)$ when $d \geq 1$. The reason is due to the degree of the dependency digraph when applying the Local Lemma. The following lemmas compare the degree of this digraph in the quasi-arithmetic progression and pseudo-arithmetic progression cases. We then use these lemmas to prove exponential lower bounds using the Local Lemma.

First, we find the dependence degree for (k,d)-QAPs is exponential in k with base $d+1$.

Lemma 5.16. *Consider $N \geq k \geq 3$ and let X be a (k,d)-QAP in $[N]$. Then, there are at most $Nk(d+1)^{k-1}(1+o(1))$ (k,d)-QAPs $Y \subseteq [N]$ that intersect X in at least one point.*

Proof. We shall select four parameters based on X and N that specify a unique (k,d)-QAP $Y \subseteq \mathbb{Z}$, and hence we will (over) count the number of such Y. There are k elements $x \in X$. There are k positions in Y where x could be. If $Y \subseteq [N]$, the low difference of Y is between 1 and $N/(k-1)$. Finally, there are $(d+1)^{k-1}$ ways to select the excess differences $d_2, \ldots, d_k$ so that $0 \leq d_i \leq d$ for each $i \in \{2, \ldots, k\}$. Thus there are at most $Nk(d+1)^{k-1}(1+o(1))$ such (k,d)-QAPs Y. □

In contrast, for fixed d, the dependence degree is polynomial in k.

Lemma 5.17. *Consider $N \geq k \geq 3$ and let X be a (k,d)-PAP in $[N]$. Then, there are at most $\frac{1}{d!}k^{d+1}N$ (k,d)-PAPs $Y \subseteq [N]$ that intersect X in at least one point.*

Proof. We shall select four parameters based on X and N that specify a unique $Y \subseteq \mathbf{Z}$, and hence we will (over) count the number of such Y. There are k elements $x \in X$. There are k positions in Y where x could be. If $Y \subseteq [N]$, the low difference of Y is between 1 and $N/(k-1)$. Finally, there are $\binom{k-1}{d} \leq \frac{1}{d!}(k-1)^d$ ways to select the excess differences $d_1, d_2, \ldots, d_k$ so that $\sum_{i=1}^{k} d_j = d$ (and hence $\sum_{i=2}^{k} d_i \leq d$). Thus there are at most $\frac{1}{d!}Nk^2(k-1)^{d-1} \leq \frac{1}{d!}Nk^{d+1}$ such (k,d)-PAPs Y. □

A simple application of the Lovász Local Lemma proves the following theorems.

Theorem 5.18. *Fix $r \geq 2$ and $d \geq 0$.* $Q_d^r(k) \geq \frac{1}{ek}\left(\frac{r}{d+1}\right)^{k-1}(1+o(1))$.

Theorem 5.19. *Fix $r \geq 2$ and $d \geq 0$.* $P_d^r(k) \geq \frac{d!r^{k-1}}{ek^{d+1}}(1+o(1))$.

Proof sketch. Randomly color (uniformly and independently) $[N]$ with colors $\{1, \ldots, r\}$. The event that a given (k,d) quasi- or pseudo-arithmetic progression is monochromatic has probability $p = r^{-(k-1)}$. The degree Δ of the dependency digraph on these events is bounded above by Lemmas 5.16 and 5.17. When the inequality $ep(\Delta+1) < 1$ holds, there exists an r-coloring of $[N]$ which does not have a monochromatic (k,d) quasi- or pseudo-arithmetic progressions. □

Note that Theorem 5.18 is non-trivial only when the number of colors is larger than the diameter (specifically, $r > d+1$). However, Theorem 5.19 provides an exponential lower bound on $P_d^r(k)$ for all $r \geq 2$ when d is fixed. This is likely not the best lower bound on $P_d^r(k)$.

5.5 PAP Numbers of High Diameter

We shall determine some bounds on $P_{k-i}^2(k)$ for small values of i. The following lemma is one of the simplest constraints on a (k,d)-PAP-avoiding coloring, but is a crucial step to proving upper bounds on $P_{k-i}^2(k)$ for $i \in \{1,2\}$.

Lemma 5.20. *If an r-coloring of $\{1, \ldots, n\}$ is (k,d)-PAP-avoiding, then every set of $k+d$ consecutive elements in $\{1, \ldots, n\}$ has at most $k-1$ elements of the same color.*

Proof. If there are k elements of the same color in an interval of length $k+d$, these elements form a (k,d)-PAP with low-difference 1. $\square$

Theorem 5.21. $P^2_{k-1}(k) = 2(k-1)+1$

Proof. Note that any assignment of two colors so that each color class has order at most $k-1$ will not contain any monochromatic progressions of length k, let alone monochromatic $(k,k-1)$-PAPs, so the coloring given by the block representation $0^{k-1}1^{k-1}$ shows $P^2_{k-1}(k) > 2(k-1)$. However, any assignment of two colors to $\{1,\ldots,2(k-1)+1\}$ must have at least one color class of order at least k and by Lemma 5.20 this contains a monochromatic $(k,k-1)$-PAP. $\square$

Theorem 5.22. *For $k \geq 3$, $P^2_{k-2}(k) \leq 2(k-1)^2+1$.*

Proof. Let $n = 2(k-1)^2+1$. We shall prove that every 2-coloring of $\{1,\ldots,n\}$ must contain a monochromatic $(k,k-2)$-PAP.

Fix an arbitrary 2-coloring of $\{1,\ldots,n\}$. By Lemma 5.20, every set $F_j = \{j, j+1, \ldots, j+k+d-1\}$ of $k+d$ consecutive elements has at most $k-1$ elements in each color class. Since $k+d = 2(k-1)$, there are exactly $k-1$ elements in each color class within F_j for each $j \in \{1,\ldots,n-2(k-1)+1\}$. Also note that the frames F_j and F_{j+1} intersect in $2(k-1)-1$ positions, so the color at position j and position $j+2(k-1)$ must be the same. Thus, the set $X = \{1+2(k-1)t : t \in \{0,\ldots,k-1\}\}$ is a monochromatic k-AP in this coloring, a contradiction. $\square$

We now show some lower bounds for $P^2_{k-i}(k)$ when $i \in \{2,3\}$.

Proposition 5.23. *For k odd, $P^2_{k-2}(k) > 2(k-1)^2$. For all k, $P^2_{k-3}(k) > 2(k-1)^2$.*

Proof. We will show that the 2-coloring of $\{1,\ldots,2(k-1)^2\}$ given by alternating blocks of size $k-1$ avoids $(k,k-2)$-PAPs when k is odd and avoids $(k,k-3)$-PAPs when k is even. (Note that since $P^2_{k-3}(k) \geq P^2_{k-2}(k)$, this suffices to show

$P^2_{k-3}(k) > 2(k-1)^2$ for all k.) The block representation of this coloring is

$$\left(0^{k-1}1^{k-1}\right)^{k-1}.$$

Let $B_1, B_2, \ldots, B_{k-1}$ be the blocks of color zero.

We proceed by contradiction. So that we may discuss both cases simultaneously, let d be any sum diameter. Assume $X = \{x_1 < x_2 < \cdots < x_k\}$ is a monochromatic progression of length k within this coloring. Without loss of generality, all elements of X have color zero. Let ℓ be the low-difference of X and suppose that the sum diameter is at most d.

Let h be the number of blocks B_a so that $X \cap B_a \neq \emptyset$. Each block has $k-1$ elements, so X is not contained within any block. Further, there are only $k-1$ blocks, so there are two consecutive elements x_j, x_{j+1} within the same block. Thus, $\ell \leq k-2$.

Each consecutive pair x_j, x_{j+1} in different blocks has $x_{j+1} - x_j \geq k$, so these pairs contribute at least $k-\ell$ to the sum diameter of X. Since there are $h-1$ such pairs, we have $(h-1)(k-\ell) \leq d$. Therefore, $h \leq \left\lfloor 1 + \frac{d}{k-\ell} \right\rfloor$.

Since consecutive differences in X are at least ℓ, the number of elements within X and any block B_a is at most

$$|X \cap B_a| \leq \left\lceil \frac{k-1}{\ell} \right\rceil,$$

which implies $k \leq h \left\lceil \frac{k-1}{\ell} \right\rceil$.

Finally, we have

$$k \leq \left\lceil \frac{k-1}{\ell} \right\rceil h \leq \left\lceil \frac{k-1}{\ell} \right\rceil \cdot \left\lfloor 1 + \frac{d}{k-\ell} \right\rfloor. \tag{5.1}$$

We now split into cases for k odd or k even and use Equation 5.1 to complete the proof.

Case 1: k is odd and $d = k-2$. If $\ell = 1$, the inequality $h\left[1+\frac{k-2}{k-1}\right]$ implies that $h = 1$, but X spans at least two blocks, a contradiction.

If $\ell = 2$, then $\left\lceil \frac{k-1}{\ell} \right\rceil = \frac{k-1}{2}$. Hence, Equation 5.1 yields $k \leq \frac{k-1}{2}\left[1+\frac{k-2}{k-2}\right] = k-1$, a contradiction.

If $\ell \geq 3$, let $k = p\ell + q$, where $0 \leq q < \ell$. Note that $\left\lceil \frac{k-1}{\ell} \right\rceil \in \{p-1, p\}$, so Equation 5.1 implies

$$p\ell \leq k \leq p\left[1+\frac{k-2}{k-\ell}\right],$$

which reduces to $\ell \leq 1 + \frac{k-2}{k-\ell}$. With some algebra, observe that this implies $k-1 \leq \ell$. This contradicts that $\ell \leq k-2$.

Case 2: k is even and $d = k-3$. The cases $\ell = 1$ and $\ell \geq 3$ follow from Case 1. If $\ell = 2$, then $\left\lceil \frac{k-1}{\ell} \right\rceil = \frac{k}{2}$. Hence, Equation 5.1 yields

$$k \leq \frac{k}{2}\left[1+\frac{k-3}{k-2}\right] = \frac{k}{2}\left[\frac{2(k-2)-1}{k-2}\right] < k,$$

a contradiction. □

Combining Theorem 5.22 and Proposition 5.23 gives the exact value for $P^2_{k-2}(k)$ when k is odd.

Corollary 5.24. *For $k \geq 3$ odd, $P^2_{k-2}(k) = 2(k-1)^2 + 1$.*

Notice from the proof of Proposition 5.23 that the construction has a monochromatic $(k, k-2)$-PAP X in the case that k was even and the low difference of X is $\ell = 2$, since this difference is enough to pick up $k/2$ elements from two blocks and the sum-diameter is large enough to span between those two blocks.

This ends the current knowledge on tight bounds for PAP numbers. Thus, to learn more, we resort to computational methods.

5.6 Search Algorithms

In this section we discuss algorithms to exhaustively search for colorings of $[n]$ which avoid monochromatic quasi- and pseudo-arithmetic progressions with given length (k) and diameter (d). The general strategy is a standard backtracking search with varying levels of constraint propagation. We shall use tabulation to quickly determine when monochromatic progressions appear and backtrack in those situations.

5.6.1 Coloring $[n]$ While Avoiding (k, d)-QAPs

We begin with the (k, d)-QAP case and will later adapt the techniques to (k, d)-PAPs. Define functions $b : [r] \times [n] \times [\lceil \frac{n}{k-1} \rceil] \to \{0, \ldots, k\}$ and $f : [n] \times [\lceil \frac{n}{k-1} \rceil] \to \{0, \ldots, k\}$ to be the *backward* and *forward* tables for a partial r-coloring c if the following conditions hold for all colors $a \in [r]$ and pairs $j, \ell \in [n] \times [\lceil \frac{n}{k-1} \rceil]$:

1. If $t = b(a, j, \ell)$, then there is a (t, d)-QAP with low difference ℓ that *ends* at j with the first $t - 1$ terms having color a *and* there is no $(t + 1, d)$-QAP with low difference ℓ that *ends* at j with the first t terms having color a.
2. If $t = f(a, j, \ell)$, then there is a (t, d)-QAP with low difference ℓ that *begins* at j with the last $t - 1$ terms having color a *and* there is no $(t + 1, d)$-QAP with low difference ℓ that *begins* at j with the last t terms having color a.

When a color is assigned to a position j, the tables are updated by advancing from position j with each low difference ℓ and allowed diameter $d' \leq d$. That is, when $c(j)$ is assigned to be a, the backward table value $b(a, j + (\ell + d'), \ell)$ is assigned to be at least $b(a, j, \ell) + 1$. To update the forward table, the value $f(a, j - (\ell + d'), \ell)$ is assigned to be at least $f(a, j, \ell) + 1$. If the colors are assigned in increasing order, the backward tables advance at most one position at a time, as the color $c(j + \ell + d')$ will always be unset. However, updating the forward table triggers a cascading effect as long as the color $c(j - (\ell + d'))$ agrees with the color $c(j)$. Hence, the forward table is not updated unless the forward table is necessary for the constraint procedure. This procedure is discussed in the following section.

Using just the backward table, we have a tagulation approach that is very efficient to update and is capable of exhaustively searching for all (k, d)-QAP-avoiding 2-colorings of $[n]$. What this procedure lacks is a "lookahead" mechanism to determine that a given partial coloring of $[n]$ cannot extend to a full (k, d)-QAP-avoiding coloring. In order to detect such a situation, colors must be assigned until the backward table provides a contradiction.

We now define two levels of constraint propagation that provide the capability to backtrack the search by assigning colors that *must* be present in *any* (k, d)-QAP-avoiding extension of the current coloring. These techniques increase the computation cost per-node, but in some cases sufficiently decrease the number of generated colorings so that the method is much more efficient than the non-propagating search.

5.6.2 Constraint Propagation

The essential idea of our constraint propagation is to remove potential assignments of the color $c(j)$ if assigning that color to $c(j)$ would *immediately* create a monochromatic (k, d)-QAP. We set $D(j)$ to be the *domain* of j, or the set of potential colors for the position j. These sets are all initialized to $D(j) = \{1, \ldots, r\}$. Our two levels of propagation use different amounts of computation to determine when such an event occurs.

Backward Propagation uses only the backward table. If there is a color a, position j, and low-difference ℓ so that $b(a, j, \ell) = k - 1$, then assigning $c(j) = a$ would color the final point in a (k, d)-QAP the same as the previous $k - 1$ positions. Hence, we remove the color a from $D(j)$. If only one color remains in $D(j)$, then that color must be assigned to $c(j)$. This requires the backward table to be updated and may remove more colors from the domains of other positions, leading to other propagations.

Forward/Backward Propagation uses the forward and backward tables in conjunction to see when a position cannot be assigned a given color. If there is a color a, position j, and low-difference ℓ so that $f(a, j, \ell) + b(a, j, \ell) = k - 1$, then j is somewhere within a (k, d)-QAP which has $k - 1$ positions colored a. Hence, $c(j)$ cannot be assigned a or it would make this QAP be monochromatic.

5.6.3 Coloring $[n]$ While Avoiding (k, d)-PAPs

Similar to the previous section, we aim to search for r-colorings of $[n]$ that avoid monochromatic-(k, d)-PAPs. The algorithms are essentially the same, except the backward/forward tables are four-dimensional. The parameters a, j, ℓ, d' specify a color, a position, a low-difference, and an upper bound on the difference sum. Here, d' ranges from 0 to d, inclusive.

1. If $t = b(a, j, \ell, d')$, then there is a (t, d')-PAP with low difference ℓ that *ends* at j with the first $t - 1$ terms having color a *and* there is no $(t + 1, d')$-QAP with low difference ℓ that *ends* at j with the first t terms having color a.

2. If $t = f(a, j, \ell, d')$, then there is a (t, d')-QAP with low difference ℓ that *begins* at j with the last $t-1$ terms having color a *and* there is no $(t+1, d')$-QAP with low difference ℓ that *begins* at j with the last t terms having color a.

When updating the backward and forward tables, the choices for the next step vary with the difference $d - d'$. This difference $d - d'$ provides the amount of flexibility remaining to keep the difference sum at most d.

The propagation rules are similar, however the forward/backward rule requires special care. Here, we need to check if $f(a, j, \ell, d') + b(a, j, \ell, d'') \geq k - 1$ for any values of $d', d'' \in \{0, \ldots, d\}$ so that $d' + d'' \leq d$, since the difference sum of the forward progression and the backward progression must combine to be at most d.

5.6.4 Conditions and Implications for Propagation

To review the computational model for the three levels of constraint propagation, Tables 5.10–5.17 list the conditions that trigger an implication during the search. When a condition occurs, the implication is performed. After all implications are performed, the search attempts to assign the color $c(j)$ for the first position j that has no assigned color. If the implication ever assigns $D(j) = \emptyset$, then the current partial coloring of $[n]$ has no extension which avoids a (k, d)-*AP.

*AP : QAP/PAP	Condition : $\exists j, \lvert D(j)\rvert \equiv 1$
$\emptyset$/F/B : $\emptyset$, B, & F/B	Implication : $a \in D(j); c(j) \leftarrow a$.

Table 5.10: Color-Assignment Rule

*AP : QAP	Condition : $\exists a, j, c(j) \leftarrow a$
$\emptyset$/F/B : B, & F/B	Implication : $\forall \ell \in \{1, \ldots, L\}, d' \in \{0, \ldots, d\},$ $b(a, j + \ell + d', \ell) \overset{\max}{\longleftarrow} b(a, j, \ell) + 1.$

Table 5.11: QAP Backward Table Update

*AP : QAP	Condition : $\exists a, j, c(j) \leftarrow a$
$\emptyset$/F/B : F/B	Implication : $\forall \ell \in \{1, \ldots, L\}, d' \in \{0, \ldots, d\},$ $f(a, j - (\ell + d'), \ell) \overset{\max}{\longleftarrow} f(a, j, \ell) + 1.$

Table 5.12: QAP Forward Table Update

*AP : PAP	Condition : $\exists a, j, \ell, d', b(a, j, \ell, d') \geq k - 1$
$\emptyset$/F/B : B	Implication : $D(j) \leftarrow D(j) \setminus \{a\}.$

Table 5.13: Backward Domain Removal Rule

*AP : PAP	Condition : $\exists a, j, c(j) \leftarrow a$
$\emptyset$/F/B : B, & F/B	Implication : $\forall \ell \in \{1, \ldots, L\}, d' + d'' \in \{0, \ldots, d\},$ $b(a, j + \ell + d'', \ell, d' + d'') \overset{\max}{\longleftarrow} b(a, j, \ell, d') + 1.$

Table 5.14: QAP Backward Table Update

*AP : PAP	Condition : $\exists a, j, c(j) \leftarrow a$
$\emptyset$/F/B : F/B	Implication : $\forall \ell \in \{1, \ldots, L\}, d' + d'' \in \{0, \ldots, d\},$ $f(a, j - (\ell + d''), \ell, d' + d'') \overset{\max}{\longleftarrow} f(a, j, \ell, d') + 1.$

Table 5.15: QAP Forward Table Update

*AP : PAP	Condition : $\exists a, j, \ell, d', b(a, j, \ell, d') \geq k - 1$
$\emptyset$/F/B : B	Implication : $D(j) \leftarrow D(j) \setminus \{a\}.$

Table 5.16: Backward Domain Removal Rule

*AP : PAP	Condition : $\exists a, j, \ell, d', f(a,j,\ell,d') + b(a,j,\ell,d-d') \geq k-1$
$\emptyset$/F/B : F/B	Implication : $D(j) \leftarrow D(j) \setminus \{a\}$.

Table 5.17: Forward/Backward Domain Removal Rule

5.7 Skew-Symmetric Colorings

A coloring $c : \{1, \dots, n\} \to \{0, 1\}$ is *skew-symmetric* if for all $i \in \{1, \dots, n\}$, $c(i) = 1 - c(n-i+1)$. Skew-symmetric colorings are invariant under the action of reversing the coloring and flipping the colors. Note that skew-symmetric colorings are only possible when n is an even number, as when n is odd the number $\frac{n+1}{2}$ is invariant under the reversal so flipping the colors creates a different coloring. Many of the extremal (k, d)-QAP-avoiding colorings were skew-symmetric or appeared very close to skew-symmetric. This led to the question of how large are the Ramsey numbers for avoiding monochromatic (k, d)-QAPs or (k, d)-PAPs over all skew-symmetric colorings.

Definition 5.25. Fix $k \geq 3$ and $d \geq 0$.

1. Let $Q_d^{\mathrm{ss}}(k) = n + 1$ where n is the largest even number so that there is a skew-symmetric 2-coloring of $\{1, \dots, n\}$ that has no monochromatic (k, d)-QAP.
2. Let $P_d^{\mathrm{ss}}(k) = n + 1$ where n is the largest even number so that there is a skew-symmetric 2-coloring of $\{1, \dots, n\}$ that has no monochromatic (k, d)-PAP.

While finding the numbers $Q_d^{\mathrm{ss}}(k)$ and $P_d^{\mathrm{ss}}(k)$ have independent interest, the real focus of computing these numbers is to extend the lower bounds on $Q_d^2(k)$ and $P_d^2(k)$ by searching exhaustively over a smaller search space. While there are 2^n possible 2-colorings of $\{1, \dots, n\}$, there are $2^{n/2}$ possible skew-symmetric colorings. Thus, we can cover the entire space of skew-symmetric colorings more quickly than we can cover all colorings. This leads to new lower bounds that our previous search did not find.

To search for skew-symmetric colorings, we change our base set by a translation. A coloring $c : \{-(n-1), -(n-2), \ldots, -1, 0, 1, \ldots, n\} \to \{0, 1\}$ is *skew-symmetric* if $c(i) = 1 - c(1-i)$. To search over skew-symmetric colorings, we can assign colors to the numbers $1, 2, 3, \ldots$ and let the colors for $0, -1, -2, \ldots$ be implied by the skew-symmetric constraint. Then, the forward and backward tables have the same properties as before, but over the range of colored positions.

5.8 Discussion

In this chapter, we investigated quasi-arithmetic progressions and defined pseudo-arithmetic progressions in an attempt to better understand what makes van der Waerden numbers so difficult to find. By developing a computational search with constraint propagation, we exhaustively searched for extremal r-colorings that avoid these progressions. In many cases (especially the case when $r > 2$), we could not find the exact value in a reasonable amount of time and could only resort to lower bounds given by constructions.

When lower bounds seem to be the only achievable results, local search is a powerful tool to extend the lower bounds. Local search techniques sacrifice completeness (i.e. the search cannot determine nonexistence) in favor of a high probability of finding large solutions. Therefore, to further investigate the numbers $Q_d^r(k)$ and $P_d^r(k)$, local search techniques should be employed. This is left as future work.

In the case of two colors ($r = 2$), we have several exact values of $P_{k-i}^2(k)$ for small k and i. From these data points, we can form conjectures for the behavior

k	$i = 2$		$i = 3$		$i = 4$		$i = 5$		$i = 6$		**Key**	
3	9	9	9								$Q^2_{k-i}(k)$	$P^2_{k-i}(k)$
	9	9	9								$Q^{ss}_{k-i}(k)$	$P^{ss}_{k-i}(k)$
4	11	11	19	19	35							
	11	10	19	19	35							
5	17	33	29	33	33	39	178					
	17	33	29	33	33	39	177					
6	19	27	27	51	49	61	67	99	1132			
	19	27	27	51	49	59	67	99	1131			
7	25	73	37	73	65	84	73	146	127	> 242	> 3703	
	25	73	37	73	65	83	73	135	123	**255**		
8	27	51	39	99	51	117	93	> 200	119	> 294	> 256	> 452
	27	51	39	99	51	109	91	183	119	**311**	> 262	**> 520**
9	33	129	45	129	65	> 152	115	> 269	127	> 385	> 210	> 540
	33	129	45	129	65	155	115	**289**	127	**> 424**	> 204	**> 544**
10	35	87	55	163	67	> 184	83	> 324	155		> 177	
	35	87	55	163	67	209	83	**> 334**	> 152	**> 492**	> 182	
11	41	201	57	≥ 201	75		101		184		> 187	
	41	201	57	201	73	> 260	101		> 182		**> 196**	
12	43	129	63	≥ 243	83		103		123		> 223	
	43	129	63	243	83	> 282	103		123			
13	49	289	73	≥ 289	97		115		145		> 255	
	49	≥ 289	73	**> 292**	97	> 302	115		145			
14	51	179	75		99		123		147		171	
	51	179	75	> 338	99	> 352	123		147			
15	57	393	81	≥ 393	107		133		161		197	
	57	393	81	≥ 393	105	> 398	133		161			
16	59	237	91		115		151				215	
	59	237	91	> 446	115	> 454	151		169			
17	65		93		129		153				215	
	65		93		129		153		> 180			
18	67		99		131		165		195			
	67		99		131		165		195			
19	73		109		139		173		217			
	73		109		137		173		217			
20	75		111		147		183		219			
	75		111		147		183		219			

Bold and underlined values are places where the skew-symmetric search found a larger coloring than the standard method.

Table 5.18: Values.

of these numbers. However, any conjectures based on these numbers may be subject to error due to only considering small values. For instance, looking at small odd values of k we may guess that $P^2_{k-2}(k) = P^2_{k-3}(k)$. However, from the skew-symmetric search, we found a 2-coloring of order 292 that avoids $(13, 10)$-PAPs, so $P^2_{13-3}(13) > 292$ while $P^2_{13-2}(13) = 289$. Thus, without finding more exact values of $P^2_{k-3}(k)$, it is unlikely to find a correct conjecture for the values as k increases indefinitely.

Part II

Isomorph-Free Generation

Chapter 6

Canonical Deletion

This chapter provides an overview of McKay's isomorph-free generation technique [92], commonly called *canonical deletion*. The technique guarantees that every unlabeled object of a desired property will be visited exactly once. The word *visited* means that a labeled object in the isomorphism class is generated and tested for the given property and possibly used to extend to other objects. Also, objects that do not satisfy the property are visited *at most* once because we may be able to use pruning to avoid generating objects that do not lead to solutions.

The canonical deletion technique is so called from its use of reversing the augmentation process in order to guarantee there is exactly one path of augmentations from a base object to every unlabeled object. Essentially, a deletion function is defined that selects a part of the combinatorial object to remove, and this function is invariant up to isomorphism. Then, when augmenting an object this augmentation is compared to the canonical deletion of the resulting object to check if this is the "correct" way to build the larger object. If not, the augmentation is rejected and the larger object is not visited.

By following the canonical deletion from any unlabeled object, we can reconstruct the unique sequence of augmentations that leads from a base object to that unlabeled object. Further, this process is entirely *local*: it depends only on the current object and current position within the search tree. This allows the process to be effective even when parallelizing across computation nodes without needing any communication between nodes, unlike some other isomorph-free generation methods which require keeping a list of previously visited objects.

We begin this chapter by reviewing augmentations and deletions in Section 6.1. Then, Section 6.2 discusses how to reduce augmentations by orbit calculations. In Section 6.3, we present a useful tool called a *canonical labeling* that allows the canonical deletion to be computed up to isomorphism, even though we are starting with an arbitrary labeled object. Finally, we describe canonical deletions and the full search process in Section 6.4. Some tips for optimizing this general process are presented in Section 6.5.

6.1 Objects, Augmentations, and Deletions

Suppose we are searching for combinatorial objects from a family $\mathcal{L}$ of *labeled objects*. Under the appropriate definition of isomorphism for those objects, let $\cong$ be the isomorphism relation and $\mathcal{U}$ be the family of *unlabeled objects*: the equivalence classes under $\cong$. Let $P : \mathcal{L} \to \{0,1\}$ be a *property*, and we wish to generate all objects X in $\mathcal{L}$ where $P(X) = 1$. We shall assume the property P is *invariant* under isomorphism ($\cong$): for all unlabeled objects $\mathcal{X} \in \mathcal{U}$ and labeled objects $X, X' \in \mathcal{X}$, $P(X) = P(X')$. In this case, we can define $P(\mathcal{X})$ for an unlabeled object $\mathcal{X}$ to be equal to $P(X)$ for any labeled object $X \in \mathcal{X}$.

Example. Let $\mathcal{L}$ be the set of graphs of order n. Then $\mathcal{U}$ is the family of unlabeled graphs where the standard relation of isomorphism ($\cong$) is used between graphs.

The property P could be $P(G) = 1$ if and only if G is 4-regular and G has chromatic number three. One nice aspect of the property P is that all induced subgraphs of 4-regular 3-chromatic graphs have maximum degree at most 4 and chromatic number at most 3.

We require a set $B \subset \mathcal{L}$ of *base objects* and an *augmentation*. Let $B \subset \mathcal{L}$ be a set of labeled objects where every pair $X, Y \in B$ has $X \ncong Y$.

For a labeled object $X \in \mathcal{L}$, the augmentation defines a set $\mathcal{A}(X)$ of *augmentations*. An object in $\mathcal{A}(X)$ should specify enough information to determine *how* to augment X to create a new object Y. Let $\mathcal{D}(X)$ be the set of *deletions*, which specify the information to determine *how* to delete something from X to create a smaller object Z.

There must be a bijection $\delta : \cup_{X \in \mathcal{L}} \mathcal{A}(X) \to \cup_{Y \in \mathcal{L}} \mathcal{D}(Y)$ from augmentations to deletions. In some sense, this bijection should be *natural*, in that an augmentation $A \in \mathcal{A}(X)$ maps to $\delta(A) = D \in \mathcal{D}(Y)$ if and only if performing the augmentation A on X creates the object Y and performing the deletion D on Y results in X.

We shall consider our objects as being built *up*, so the set of augmented objects $\mathcal{A}(X)$ can be called the *above* objects while the deleted objects $\mathcal{D}(X)$ are the *downward* objects[1]. Figure 6.1 provides a visualization of these sets with respect to a labeled object X.

Example. Suppose we wish to enumerate all connected graphs of order n. One augmentation step is given by adding a vertex and specifying its neighborhood. Since every connected graph contains an edge, we can start from K_2 as a base object (so $B = \{K_2\}$).

---

[1] The sets $\mathcal{A}(X)$ and $\mathcal{D}(X)$ were originally called *lower objects* $L(X)$ and *upper objects* $U(X)$ by McKay [92]. In addition to the problem that the letters $\mathcal{L}$ and $\mathcal{U}$ are already used for *labeled* and *unlabeled* objects, the notation for $L(X)$ and $U(X)$ is confusing because McKay never explicitly states which direction is "up"!

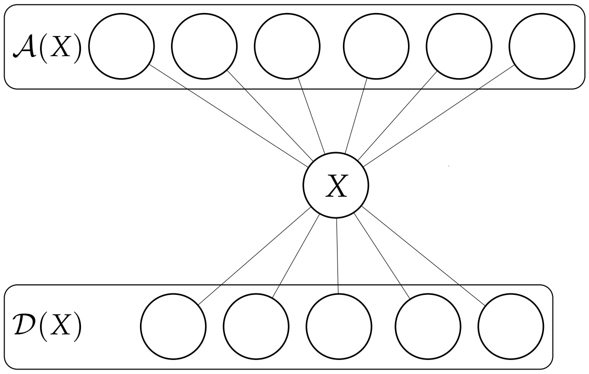


Figure 6.1: Augmentations $\mathcal{A}(X)$ and deletions $\mathcal{D}(X)$.

Every vertex augmentation is given by specifying the neighborhood of the new vertex in the graph, so every augmentation is a pair (G, S) with $S \subseteq V(G)$. To guarantee connectedness, we can assume that $S \neq \emptyset$. To delete a vertex, we must only specify the vertex, so we can let pairs (G, v) with $v \in V(G)$ be a deletion leading to the graph $G - v$. From this, let

$$\mathcal{A}(G) = \{(G, S) : S \subseteq V(G), S \neq \emptyset\}, \qquad \mathcal{D}(G) = \{(G, v) : v \in V(G)\}.$$

The natural bijection δ can be defined as mapping a graph, subset pair (G, S) to the graph, vertex pair (H, v) where the graph H has vertex set $V(G) \cup \{v\}$ and H has edges given by

$$E(H) = E(G) \cup \{uv : u \in S\}.$$

Figure 6.2 shows an example graph G of order three with all deletions and augmentations specified. The augmentations have white circles representing the vertices in S while the deletions have red circles representing the vertex v to delete. Further, the deletions, $(G, v) \in \mathcal{D}(G)$, are connected[2] to the augmentations of the

---

[2] These connections are based on the action of the bijection δ or its inverse.

two graphs of order two and the augmentations, $(G, S) \in \mathcal{A}(G)$, are connected to the deletions of the graphs of order four.

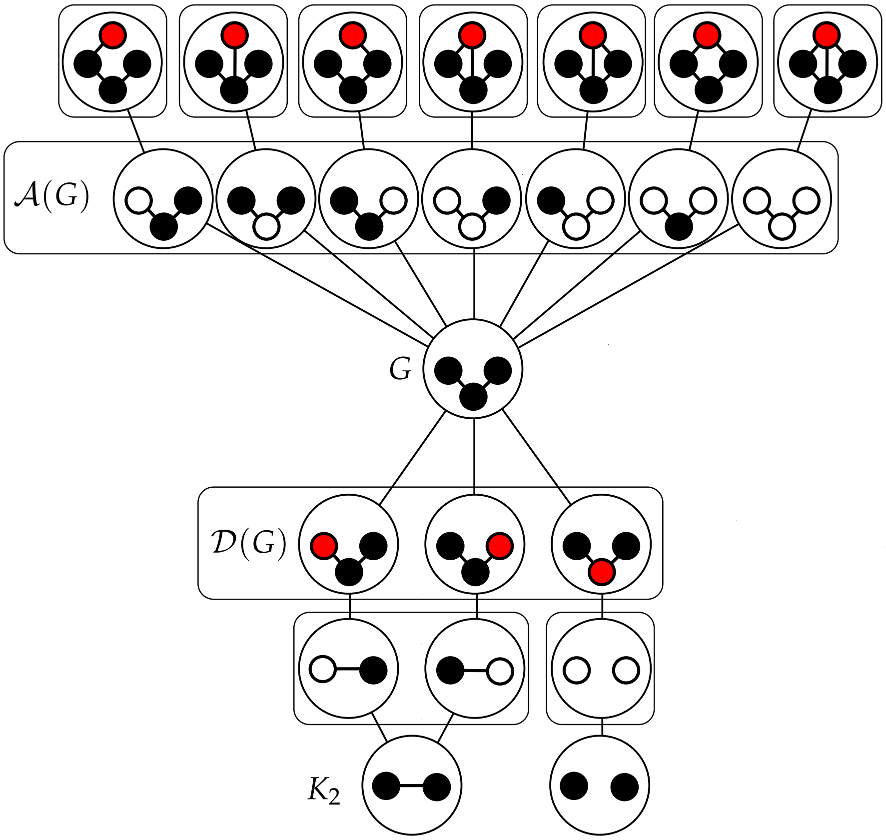


Figure 6.2: Vertex augmentations $\mathcal{A}(G)$ and deletions $\mathcal{D}(G)$ for a graph G.

These definitions of augmentations and deletions are based on labeled objects. In order to generate at most one labeled representative of every unlabeled object, we must consider what isomorphism means in this context.

6.2 Augmentations and Orbits

For isomorphism concerns, we shall assume that for an unlabeled object $\mathcal{X} \in \mathcal{U}$, any two labeled objects $X, X' \in \mathcal{X}$ have a bijection $\pi_{X,X'} : \mathcal{A}(X) \to \mathcal{A}(X')$ and

$\sigma_{X,X'} : \mathcal{D}(X) \to \mathcal{D}(X')$ so that the following statements hold:

1. For all $Y \in \mathcal{A}(X)$, the object W so that $\delta(Y) \in \mathcal{D}(W)$ and object W' so that $\delta(\pi_{X,X'}(Y)) \in \mathcal{D}(W')$ are isomorphic ($W \cong W'$).
2. For all $Z \in \mathcal{D}(X)$, the object W so that $\delta^{-1}(Z) \in \mathcal{A}(W)$ and the object W' so that $\delta^{-1}(\sigma_{X,X'}(Z)) \in \mathcal{A}(W)$ are isomorphic ($W \cong W'$).

This allows us to define the augmented and deleted objects $\mathcal{A}(\mathcal{X})$ and $\mathcal{D}(\mathcal{X})$ for unlabeled objects $\mathcal{X} \in \mathcal{U}$ as well.

Example. If two graphs, G and H, are isomorphic via a bijection $\tau : V(G) \to V(H)$, then an augmentation $(G, S) \in \mathcal{A}(G)$ maps to (H, S') where $S' = \{\tau(x) : x \in S\}$. Further, a deletion (G, v) maps directly to $(H, \tau(v))$. Therefore, any two isomorphic graphs G and H have a bijection $\pi_{G,H} : \mathcal{A}(G) \to \mathcal{A}(H)$.

Now, we wish to remove isomorphic duplicates, so the most natural first step is to remove duplicate augmentations. That is, if we are augmenting an object X via two augmentations $A, A' \in \mathcal{A}(X)$, we should make sure that A is not isomorphic to A' or else $\delta(A)$ and $\delta(A')$ will be isomorphic. These types of decisions can be made using the automorphism group of the object X.

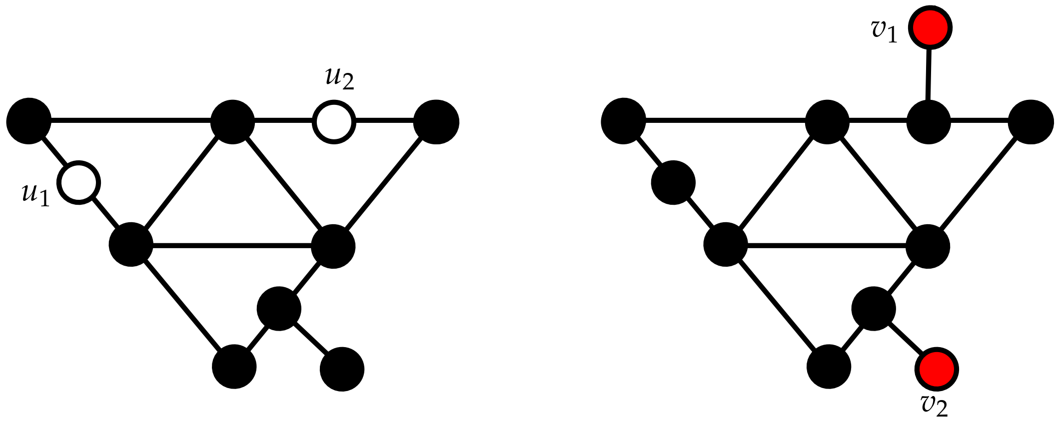
Example. When augmenting a graph G by adding a vertex, if there is an automorphism $\sigma : V(G) \to V(G)$ so that σ maps a set $S \subseteq V(G)$ to a set S', then the augmentations (G, S) and (G, S') create isomorphic graphs H and H'. Therefore, we should compute all set orbits and select exactly one representative from each orbit[3].

---

[3] Since this calculation seems inefficient when the sets can have arbitrary size (i.e. $\Delta(G)$ is not bounded), McKay's paper [92] has a workaround to avoid duplications without sacrificing efficiency.

It is an unfortunate fact that these local orbit calculations are not enough to guarantee that we shall not create duplicate objects. There are several reasons, including:

1. *Augmentations that are not in orbit can create isomorphic objects.* See Figure 6.3(a), where two single-vertex neighborhoods are not in orbit but augmenting by either creates the same unlabeled graph. Figure 6.3(b) shows this ambiguity in the reverse direction, where two deletions from different orbits result in the same graph.



(a) Augmenting G by neighborhoods $\{u_1\}$ or $\{u_2\}$ result in isomorphic graphs, but u_1 and u_2 are in different orbits.

(b) Deleting H by vertices v_1 or v_2 results in isomorphic graphs, but v_1 and v_2 are in different orbits.

Figure 6.3: Different orbits do not imply different augmentations and deletions.

2. *There may be two internally disjoint sequences of non-isomorphic augmentations which generate the same unlabeled object.* Figure 6.4 shows two different sequences of three vertex augmentations starting from K_2 that generate the same unlabeled object. What is most important is that these paths are internally disjoint with respect to unlabeled objects. Thus, no amount of verifying

that isomorphic duplicates are avoided at the next level (or some constant number of levels), eventually some objects will be repeated.

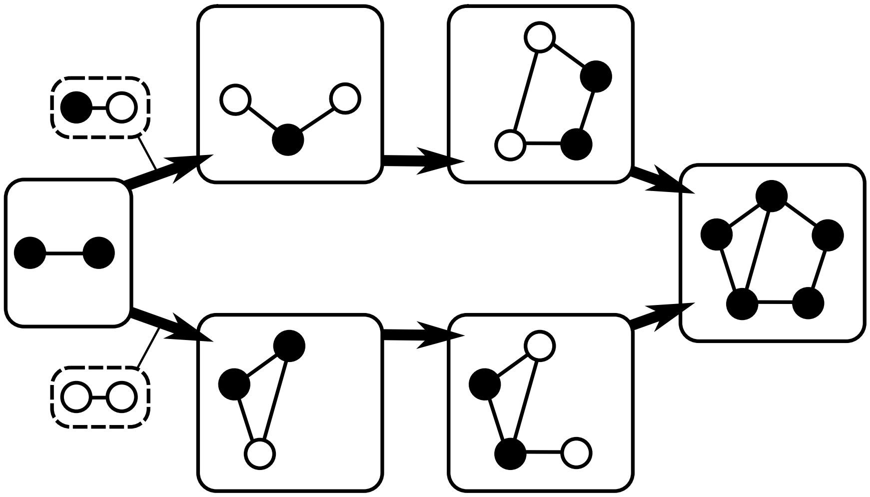


Figure 6.4: Two internally-disjoint augmentation paths leading to the same unlabeled object.

These concerns demonstrate that a "bottom up" approach to avoiding duplicate graphs is not satisfactory. The canonical deletion technique avoids this issue by reversing the process: it makes sure that every unlabeled graph has a unique sequence of deletions that leads to a base object.

6.3 Canonical Labelings

In order to avoid the previously mentioned pitfalls that lead to multiple labeled representatives of an isomorphism class being visited in the search, we must utilize a powerful tool that allows us to compute invariants.

Definition 6.1. For a family of labeled objects $\mathcal{L}$ whose unlabeled objects are $\mathcal{U}$, a *canonical labeling* is a function $\ell : \mathcal{L} \to \mathcal{L}$ where for every unlabeled object $U \in \mathcal{U}$

and every pair of labeled objects $L, L' \in \mathcal{U}$, the two labeled objects $\ell(L)$ and $\ell(L')$ are equal.

In this sense, the labeling function ℓ will *relabel* an object L to create another labeled object $\ell(L)$ of the same isomorphism class. The important behavior is that $\ell(L)$ is the same labeled object for the entire isomorphism class, so the object $\ell(L)$ is invariant for unlabeled objects.

Example. When the family of labeled objects $\mathcal{L}$ is a collection of undirected (or directed) graphs, then a canonical labeling $\ell(G)$ can be selected to be the graph $H \cong G$ with lexicographically-least adjacency matrix. This choice of canonical labeling fits the definition, but is difficult to compute. Instead, McKay's *nauty* library [93] was designed to efficiently compute canonical labelings. The algorithm defines the map ℓ, and is not easily described. See the survey paper by Hartke and Radcliffe [61] for a full description of the *nauty* algorithm.

6.4 Canonical Deletions

Finally, we are able to describe the canonical deletion. This deletion is the most fundamental concern to this search technique.

Definition 6.2. A *canonical deletion* function is a map $\text{del} : \mathcal{L} \to \cup_{X \in \mathcal{L}} \mathcal{D}(X)$ so that $\text{del}(X) \in \mathcal{D}(X)$ and for any two isomorphic objects $X \cong X'$ we have isomorphism of their deletions: $\text{del}(X) \cong \text{del}(X')$.

Example. When building graphs by vertex augmentations, a graph G has deletion set $\mathcal{D}(G) = \{(G, v) : v \in V(G)\}$. Thus, we need only select a single vertex up to isomorphism. By computing the canonical labeling $\ell(G)$, we find an ordering of $V(G)$ that is invariant up to isomorphism. Therefore, let $v \in V(G)$ be the vertex which maps to the smallest-index vertex in $\ell(G)$ when using an isomorphism from G to $\ell(G)$. Since the isomorphism used above is arbitrary, this only determines the canonical deletion up to vertex orbits, but this is still appropriately invariant under isomorphism.

WARNING: When restricting to *connected* graphs, we must be sure that the deletion we select leads to a connected graph. Therefore, the canonical deletion should select the smallest-index vertex that is not a cut vertex.

Given access to a canonical deletion, we can now describe the full canonical deletion algorithm, given as Algorithm 6.1.

The reason Algorithm 6.1 is correct is due to the fact that following the canonical deletions $\text{del}(X)$ in reverse allows you to reconstruct the unique sequence of augmentations that correspond to those deletions (just in reverse order).

Definition 6.3. Given a labeled object $X \in \mathcal{L}$, the *deletion sequence* from X is a sequence $X_0, X_1, X_2, \dots, X_k$ of labeled objects so that

1. The first object X_0 is equal to X.
2. For $i \in \{1, \dots, k\}$, $\delta^{-1}(\text{del}(X_{i-1})) \in \mathcal{A}(X_i)$. That is, the canonical deletion from X_{i_1} corresponds to an augmentation from X_i.
3. The last object X_k is a base object: $X_k \in B$.

Thus, the objects that are visited by Algorithm 6.1 are exactly those with deletion sequences $X_0, X_1, \dots, X_k$ where Prune(X_i) never holds and $X_k \in B$. In the call to CanonicalDeletion(X_{i-1}), only the augmentation $A \in \mathcal{A}(X_{i-1})$ that has $\delta(A) \cong \text{del}(X_i)$ will succeed in generating X_i, leading to the call CanonicalDeletion(X_i). Since we are restricting to a single representative of the augmentation orbits, this leads to exactly one generation of X_i.

Algorithm 6.1 CanonicalDeletion(n, X)

```
if Prune($X$) then
    There are no solutions "above" this object.
    return
end if
if IsSolution($X$) then
    This object is a solution.
    Output $X$
end if
if Order($X$) $\geq n$ then
    This object is too large to augment.
    return
end if
for all augmentations $A \in \mathcal{A}(X)$, up to isomorphism do
    Convert the augmentation into a deletion.
    $D \leftarrow \delta(A)$.
    Find the labeled object that corresponds to that deletion.
    $Y \leftarrow \mathcal{D}^{-1}(D)$.
    Compute that object's canonical deletion.
    $D' \leftarrow \mathrm{del}(Y)$.
    Test if the current augmentation corresponds to that canonical deletion.
    if $D \cong D'$ then
        Visit the object $Y$.
        call CanonicalDeletion($n, Y$)
    end if
end for
return
```

Note 6.4. It is very important that we make the distinction that for an augmentation $A \in \mathcal{A}(X)$ we have $\delta(A) \cong \mathrm{del}(Y)$ for an augmented object Y and not simply that X is isomorphic to the object resulting from performing the deletion $\mathrm{del}(Y)$ on Y. This is due to the fact that multiple non-isomorphic deletions may lead to the same unlabeled object. Recall that Figure 6.3(b) gave an example of such an ambiguity.

Figure 6.5 presents an example of the complicated network that may exist between the augmentations and deletions of combinatorial objects, but that the canonical deletion selects a subtree structure to this network. Specifically, the canonical deletions are presented as thick lines and there is exactly one path from a given object to the base object.

Example. Algorithm 6.2 is a specific implementation of the canonical deletion algorithm for generating graphs by vertex augmentations.

Algorithm 6.2 GraphCanonicalDeletion(n, G)

```
if $\Delta(G) \geq 5$ or $\chi(G) \geq 4$ then
    There are no solutions with G as an induced subgraph.
    return
end if
if $n(G) \equiv n$ and $\delta(G) \equiv \Delta(G) \equiv 4$ and $\chi(G) \equiv 3$ then
    This graph is a solution.
    Output $G$
    return
end if
if $n(G) \geq n$ then
    This graph is too big for our target.
    return
end if
for all orbits $\mathcal{O}$ of non-empty sets $S \subseteq V(G)$ with $|S| \leq 4$ do
    Let $S \in \mathcal{O}$ be any representative.
    Create the augmented graph.
    $H \leftarrow G + v_S$.
    Compute that object's canonical deletion.
    $(H, v') \leftarrow \mathrm{del}(H)$.
    Test if the current augmentation corresponds to that canonical deletion.
    if $v_S$ and $v'$ are in the same $H$-orbit then
        Visit the graph H.
        call GraphCanonicalDeletion($n, H$)
    end if
end for
return
```

6.5 Efficiency Considerations

Based on my experience in creating specific implementations of the canonical deletion algorithm, I contribute a few suggested methods for writing more efficient software.

1. *Deletion by filtering*. Calling *nauty* is an expensive computation, so it should be avoided whenever possible. However, we cannot exactly compute the canonical deletion without it, but we can sometimes determine that our current augmentation does not correspond to the canonical deletion without using *nauty*. Therefore, I instead create a method IsCanonical(H, A), where H is the augmented graph and A is the augmentation I used to create H. Then, the method can return `False` at the earliest time that it detects this is NOT the canonical deletion. This process is then helped by a staged selection of canonical deletion:

   a) Start with an easy invariant metric, such as minimum degree of a vertex. This restricts the possible deletions very quickly in most cases.

   b) Define a sequence of more complicated invariant metrics, such as minimizing the sum of the squares of all the neighbor degrees. Such invariants can remove even more choices without being overly costly to compute.

   c) If all previous restrictions on the choice of canonical deletion did not prove that the current augmentation is not canonical, we have two options. We can check if the list of feasible deletions that fit the previous invariants has size exactly one. If so, then we know without a doubt that this deletion is canonical. Otherwise, we need to select a deletion using a canonical labeling and then check if the current augmentation is in orbit with that canonical deletion. Both of these checks can be done using a single call to *nauty*.

2. *Reduced augmentation count*. In the previous suggestion, we restricted the choice of canonical deletion to a simple invariant minimization, such as minimizing the vertex degree. Making such a restriction as part of your canonical deletion can reduce the number of augmentations you attempt, for instance by only augmenting vertices of degree at most $\delta(G) + 1$.

3. *Pruning in deletion*. Again, since calling *nauty* is probably the most computationally expensive subroutine in this algorithm, it may be useful to check if the augmented graph should be pruned even before finishing the canonical deletion. This may depend on how complicated your Prune(G) method is, but if Prune(G) holds, there is no reason to visit that graph and hence no reason to compute a canonical labeling.

4. *Skipping augmentation orbit calculation*. For some augmentations, it may be difficult to completely compute the orbits of augmentations. Vertex augmentations is such an example, because there are many possible neighborhood sets. It may be more beneficial to ignore the automorphism calculation and instead just attempt every augmentation. For every successful augmentation, store a canonical labeling of the augmented object. Then, the objects generated from the current object are visited if and only if the augmentation corresponds to the canonical deletion *and* that augmented object is not isomorphic to any previous augmentation. By storing a list of visited objects at a given node, we are using the canonical labelings *locally*, which can be very efficient. See McKay's original paper [92] for more details about this strategy.

5. *Experiment with the augmentation step*. Depending on what type of objects you are generating, there may be something special about their structure that you can exploit in the augmentation step. McKay [92] provides an example of generating triangle-free graphs by making the vertex augmentations only use independent sets[4]. More radical experimentation of augmentations is described in the next section.

6.6 Big Augmentations

One major insight of this thesis is the use of augmentations that are customized to the given problem. Most previous implementations of canonical deletion focused on using vertex or edge augmentations. By customizing the augmentation to the specific problem, we can gain some properties that vertex or edge augmentations lack, such as monotonicity of invariants, strength in pruning, or simply a smaller set of objects to generate.

In Chapter 7, we develop a method to generate 2-connected graphs by *ear augmentations*. While this can be used to generate 2-connected graphs, the technique generates 2-connected graphs slower than using vertex augmentations and filtering. In Chapter 8, we use this method to verify the Edge Reconstruction Conjecture on 2-connected graphs. Since dense graphs are edge-reconstructible, the number of graphs to verify decreases. Also, the augmentation step allows a method to test pairs of graphs without needing to check all pairs, just pairs with the same canonical deletion.

The ear augmentations become particularly powerful when applied to an extremal problem in Chapter 9. A graph of order n with p perfect matchings is p-

---

[4] Observe that in a triangle-free graph every vertex neighborhood is an independent set.

extremal if it has the maximum number of edges for that n and p. After proving some structure theorems about the infinite family of p-extremal graphs, we find that the structure of all p-extremal graphs depends on a finite[5] list of fundamental graphs. To determine this list of fundamental graphs, we find by the Lovász Two Ear Theorem [85] that they can be built using a very restricted type of ear augmentations. One important property of these augmentations is that the number of perfect matchings is monotone: more augmentations leads to more perfect matchings. Vertex augmentations do not preserve the number of perfect matchings at all. Further, there is a list of special subsets of the vertices called *barriers*. The list of barriers is difficult to compute from scratch, but using ear augmentations we can update the list using a very simple algorithm. Finally, by proving a new extremal theorem, we are able to use the ear augmentations to significantly prune the search space as we can bound the number of edges possible in further augmentations. Executing this search significantly extended the current knowledge of p-extremal graphs.

In Chapter 11, we search for *uniquely* K_r*-saturated graphs*. The main approach is to augment by a copy of K_r^- (a K_r with one edge deleted) at every step. We started by implementing this augmentation within canonical deletion. This approach was successful in that the search was more efficient than vertex augmentations. However, it was not significantly better than previous augmentations: we could generate all examples up to 14 or 15 vertices, while vertex augmentations with pruning could generate all examples up to 13 vertices. Thus, in Chapter 11 we use a technique called *orbital branching* with a similar augmentation step and exhaustively search for uniquely K_r-saturated graphs over a larger number of vertices.

Determining which of these techniques to use (and which augmentation to use)

---

[5] The list of fundamental graphs is finite for a fixed p.

requires experience, experimentation, and a bit of luck.

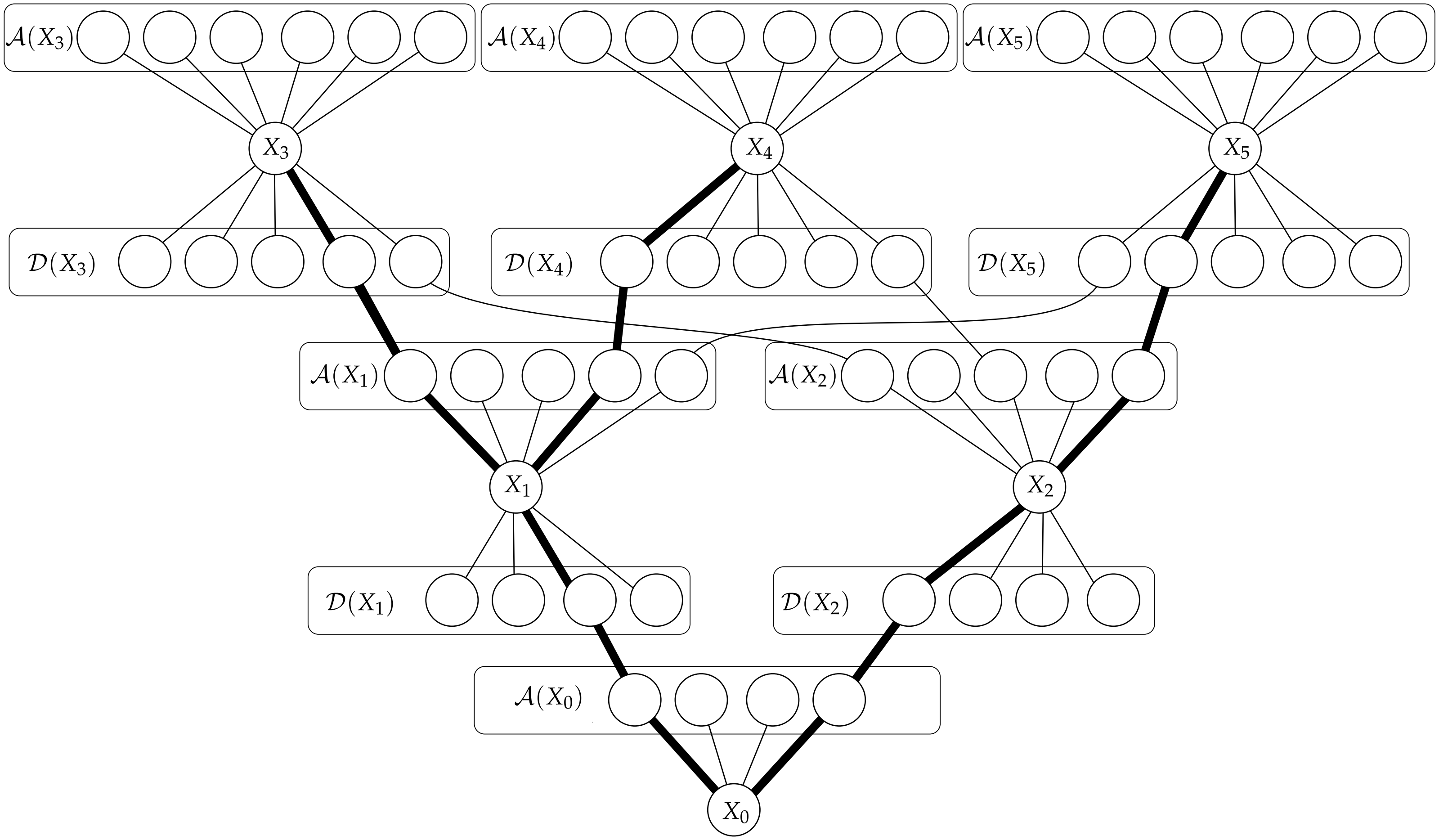


Figure 6.5: The canonical deletion tree within the lattice of augmentations and deletions.

Chapter 7

Ear Augmentations

If a connected graph G has a vertex x so that $G - x$ is disconnected or a single vertex, then G is *separable*. Otherwise, G is 2-*connected*, and there is no single vertex whose removal disconnects the graph. Many interesting graph families contain only 2-connected graphs, so we devise a generation technique that exploits the structure of 2-connected graphs.

A fundamental and well known property of 2-connected graphs is that they have an *ear decomposition*. An *ear* is a path $x_0, x_1, \dots, x_k$ so that x_0 and x_k have degree at least three and x_i has degree exactly two for all $i \in \{1, \dots, k-1\}$. An *ear augmentation* on a graph G is the addition of a path with at least one edge between two vertices of G. The augmentation process is also invertible: an *ear deletion* takes an ear $x_0, x_1, \dots, x_k$ in a graph and deletes all vertices $x_1, \dots, x_{k-1}$ (or the edge x_0x_1 if $k = 1$). Every 2-connected graph G has a sequence of subgraphs $G_1 \subset \cdots \subset G_\ell = G$ so that G_1 is a cycle and for all $i \in \{1, \dots, \ell - 1\}$, G_{i+1} is the result of an ear augmentation of G_i [146].

In this chapter, we describe a method for generating 2-connected graphs using ear augmentations. While we wish to generate unlabeled graphs, any computer implementation must store an explicit labeling of the graph. Without explicitly controlling the number of times an isomorphism class appears, a singe unlabeled graph may appear up to $n!$ times. An *isomorph-free* generation scheme for a class of combinatorial objects visits each isomorphism class exactly once. To achieve this goal, our strategy will make explicit use of isomorphisms, automorphisms, and orbits. The technique used in this work is an implementation of McKay's isomorph-free generation technique [92], which is sometimes called "canonical augmentation" or "canonical deletion". See [73] for a discussion of similar techniques. We implement this technique to generate only 2-connected graphs using ear augmentations.

Almost all graphs are 2-connected [142], even for graphs with a small number of vertices[1], so as a method of generating all 2-connected graphs, this method cannot significantly reduce computation compared to generating all graphs and ignoring the separable graphs. The strength of the method lies in its application to search over ear-monotone properties and to use the structure of the search to reduce computation. These strengths are emphasized in two applications of the technique.

In Chapter 8, we verify the Edge Reconstruction Conjecture on small 2-connected graphs. The structure of the search allows for a reduced number of pairwise comparisons between edge decks. Also, it is known that the Reconstruction Conjecture holds if all 2-connected graphs are reconstructible. Since graphs with more than $1 + \log(n!)$ edges are edge-reconstructible, we focus only on 2-connected graphs with at most this number of edges, providing a sparse set of graphs to examine. This verifies the conjecture on all 2-connected graphs up to 12 vertices, extending

---

[1] To see the overwhelming majority of 2-connected graphs, compare the number of unlabeled graphs [119] to the number of unlabeled 2-connected graphs [120].

previous results [91].

In Chapter 9, we use the technique to study graphs which have an extremal number of edges in the class of graphs with exactly p perfect matchings. The ear-augmentation technique is particularly effective due to a structural theorem which uses ear decompositions.

For a 2-connected graph, a vertex of degree at least three is a *branch vertex*. Vertices of degree two are *internal* vertices, as they are contained between the endpoints of an ear. Ears will be denoted with ε. For an ear ε, the *length* of ε is the number of edges between the endpoints and its *order* is the number of internal vertices between the endpoints. We will focus on the order of an ear. An ear of order 0 (length 1) is a single edge, called a *trivial* ear. Ears of larger order are *non-trivial*.

Given a graph G and an ear $\varepsilon = x_0, x_1, \dots, x_k$, the *ear deletion* $G - \varepsilon$ is the graph $G - x_1 - x_2 - \cdots - x_{k-1}$, where all internal vertices of ε are removed. For an ear $\varepsilon = x_0, x_1, \dots, x_{k-1}, x_k$ where $x_0, x_k \in V(G)$ but $x_1, x_2, \dots, x_{k-1}$ are not vertices in G, the *ear augmentation* $G + \varepsilon$ is given by adding the internal vertices of ε to G and adding the edges $x_i x_{i+1}$ for $i \in \{0, \dots, k-1\}$.

7.1 The search space and ear augmentation

In this section, we describe a general method for performing isomorph-free generation in specific families of 2-connected graphs.

Consider a family $\mathcal{F}$ of unlabeled 2-connected graphs. We say $\mathcal{F}$ is *deletion-closed* if every graph G in $\mathcal{F}$ which is not a cycle has an ear ε so that the ear deletion $G - \varepsilon$ is also in $\mathcal{F}$. For an integer $N \geq 3$, $\mathcal{F}_N$ is the set of graphs in $\mathcal{F}$ with at most N vertices.

This requirement implies that for every graph $G \in \mathcal{F}$, there exists a sequence

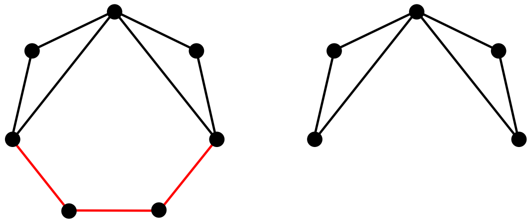


Figure 7.1: A 2-connected graph G and an ear ε whose removal makes $G - \varepsilon$ separable.

$G \supset G_1 \supset G_2 \cdots$ of ear deletions $G_{i+1} = G_i - \varepsilon_i$ where each graph G_i is in $\mathcal{F}$ and the sequence $\{G_i\}$ terminates at some cycle $C_k \in \mathcal{F}$. By selecting an ear deletion which is invariant to the representation of each G_i, we define a canonical sequence of ear-deletions that terminates at such a cycle. While generating graphs of $\mathcal{F}$, we shall only follow augmentations that correspond to these canonical deletions, giving a single sequence of augmentations for each isomorphism class in $\mathcal{F}$. This allows us to visit each isomorphism class in $\mathcal{F}$ exactly once using a backtracking search and without storing a list of previously visited graphs.

The search structure is that of a rooted tree: the root node is an empty graph, with the first level of the tree given by each cycle C_k in $\mathcal{F}_N$. Each subsequent search node is extended upwards by all canonical ear augmentations. Since the search does not require a list of previously visited graphs, disjoint subtrees are independent and can be run concurrently without communication. This leads to a search method which can be massively parallelized without a significant increase in overhead.

Note that being deletion-closed does not imply that *every* ear ε in G has $G - \varepsilon$ in the family. In fact, this does not even hold for the family of 2-connected graphs, as removing some ears leave the graph separable. See Figure 7.1 for an example of such an ear deletion.

Also, if $\mathcal{F}$ is deletion-closed, then so is $\mathcal{F}_N$. While the algorithms described could operate over $\mathcal{F}$, a specific implementation will have a bounded number (N) of vertices to consider. Operating over $\mathcal{F}_N$ allows for a finite number of possible ear augmentations at each step.

To augment a given labeled graph G, enumerate all pairs of vertices $x, y \in V(G)$ and orders $r \geq 0$ so that $|V(G)| + r \leq N$ and attempt adding an ear between x and y of order r. If an edge exists between x and y, then adding an ear of order 0 will immediately fail. However, all other orders produce valid 2-connected graphs. We then test if the augmentation $G + \varepsilon$ is in $\mathcal{F}$, discarding graphs which are not in the family.

7.2 Augmenting by orbits

By considering the automorphisms of a given graph, we can reduce the number of attempted ear augmentations. First, note that between a given pair of vertices, multiple ears of the same order are in orbit with each other. Second, if ε_1 is an ear between x_1 and y_1 and ε_2 is an ear between x_2 and y_2, then ε_1 and ε_2 are in orbit if and only if they have the same order and the vertex sets $\{x_1, y_1\}$, $\{x_2, y_2\}$ are in orbit under the automorphism group of G. Third, if the sets of vertices $\{x_1, y_1\}$ and $\{x_2, y_2\}$ are in orbit under the automorphism group of G, then the augmentations formed by adding an ear of order r between x_1 and y_1 is isomorphic to adding an ear of order r between x_2 and y_2.

This redundancy under graphs with non-trivial automorphism group is removed by computing the orbits of vertex pairs, then only augmenting ears between a single representative of a pair orbit. Pair orbits are computed by applying the generators of the automorphism group of G to the set of vertex pairs.

7.3 Canonical deletion of ears

While augmenting by orbits reduces the number of generated graphs, a *canonical deletion* is defined to guarantee that each unlabeled graph in $\mathcal{F}_N$ is enumerated exactly once. This selects a unique ear $\varepsilon = \text{Delete}_{\mathcal{F}}(G)$ so that $G - \varepsilon$ is in $\mathcal{F}$ and ε is invariant to the labeling of G. That is, if G_1 and G_2 are isomorphic graphs with deletions $\text{Delete}_{\mathcal{F}}(G_1) = \varepsilon_1$ and $\text{Delete}_{\mathcal{F}}(G_2) = \varepsilon_2$, then there is an isomorphism π from G_1 to G_2 so that π maps ε_1 to ε_2.

In order to compute a representative $\text{Delete}_{\mathcal{F}}(G)$ that is invariant to the labels of G, a canonical labeling of $V(G)$ is computed. A *canonical labeling* is a map $\text{lab}(G)$ which maps graphs G to permutations $\pi_G : V(G) \to \{0, 1, 2, \dots, |V(G)| - 1\}$ so that for every labeled graph $G' \cong G$, the map $\phi : V(G) \to V(G')$ given by $\phi(v) = \pi_{G'}^{-1}(\pi_G(v))$ for each $v \in V(G)$ is an isomorphism from G to G'. In this sense, the map π_G is invariant to the labels of $V(G)$. McKay's `nauty` library [93, 61] is used to compute this canonical labeling.

Once the canonical labeling is computed, the canonical deletion can be chosen by considering all ears ε whose deletion $(G - \varepsilon)$ remains in $\mathcal{F}_N$, and selecting the ear with (a) minimum length, and (b) lexicographically-least canonical label of branch vertices. Algorithm 7.1 details this selection procedure.

7.4 Full implementation

This isomorph-free generation scheme is formalized by the recursive algorithm $\text{Search}_{\mathcal{F}}(G, N)$, given in Algorithm 7.2. The full algorithm $\text{Search}_{\mathcal{F}}(N)$ searches over all graphs of order at most N in $\mathcal{F}$ and is initialized by calling $\text{Search}_{\mathcal{F}}(C_k, N)$ for each $k \in \{3, 4, \dots, N\}$. Since the recursive calls to $\text{Search}_{\mathcal{F}}(G, N)$ are indepen-

Algorithm 7.1 $\text{Delete}_{\mathcal{F}}(G)$ — The Default Canonical Deletion in $\mathcal{F}$

```
minOrder ← $n(G)$
minLabel ← $n(G)^2$
bestEear ← null
for all vertices $x \in V(G)$ with $\deg x \geq 3$ do
  for all ears $e$ incident to $x$ do
    Let $y$ be the opposite endpoint of $e$
    label ← $\min\{n(G)\pi_G(x) + \pi_G(y), n(G)\pi_G(y) + \pi_G(x)\}$
    $r$ ← order of $e$
    if $G - e \in \mathcal{F}_N$ then
      if $r <$ minOrder then
        minOrder ← $r$
        minLabel ← label
        bestEar ← $(x, y, r)$
      else if $r =$ minOrder and label < minLabel then
        minLabel ← label
        bestEar ← $(x, y, r)$
      end if
    end if
  end for
end for
return bestEar
```

dent, they can be run concurrently without communication.

For some applications, it is possible to determine that no solutions are reachable under any sequence of ear augmentations. In such a case, the algorithm can stop searching at the current node to avoid computing all augmentations and canonical deletions. Let $\text{Prune}_{\mathcal{F}}(G)$ be the subroutine which detects if such a pruning is possible.

The framework for Algorithm 7.2 was implemented in the *TreeSearch* library (see Chapter 3), a C++ library for managing a distributed search using the Condor scheduler [134]. This implementation was executed on the Open Science Grid [107] using the University of Nebraska Campus Grid [143]. Performance calculations in this chapter are based on the accumulated CPU time over this heterogeneous set

Algorithm 7.2 Search$_{\mathcal{F}}(G, N)$ — Search all canonical augmentations of G in $\mathcal{F}_N$

```
if $\text{Prune}_{\mathcal{F}}(G) = $ true then
    return
end if
if $G$ is a solution then
    Store $G$
end if
$R \leftarrow N - n(G)$
for all vertex-pair orbits $\mathcal{O}$ do
    $\{x, y\} \leftarrow$ representative pair of $\mathcal{O}$
    for all orders $r \in \{0, 1, \dots, R\}$ do
        $G' \leftarrow G + \text{Ear}(x, y, r)$
        $(x', y', r') \leftarrow \text{Delete}_{\mathcal{F}}(G')$
        if $r = r'$ and $\{x', y'\} \in \mathcal{O}$ then
            $\text{Search}_{\mathcal{F}}(G', N)$
        end if
    end for
end for
return
```

of computation servers. For example, the nodes available on the University of Nebraska Campus Grid consist of Xeon and Opteron processors with a speed range of 2.0-2.8 GHz. All code and documentation written for this chapter is available as the *EarSearch* library, detailed in Appendix E.

7.5 Generating all 2-connected graphs

Using the isomorph-free generation scheme of canonical ear deletions, we can generate all unlabeled 2-connected graphs on N vertices or graphs on N vertices with exactly E edges.

Definition 7.1. Let N and E be integers. Set g_N to be the number of unlabeled 2-connected graphs on N vertices and $g_{N,E}$ to be the number of unlabeled 2-connected graphs on N vertices and E edges. $\mathcal{G}_N$ is the family of 2-connected graphs on up to

N	g_N	CPU time
5	10	0.01s
6	56	0.11s
7	468	0.26s
8	7123	10.15s
9	194066	5m 17.27s
10	9743542	7h 39m 28.47s
11	900969091	71d 22h 22m 49.12s

Table 7.1: Comparing g_N and the time to generate $\mathcal{G}_N$.

N vertices. $\mathcal{G}_{N,E}$ is the family of 2-connected graphs on up to N vertices and up to E edges.

Robinson [113] computed the values of g_N and $g_{N,E}$, listed in [120, 112]. Note that $\mathcal{G}_N$ and $\mathcal{G}_{N,E}$ are deletion-closed families, and can be searched using isomorph-free generation via ear augmentations. We revisit the three main behaviors of the algorithm: canonical deletion, pruning, and determining solutions.

Canonical Deletion: The canonical deletion algorithm in Algorithm 7.1 suffices for the class of 2-connected graphs. Recall this algorithm selects from ears ε so that $G - \varepsilon$ remains 2-connected, selecting one of minimum length and breaking ties by using the canonical labels of the endpoints.

Pruning: If the number of edges is fixed to be E, a graph with more than E edges should be pruned. Also, a graph on $n(G) < N$ vertices must add at least $N - n(G) + 1$ edges during ear augmentations in order to achieve N total vertices. If $e(G) + (N - n(G) + 1) > E$, then no graph on N vertices with at most E edges can be reached by ear augmentations from G. In this case, the node can be pruned.

Solutions: A 2-connected graph is a solution if and only if $n(G) = N$, and if E is specified then $e(G) = E$ must also hold.

Table 7.1 compares the number of 2-connected graphs of order N and the CPU time to enumerate all such graphs. Both the computation times and the sizes of

N	$E = 11$	$E = 12$	$E = 13$	$E = 14$	$E = 15$	$E = 16$	$E = 17$	$E = 18$	$E = 19$	$E = 20$
10	9	121	1034	5898	23370	69169	162593	317364	530308	774876
	0.01	0.16	1.73	12.99	65.88	167.12	472.68	972.62	2048.85	3631.71
11		11	189	2242	17491	94484	380528	1212002	3194294	7197026
		0.02	0.38	5.52	56.10	260.53	1212.89	4069.09	13104.24	32836.53
12			13	292	4544	46604	334005	1747793	7274750	24972998
			0.03	0.86	17.56	286.00	1226.71	6930.00	33066.80	125716.68
13				15	428	8618	113597	1031961	6945703	36734003
				0.05	1.83	44.64	469.02	5174.92	39018.15	227436.84
14					18	616	15588	257656	2925098	24532478
					0.08	3.82	90.51	1573.81	21402.18	183482.70
15						20	855	26967	519306	7654299
						0.12	7.56	198.84	4567.43	76728.79
16							23	1176	44992	1111684
							0.18	15.56	498.20	13176.05

Table 7.2: Comparing $g_{N,E}$ (above) and the time to generate $\mathcal{G}_{N,E}$ (below, in seconds).

the sets grow exponentially. Since the number of 2-connected graphs on N vertices grows so quickly, to test the performance for larger orders, the number of edges was also fixed to be slightly more than N. Table 7.2 shows these computation times.

Chapter 8

The Edge-Reconstruction Conjecture

In this chapter, we apply the isomorph-free generation of 2-connected graphs to test the Edge Reconstruction Conjecture. We restrict the search to sparse 2-connected graphs and utilize the structure of the search tree in order to minimize pairwise comparisons among the list of generated graphs.

8.1 Background

The Reconstruction Conjecture and Edge Reconstruction Conjecture are two of the oldest unsolved problems in graph theory. Given a graph G, the *vertex deck* of G is the multiset of unlabeled graphs given by the vertex-deleted subgraphs $\{G - v : v \in V(G)\}$. The *edge deck* of G is the multiset of unlabeled graphs given by the edge-deleted subgraphs $\{G - e : e \in E(G)\}$. A graph G is *reconstructible* if all graphs with the same vertex deck are isomorphic to G. G is *edge reconstructible* if all graphs with the same edge deck are isomorphic to G.

Conjecture 8.1 (The Reconstruction Conjecture). *Every graph on at least three vertices is reconstructible.*

Conjecture 8.2 (The Edge Reconstruction Conjecture)**.** *Every graph with at least four edges is edge reconstructible.*

Bondy's survey [19] discusses many classic results on this topic. Greenwell [54] showed that the vertex deck is reconstructible from the edge deck, so a reconstructible graph is also edge reconstructible. Therefore, the Edge Reconstruction Conjecture is weaker than the Reconstruction Conjecture.

Yang [149] showed that the Reconstruction Conjecture can be restricted to 2-connected graphs.

Theorem 8.3 (Yang [149])**.** *If all 2-connected graphs are reconstructible, then all graphs are reconstructible.*

The proof considers a separable graph G and tests if the complement $\overline{G}$ is 2-connected. If $\overline{G}$ is 2-connected, $\overline{G}$ is reconstructible (by hypothesis) and since the vertex deck of $\overline{G}$ is reconstructible from the vertex deck of G, G is also reconstructible. If $\overline{G}$ is not 2-connected, Yang reconstructs G directly using a number of possible cases for the structure of G. There has been work to make Yang's theorem unconditional by reconstructing separable graphs such as trees [88], cacti [49, 96], and separable graphs with no vertices of degree one [89], but separable graphs with vertices of degree one have not been proven to be reconstructible.

Verifying the Reconstruction Conjecture requires that every pair of non-isomorphic graphs have non-isomorphic decks. Running a pair-wise comparison on every pair of isomorphism classes on n vertices is quickly intractable. McKay [91] avoided this issue and verified the conjecture on graphs up to 11 vertices by incorporating the vertex deck as part of the canonical deletion. McKay used vertex augmentations to generate the graphs, so a canonical deletion in this search is essentially selecting a canonical vertex-deleted subgraph. His technique selects the deletion based only on the vertex deck, so two graphs with the same vertex deck would be immediate siblings in the search tree. With this observation, only siblings require pairwise comparison, making the verification a reasonable computation. We use a modification of McKay's technique within the context of 2-connected graphs to test the Edge Reconstruction Conjecture on small graphs. This strategy was first proposed in unpublished work of Hartke, Kolb, Nishikawa, and Stolee [59].

8.2 The Search Space

To search for pairs of non-isomorphic graphs with the same edge deck, we adapt McKay's sibling-comparison strategy as well as a density argument. If a graph has sufficiently high density, then the graph is edge reconstructible.

Theorem 8.4 (Lovász, Müller [84, 97])**.** *A graph on N vertices and E edges with either $E > \frac{1}{2}\binom{N}{2}$ or $E > 1 + \log_2(N!)$ is edge reconstructible.*

Note that for all $N \geq 11$, $1 + \log_2(N!) < \frac{1}{2}\binom{N}{2}$.

Definition 8.5. Let $\mathcal{R}_N$ be the class of 2-connected graphs G with at most N vertices and at most $1 + \log_2(N!)$ edges.

Note that this definition of $\mathcal{R}_N$ bounds the number of edges as a function of N which is independent of the number of vertices of a specific graph.

Corollary 8.6. *For $N \geq 11$, all 2-connected graphs G with at most N vertices and $G \notin \mathcal{R}_N$ are edge reconstructible.*

We shall use $\mathcal{R}_N$ as our search space. It is deletion-closed, since removing an ear will always decrease the number of edges.

Within the context of the ear-augmentation generation algorithm, we generate 2-connected graphs. When trivial ears are added, these are the same as edge-augmentations. We will show that if a non-trivial ear is added, then the resulting graph is edge reconstructible and its edge deck does not need to be compared to other edge decks. Hence, an edge deck must be compared only when the final augmentation that generated the graph is an edge augmentation, where the canonical deletion can be selected using the edge deck.

We begin by discussing graphs which are known to be reconstructible or edge reconstructible.

Proposition 8.7. *A 2-connected graph G is edge reconstructible if any of the following hold:*

1. *There is an ear with at least two internal vertices.*
2. *There is a branch vertex v which is incident to only non-trivial ears.*
3. *G is regular.*

Proof. **(**1) By reconstructing the degree sequence, we recognize that all vertices have degree at least two. Since there is an ear with at least two internal vertices, there is an edge internal to that ear with endpoints of degree two. In that edge-deleted card, there are exactly two vertices of degree one, which must be connected by the missing edge, giving G.

(2) Let d be the degree of v. By reconstructing the vertex deck, we can recognize that the card for $G - v$ is missing a vertex of degree d and that there are d vertices of degree one in $G - v$. Attaching v to these vertices reconstructs G.

(3) For a d-regular graph G, every edge-deleted subgraph $G - e$ has exactly two vertices of degree $d - 1$ corresponding to the endpoints of e. □

Graphs satisfying any of the conditions of Proposition 8.7 are called *detectably edge reconstructible* graphs.

8.3 Canonical deletion in $\mathcal{R}_N$

In this section, we describe a method for selecting a canonical ear to delete from a graph in $\mathcal{R}_N$.

If we are able to determine that G is edge reconstructible, then the canonical deletion does not need to be generated from the edge deck. In such a case, we default to the canonical deletion algorithm $\text{Delete}_{\mathcal{F}}(G)$, where the canonical labeling of G gives the lex-first ear ε of minimum length so that $G - \varepsilon$ 2-connected.

If G is not detectably edge reconstructible, then all ears of G have at most one internal vertex, and every branch vertex is incident to at least one trivial ear. These properties allow us to find either a trivial ear or an ear of order one whose deletion remains 2-connected. Compute the minimum r so that there exists an ear ε in G of order r so that $G - \varepsilon$ is 2-connected. We prefer to select a trivial ear when available.

Out of the choices of possible order-r ear deletions, count the multiplicities for the degree set of the ear endpoints. Find the pair $\{d_1, d_2\}$ of endpoint degrees which has minimum multiplicity over all deletable ears of order r in G breaking ties by using the lexicographic order. Out of the deletable ears of order r and endpoint degrees $\{d_1, d_2\}$, we must select a canonical ear using the edge deck. If $r = 0$, any trivial deletable ear ε corresponds to the edge-deleted subgraph $G - \varepsilon$. By computing the canonical labels of these cards and selecting the lexicographically-least canonical string, we can select a canonical edge. If $r = 1$, there are two edges in the ear that can be deleted to form edge-deleted subgraphs with a single vertex of degree 1 connected to a 2-connected graph. We compute the canonical labels of both cards, select the lexicographically-least canonical string, then find the lex-least string of those strings.

Due to the nature of the reconstruction problem, this canonical deletion procedure is not perfect. There are graphs G containing trivial ears $\varepsilon_1, \varepsilon_2$ whose deletions $G - \varepsilon_1$ and $G - \varepsilon_2$ are isomorphic, but ε_1 and ε_2 are not in orbit within G. If the edge-deleted subgraph $G - \varepsilon_1$ is selected as the canonical edge card, the deletion algorithm must accept both ε_1 and ε_2 as canonical deletions. This leads to a duplication of G in the search tree, but only in the limited case of a graph G which is not detectably edge reconstructible *and* such ambiguity appears. A similar concern occurs for the vertex-deletion case, but is not explained in [91].

To compare graphs with the same canonical deletion, we use three comparisons. The first compares the degree sequences. The second compares a custom reconstructible invariant[1], which is based on the degree sequence of the neighborhood of each vertex. The third and final check compares the sorted list of canonical strings for the edge-deleted subgraphs. During the search, there was no pair of graphs which satisfied all three of these checks.

8.3.1 Results

With the canonical deletion $\text{Delete}_{\mathcal{R}}(G)$, $\mathcal{R}_N$ was generated and checked for collisions in the edge decks of graphs which are not detectably reconstructible. Table 8.1 describes the computation time for $N \in \{8, \dots, 12\}$.

With this computation, we have the following theorem.

Theorem 8.8. *All 2-connected graphs on at most* 12 *vertices are edge reconstructible.*

---

[1] This invariant is not theoretically interesting, but is available in the source code. See the `GraphData::computeInvariant()` method.

N	$g(N)$	$\lvert\mathcal{R}_N\rvert$	Diff 1	Diff 2	Diff 3	CPU time
8	16	4804	145	177	187	8.01s
9	19	111255	6.19×10^3	5.72×10^3	4.77×10^3	5m 33.85s
10	22	3051859	7.13×10^5	6.00×10^5	4.21×10^5	6h 33m 40.59s
11	26	308400777	9.44×10^7	7.28×10^7	3.83×10^7	32d 20h 38m 08.16s
12	29	25615152888	12.00×10^9	9.60×10^9	4.47×10^9	10y 362d 13h 05m 39.13s

Table 8.1: Comparing $|\mathcal{R}_N|$ and the time to check $\mathcal{R}_N$. $g(N) = 1 + \lfloor \log_2(N!) \rfloor$.

This computation extends the previous result that all graphs of order at most 11 are vertex reconstructible [91]. To remove the 2-connected condition of Theorem 8.8, there are three possible methods. First, prove Yang's Theorem (Theorem 8.3) for the edge reconstruction problem. Second, Yang's Theorem could be made unconditional by proving that separable graphs are reconstructible or edge reconstructible. Third, a second stage of search could be designed to combine a list of two-connected graphs to form sparse separable graphs and test edge reconstruction on those cases.

Chapter 9

Extremal Graphs with a Given Number of Perfect Matchings

For even n and positive integer p, Dudek and Schmitt [38] defined $f(n,p)$ to be the maximum number of edges in an n-vertex graph having exactly p perfect matchings. Say that such a graph with $f(n,p)$ edges is p-*e*xtremal. We study the behavior of $f(n,p)$ and the structure of p-extremal graphs.

Although existence of a perfect matching can be tested in time $O(n^{1/2}m)$ for graphs with n vertices and m edges [95], counting the perfect matchings is #P-complete, even for bipartite graphs [139]. Let $\Phi(G)$ denote the number of perfect matchings in G. Bounds on $\Phi(G)$ are known in terms of the vertex degrees in G. For a bipartite graph G with n vertices in each part and degrees $d_1, \ldots, d_n$ for the vertices in one part, Brègman's Theorem [21] states that $\Phi(G) \leq \prod_{i=1}^{n}(d_i!)^{1/d_i}$. Kahn and Lovász (unpublished) proved an analogue for general graphs (other proofs were given by Friedland [46] and then by Alon and Friedland [5]). For a graph G with vertex degrees $d_1, \ldots, d_n$, the Kahn–Lovász Theorem states that $\Phi(G) \leq \prod_{i=1}^{n}(d_i!)^{1/2d_i}$. Both results were reproved using entropy methods by

Radhakrishnan [108] and by Cutler and Radcliffe [36], respectively. Gross, Kahl, and Saccoman [56] studied $\Phi(G)$ for a fixed number of edges; they determined the unique graphs minimizing and maximizing $\Phi(G)$.

Maximizing the number of edges when $\Phi(G)$ and n are fixed has received less attention. Hetyei proved that $f(n,1) = n^2/4$ (see [87, Corollary 5.3.14, page 173]). We describe Hetyei's construction inductively in a more general context.

Construction 9.1. The *Hetyei-extension* of G is the graph G' formed from G by adding a vertex x adjacent to all of $V(G)$ and one more vertex y adjacent only to x. Every perfect matching of G' contains xy and a perfect matching of G, so $\Phi(G') = \Phi(G)$. Starting with $G = K_2$, Hetyei-extension yields graphs with one perfect matching for all even orders.

When G has n vertices, $|E(G')| = |E(G)| + n + 1$. Since $(k+2)^2/4 = k^2/4 + k + 1$, we obtain $f(n,1) \geq n^2/4$ for all even n. (Note that when G has a unique perfect matching M, at most two edges join the vertex sets of any two edges of M; hence $f(n,1) \leq n/2 + 2\binom{n}{2} = n^2/4$.)

More generally, when $\Phi(G) = p$ and $|E(G)| = n^2/4 + c$, the Hetyei-extension of G yields $f(n+2,p) \geq (n+2)^2/4 + c$. This observation is due to Dudek and Schmitt [38]. $\square$

In light of the observation in Construction 9.1, we let $c(G) = |E(G)| - |V(G)|^2/4$ and call $c(G)$ the *excess* of G. For fixed p, Dudek and Schmitt proved that the maximum excess is bounded by a constant.

Theorem 9.2 (Dudek and Schmitt [38]). *For $p \in \mathbb{N}$, there is an integer c_p and a threshold n_p such that $f(n,p) = n^2/4 + c_p$ when $n \geq n_p$ and n is even. Also, $-(p-1)(p-2) \leq c_p \leq p$.*

Dudek and Schmitt determined c_p and n_p for $1 \leq p \leq 6$, although the proofs for $p \in \{5, 6\}$ were omitted since they were prohibitively long. They conjectured that $c_p > 0$ when $p \geq 2$. We prove their conjecture in Section 9.1 by generalizing Hetyei's construction. The construction yields $c_p > 0$ but does not generally give the best lower bounds. We give better lower bounds in Section 9.6; first we must analyze the structure of extremal graphs.

We develop a systematic approach to computing c_p. With this we give shorter proofs for $p \leq 6$ and identify the values c_p and n_p for $7 \leq p \leq 10$. Later in this chapter, we shall combine the ear-augmentation technique with these structure theorems to determine c_p and n_p for all $p \leq 27$. The complete behavior of c_p for larger p remains unknown.

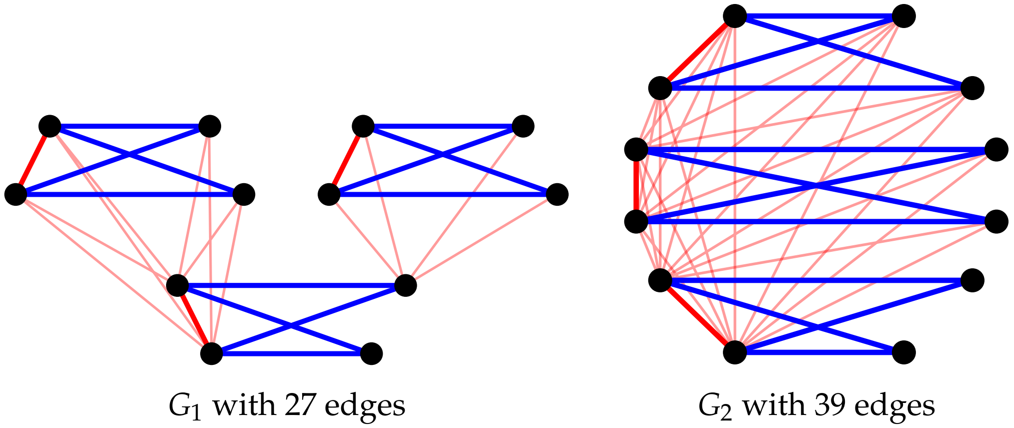
Definition 9.3. Let $\mathcal{F}_p$ denote the family of graphs that are p-extremal and have excess c_p; that is, $\mathcal{F}_p = \left\{G\colon\ \Phi(G) = p \text{ and } |E(G)| = \frac{|V(G)|^2}{4} + c_p\right\}$. Equivalently, $\mathcal{F}_p$ is the set of p-extremal graphs with at least n_p vertices.

We study the extremal graphs as a subfamily of a larger family.

Definition 9.4. A graph is *saturated* if the addition of any missing edge increases the number of perfect matchings.

Extremal graphs are contained in the much larger family of saturated graphs. Figure 9.1 shows a saturated graph G_1 with 12 vertices, eight perfect matchings, and 27 edges. Although G_1 is saturated, it is not 8-extremal, since the graph G_2 in Figure 9.1 has the same number of vertices and perfect matchings but has 39 edges.

Lovász's Cathedral Theorem (see [87]) gives a recursive decomposition of all saturated graphs; we describe it in Section 9.2. In terms of this construction, we describe the graphs in $\mathcal{F}_p$. In Sections 9.3 and 9.4, study of the cathedral con-



G_1 with 27 edges G_2 with 39 edges

Figure 9.1: Two graphs with eight perfect matchings

struction for extremal graphs allows us to reduce the problem of computing c_p to examining a finite (but large) number of graphs.

In Section 9.5, we extend $f(n, p)$ to odd n and study the corresponding extremal graphs. Section 9.6 gives constructions for improved lower bounds on c_p. In Section 9.7, we conjecture an upper bound on c_p that would be sharp for infinitely many values of p. The conjectured bound would be the best possible monotone upper bound, if true. Section 9.8 mentions several conjectures and discusses a computer search based on our structural results; the search found the extremal graphs for $4 \leq p \leq 10$.

We then take a second look at the problem, by using Lovász's Two-Ears Theorem directly in a new computational technique. Sections 9.9 through 9.13 contain the description of the computational technique and optimization strategies. With this new technique, we find the extremal graphs for $11 \leq p \leq 27$.

9.1 The Excess is Positive

We begin with a simple construction proving the Dudek-Schmitt conjecture that $c_p > 0$.

The *disjoint union* of graphs G and H (with disjoint vertex sets) is denoted $G + H$. The *join* of G and H, denoted $G \vee H$, consists of $G + H$ plus edges joining each vertex of G to each vertex of H. Thus the Hetyei-extension of G is $(G + K_1) \vee K_1$. A split graph is a graph whose vertex set is the union of a clique and an independent set.

Definition 9.5. The *Hetyei graph* with $2k$ vertices, produced iteratively in Construction 9.1 from K_2 by repeated Hetyei-extension, can also be described explicitly. It is the split graph with clique $\ell_1, \ldots, \ell_k$, independent set $r_1, \ldots, r_k$, and additional edges $\ell_i r_j$ such that $i \leq j$.

The Hetyei graph is the unique extremal graph of order $2k$ with exactly one perfect matching. It has $\frac{(2k)^2}{4}$ edges, so $c_1 = 0$. In the constructions here and in Section 9.6, the Hetyei graph is a proper subgraph, so the excess is larger.

In a graph having an independent set S with half the vertices, every perfect matching joins S to the remaining vertices. Therefore, to study the perfect matchings in such a graph it suffices to consider the bipartite subgraph consisting of the edges incident to S. In the Hetyei graph, the only perfect matching consists of the edges $\ell_i r_i$ for all $1 \leq i \leq k$.

For $m \in \mathbb{N}$, let $w(m)$ denote the number of 1s in the binary expansion of m.

Definition 9.6. For $p \geq 2$ and $k = \lceil \log_2(p-1) \rceil + 1$, let $(x_{k-2}, \ldots, x_0)$ be the binary $(k-1)$-tuple such that $p - 1 = \sum_{j=0}^{k-2} 2^j x_j$. The *binary expansion construction* for p, denoted $B(p)$, consists of the Hetyei graph with $2k$ vertices plus the edges

$\{\ell_{i+2}r_1\colon\ x_i = 1\}$ (see Fig. 9.2).

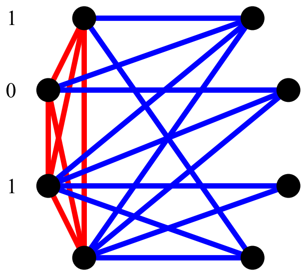


Figure 9.2: The graph $B(6)$

Theorem 9.7. *If $p \geq 2$, then $\Phi(B(p)) = p$ and $c(B(p)) = w(p-1)$. Thus $c_p \geq w(p-1) \geq 1$.*

Proof. Name the vertices of $B(p)$ as in the Hetyei graph. We construct perfect matchings in $B(p)$ by successively choosing the edges that cover $r_1, \ldots, r_k$. The matching $\{\ell_i r_i\colon 1 \leq i \leq k\}$ from the Hetyei graph is always present. If r_1 is matched to ℓ_{i+2} instead of to ℓ_1 for some nonnegative i, then for $r_2, \ldots, r_{i-1}$ exactly two edges are available when we choose the edge to cover this vertex. For vertices $r_i, \ldots, r_k$ in order, only one choice then remains. Therefore, each edge of the form $\ell_{i+2}r_1$ lies in 2^{i-2} perfect matchings.

The edge $\ell_{i+2}r_1$ exists if and only if $x_i = 1$ in the binary representation of $p-1$. Thus $\Phi(B(p)) = 1 + \sum_{i=2}^{k} 2^{i-2}x_{i-2} + 1 = 1 + p - 1 = p$. Since $B(p)$ is formed by adding $w(p-1)$ edges to the Hetyei graph, $c(B(p)) = w(p-1)$. $\square$

9.2 Lovász's Cathedral Theorem

As we have mentioned, Lovász's Cathedral Theorem characterizes saturated graphs. Since the extremal graphs are saturated, this characterization will be our starting point. Chapters 3 and 5 of Lovász and Plummer [87] present a full treatment of the subject. Another treatment appears in Yu and Liu [150]. A 1-*factor* of a graph G is a spanning 1-regular subgraph; its edge set is a perfect matching. An edge is *extendable* if it appears in a 1-factor.

Definition 9.8. A graph is *matchable* if it has a perfect matching. The *extendable subgraph* of a matchable graph G is the union of all the 1-factors of G. An induced subgraph H of G is a *chamber* of G if $V(H)$ is the vertex set of a component of the extendable subgraph of G.

Every vertex of a matchable graph G is incident to an extendable edge, so the chambers of G partition $V(G)$. Perfect matchings in G are formed by independently choosing perfect matchings in the chambers of G.

Lemma 9.9. *If a matchable graph G has chambers $H_1, \dots, H_k$, then $\Phi(G) = \prod_{i=1}^{k} \Phi(H_i)$.*

The chambers form the outermost decomposition in Lovász's structure (see Fig. 9.3). When the extendable subgraph is connected, there is only one chamber and no further breakdown.

Definition 9.10. A graph is *elementary* if it is matchable and its extendable subgraph is connected.

Tutte [138] characterized the matchable graphs. An *odd component* of a graph H is a component having an odd number of vertices; $o(H)$ denotes the number of odd components. An obvious necessary condition for existence of a perfect matching in G is that $o(G - S) \leq |S|$ for all $S \subseteq V(G)$. Tutte's 1-Factor Theorem states that this condition is also sufficient.

Definition 9.11. A *barrier* in a matchable graph G is a set $X \subseteq V(G)$ with $o(G - X) = |X|$.

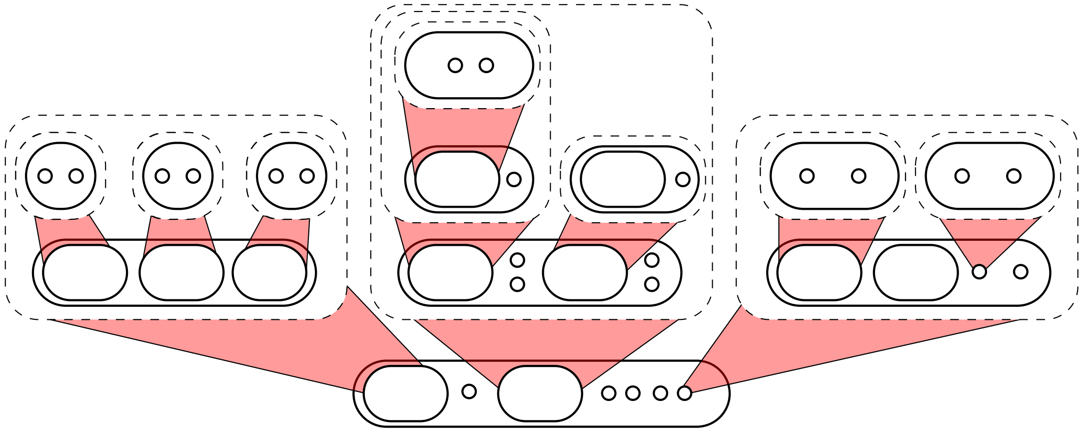


Figure 9.3: An example cathedral construction.

Lemma 9.12 (Lemma 5.2.1 [87])**.** *If G is elementary, then the family of maximal barriers in G is a partition of $V(G)$, denoted $\mathcal{P}(G)$.*

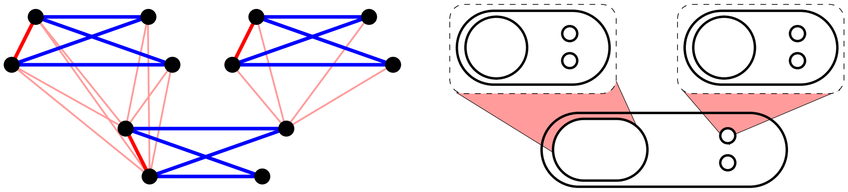
Construction 9.13 (The Cathedral Construction)**.** A graph G is a *cathedral* if it consists of (1) a saturated elementary graph G_0, (2) disjoint cathedrals $G_1, \ldots, G_t$ corresponding to the maximal barriers $X_1, \ldots, X_t$ of G_0, and (3) edges joining every vertex of X_i to every vertex of G_i, for $1 \leq i \leq t$. The graph G_0 is the *foundation* of the cathedral. The cathedral G_i may have no vertices when $i > 0$; thus every saturated elementary graph is a cathedral (with empty cathedrals over its barriers).

Since the cathedral construction has a cathedral “above" each maximal barrier of G_0, the construction is recursive, built from saturated elementary graphs. Each nonempty subcathedral G_i contains a saturated elementary graph $G_{i,0}$, and each maximal barrier $X_{i,j} \in \mathcal{P}(G_{i,0})$ has a cathedral $G_{i,j}$ over it in G_i. Figure 9.3 illustrates the cathedral construction. Here cathedrals are indicated by dashed curves (except for the full cathedral). Each foundation is indicated by a solid curve, as are the barriers within it.

Theorem 9.14 (The Cathedral Theorem; Theorem 5.3.8 [87])**.** *A graph G is saturated if and only if it is a cathedral. The foundation G_0 in the cathedral construction of G is unique, and every perfect matching in G contains a perfect matching of G_0.*

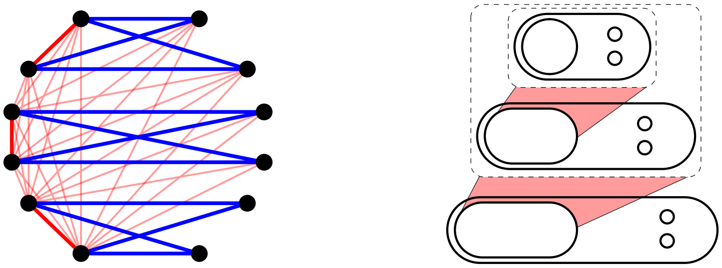
Since each perfect matching in a cathedral G contains a perfect matching of G_0, the edges joining G_0 to the cathedrals $G_1, \dots, G_t$ appear in no perfect matching. Therefore, G_0 is a chamber in G. Recursively, the foundations of the subcathedrals are the chambers of G.

The saturated graphs of Figure 9.1 are cathedrals having the same chambers (and hence the same number of perfect matchings). Their cathedral structures are shown in Figure 9.4.



G_1 with 27 edges

G_1 cathedral



G_2 with 39 edges

G_2 cathedral

Figure 9.4: The saturated graphs from Figure 9.1 and their cathedral structures.

Let $G \in \mathcal{F}_p$ be a p-extremal graph. Since G is extremal, it is saturated, and hence it is a cathedral. Recall that the Hetyei-extension of G is $(G + K_1) \vee K_1$. The complete graph K_2 is a saturated elementary graph; its barriers are single vertices, say $\{x_1\}$ and $\{x_2\}$. Letting $G_1 = G$ and $G_2 = \text{null}$, we obtain the Hetyei-extension of G as a cathedral with $G_0 = K_2$.

9.3 Extremal Graphs are Spires

From the cathedral structures of the two graphs in Figure 9.4, it is easy to see why G_2 has many more edges. Nesting of cathedrals generates many edges from foundations to the cathedrals over them. We introduce a special term for cathedrals formed in this way.

Definition 9.15. A *spire* is a cathedral in which at most one maximal barrier in the foundation has a nonempty cathedral over it, and that nonempty cathedral (if it exists) is a spire. In particular, every saturated elementary graph is a spire.

In Figure 9.4, the graph G_2 is a spire, while G_1 is not. By the recursive definition, the chambers of a spire G form a list $(H_0, \ldots, H_k)$ such that each H_i is the foundation of the spire induced by $\bigcup_{j=i}^{k} V(H_j)$, and in H_i with $i < k$ there is a maximal barrier Y_i that is adjacent to the vertices of the spire induced by $\bigcup_{j=i+1}^{k} V(H_i)$. We then say that G is a spire generated by $H_0, \ldots, H_k$ over $Y_0, \ldots, Y_k$.

Our first goal is to prove that extremal graphs are spires.

Lemma 9.16. *Every p-extremal graph is a spire such that in each chamber, the maximal barrier having neighbors in later chambers is a barrier of maximum size.*

Proof. Since a p-extremal graph is saturated, it is a cathedral. Let G be a cathedral having nonempty cathedrals G_i and G_j over maximal barriers X_i and X_j in its foundation, with $|X_i| \geq |X_j|$. Let G' be the cathedral obtained from G by removing G_j from the neighborhood of X_j and attaching it instead as a cathedral over a barrier in an innermost chamber of G_i. The cathedrals over the barriers of innermost chambers are empty, so G' is a cathedral.

The chambers of G and G' are isomorphic, so $\Phi(G) = \Phi(G')$, but G' has more edges. We replaced $|X_j| \cdot |V(G_j)|$ edges with $|X_i| \cdot |V(G_j)|$ edges, and also new edges were created incident to an innermost chamber over X_i. We conclude that in a p-extremal graph, only one maximal barrier of the foundation has a nonempty chamber over it. Also, that must be a largest barrier, since otherwise shifting to a larger one increases the number of edges, again without changing the number of perfect matchings. The claim follows by induction. □

The number of edges in a spire is maximized by ordering the chambers greedily.

Lemma 9.17. *Let* $\{H_0, \ldots, H_k\}$ *be saturated elementary graphs. Let* $n_i = |V(H_i)|$, *and let* s_i *be the maximum size of a barrier in* H_i. *Among the spires having* $H_0, \ldots, H_k$ *as chambers, the number of edges is maximized by indexing the chambers so that* $\frac{s_0}{n_0} \geq \cdots \geq \frac{s_k}{n_k}$.

Proof. For a spire G generated by $H_0, \ldots, H_k$ indexed in a different order, let i be an index such that $\frac{s_i}{n_i} < \frac{s_{i+1}}{n_{i+1}}$. Form a spire G' from G by interchanging H_i and H_{i+1} in the ordering (always the spire after H_j is built over a largest barrier Y_j of H_j).

In G' and G, the edges from $Y_i \cup Y_{i+1}$ to other chambers are the same. Only the edges joining $V(H_i)$ and $V(H_{i+1})$ change. In G, there are $s_i n_{i+1}$ such edges, and in G' there are $s_{i+1} n_i$ of them. By the choice of i, the change increases the number of edges. The number of perfect matchings remains unchanged.

Hence a p-extremal spire has its chambers ordered as claimed. □

Note that always $\frac{s_i}{n_i} \leq \frac{1}{2}$ for a chamber H_i in a spire G, since $o(H_i - X_i) = |X_i|$ for any barrier X_i in H_i. We show next that the excess $c(G)$ is subadditive over the chambers.

Lemma 9.18. *If* G *is a spire generated by* $H_0, \ldots, H_k$ *over* $Y_0, \ldots, Y_k$, *with* $s_i = |Y_i|$ *and* $n_i = |V(H_i)|$, *then* $c(G) \leq \sum_{i=0}^{k} c(H_i)$, *with equality if and only if* $\frac{s_0}{n_0} = \cdots = \frac{s_{k-1}}{n_{k-1}} = \frac{1}{2}$.

Proof. Let $m_i = |E(H_i)|$. Counting edges within chambers and from chambers to barriers in earlier chambers, we have $|E(G)| = \sum_{i=0}^{k} m_i + \sum_{0\le i<j\le k} s_i n_j$. Always $s_i \le \frac{1}{2}n_i$ and $m_i = \frac{1}{4}n_i^2 + c(H_i)$. Thus

$$\frac{n^2}{4} + c(H) \le \sum_{i=1}^{k} \left[\frac{n_i^2}{4} + c(H_i)\right] + \sum_{0\le i<j\le k} \frac{1}{2} n_i n_j = \frac{1}{4}\left[\sum_{i=0}^{k} n_i\right]^2 + \sum_{i=0}^{k} c_i.$$

Therefore, $c(G) \le \sum_{i=0}^{k} c(H_i)$, with equality if and only if $\frac{s_i}{n_i} = \frac{1}{2}$ for $i < k$. □

9.4 Extremal Chambers

We now know how to combine chambers in the best way, so it remains to determine which chambers should be used. A chamber is a saturated elementary graph, meaning that its extendable subgraph has just one component. We will bound the size of a saturated elementary graph with n vertices by bounding separately the *extendable* edges (those in perfect matchings) and the *free* edges (those in no perfect matching).

When G is elementary, the maximal barriers partition $V(G)$. Since each barrier matches to vertices outside it in any perfect matching, all edges within barriers are free. Also, adding such edges does not increase the number of perfect matchings. Thus in a saturated graph, the barriers are cliques. To bound the number of free edges, the crucial fact is that in a saturated elementary graph, the only free edges are those within barriers (proved in Lemma 5.2.2.b of Lovász and Plummer [87]).

Lemma 9.19. *If G is a saturated elementary n-vertex graph with ℓ maximal barriers, then G has at most $q\binom{\ell-1}{2} + \binom{r+1}{2}$ free edges, where $q = \left\lfloor \frac{n-\ell}{\ell-2} \right\rfloor$ and $r = n - \ell - q(\ell - 2)$.*

Proof. Let $x_1, \dots, x_\ell$ be the sizes of the barriers, so $\sum_{i=1}^{\ell} x_i = n$. Since each barrier is a clique, there are exactly $\sum_{i=1}^{\ell}\binom{x_i}{2}$ free edges. The sizes of the barriers are further restricted because deleting a barrier of size x_i must leave x_i odd components. Since the other barriers are cliques, deleting a barrier leaves at most $\ell - 1$ components. Thus $1 \le x_i \le \ell - 1$ for all i.

If $a \le b$, then $\binom{a-1}{2} + \binom{b+1}{2} > \binom{a}{2} + \binom{b}{2}$ (shifting a vertex from an a-clique to a b-clique increases the number of edges). Subject to the constraints we have specified, the number of free edges is thus bounded by greedily choosing as many of $x_1, \ldots, x_\ell$ to equal $\ell - 1$ as possible, given that at least one unit must remain for each remaining variable. Let q be the number of values equal to $\ell - 1$. Among the remaining values, whose total is less than $\ell - 1$, all values should be 1 except for one. After allocating 1 to each of these $\ell - q$ values, a total of r remains, where $0 \le r < \ell - 2$. Thus $n = q(\ell - 1) + (\ell - q) + r$, which we write as $n - \ell = q(\ell - 2) + r$.

The specified choice of q and r satisfies all the conditions, and the bound on the number of free edges is then as claimed. □

We show next that the bound in Lemma 9.19 is maximized when all barriers except one are singletons, producing $\ell = 1 + n/2$.

Corollary 9.20. *A saturated elementary n-vertex graph has at most* $\frac{n^2}{8} - \frac{n}{4}$ *free edges.*

Proof. The proof of Lemma 9.19 describes how to maximize $\sum_{i=1}^{\ell}\binom{x_i}{2}$ subject to $1 \le x_i \le \ell - 1$. Since barriers in saturated graphs are cliques, the number of odd components left by deleting a barrier is at most the number of other barriers, but it must equal the size of the barrier deleted. Hence each barrier has size at most $n/2$, which yields $\ell \le n/2 + 1$.

Thus $2 \le \ell \le n/2 + 1$. Since $0 \le r < \ell - 2$, we have $\binom{r+1}{2} \le r(\ell - 1)/2$ (with equality only when $r = 0$). Hence

$$q\binom{\ell-1}{2} + \binom{r+1}{2} \le \frac{q(\ell-1)(\ell-2) + r(\ell-1)}{2} = \frac{(\ell-1)(n-\ell)}{2}.$$

The upper bound is maximized at $(\ell - 1) = (n-1)/2$, among integers when $\ell \in \{n/2, n/2+1\}$. The value there is $\frac{1}{2}\left(\frac{n}{2}-1\right)\left(\frac{n}{2}\right)$, which is the claimed bound.

□

Next consider the extendable edges. Deleting the edges within barriers yields a graph in which every edge is extendable. Such graphs are called *1-extendable*, which motivates our name for extendable edges (the term *matching-covered* has also been used for 1-extendable graphs). Since the extendable edges form a 1-extendable graph, we seek a bound on the size of 1-extendable graphs with n vertices. All such graphs are 2-connected, and 2-connected graphs are precisely those constructed by ear decompositions. The 1-extendable graphs have special ear decompositions that yield a bound on the number of edges, described by the "Two Ears Theorem" of Lovász.

Definition 9.21. Let G be a 1-extendable graph. A *graded ear decomposition* of G is a list $G_0, \ldots, G_k$ of 1-extendable graphs such that $G_k = G$, each $G - V(G_i)$ is matchable, and each G_i for $i > 1$ is obtained from G_{i-1} by adding disjoint ears of odd length. A graded ear decomposition of G is *non-refinable* if no other graded ear decomposition of G contains it.

Theorem 9.22 (Two Ears Theorem; Lovász and Plummer [86]; see also Section 5.4 of [87])**.** *Every* 1*-extendable graph has a non-refinable graded ear decomposition in which each subgraph arises by adding at most two ears to the previous one (starting with any single edge).*

For example, such a decomposition of K_4 starts with any edge, adds one ear to complete a 4-cycle, and then adds both remaining edges as ears. Both ears must be added in the last step, because adding just one of them does not produce a 1-extendable graph.

Lovász and Plummer [87, page 178] remark that long graded ear decompositions are desirable, because $\Phi(G) \geq k+1$ when G has a graded ear decomposition $G_0, \ldots, G_k$. We explain and use this fact in our next lemma.

Lemma 9.23. *For $p \geq 2$, a 1-extendable graph G with $\Phi(G) = p$ has at most $2p - 4 + n$ edges.*

Proof. Let $G_0, \ldots, G_k$ be an ear decomposition as guaranteed by Theorem 9.22. Since the decomposition is non-refinable, G_1 is an even cycle, so $\Phi(G_1) = 2$.

Let $m = |E(G)|$. The number of edges added at each step after G_1 is at most two more than the number of vertices added. Hence $m \leq n + 2(k-1)$. It suffices to show that $k - 1 \leq p - 2$. To do this, it suffices to prove that $\Phi(G_i) > \Phi(G_{i-1})$ for $i \geq 2$.

Every added ear in a graded ear decomposition has odd length and hence an even number of internal vertices. These can be matched along the ear. Since G_i arises from G_{i-1} by adding one ear of odd length or two *d*isjoint ears of odd length, every perfect matching in G_{i-1} extends to a perfect matching in G_i. In addition, since G_i is also required to be 1-extendable, it has a perfect matching using an initial edge of an added ear; such a matching is not counted by $\Phi(G_{i-1})$. □

Theorem 9.24. *For $p \geq 2$, an elementary graph with n vertices and exactly p perfect matchings has at most $\frac{n^2}{8} + \frac{3n}{4} + 2p - 4$ edges.*

Proof. Add the maximum number of extendable edges from Lemma 9.23 to the maximum number of free edges from Corollary 9.20. □

Since the coefficient on the quadratic term in this edge bound is $\frac{1}{8}$, while the leading coefficient for p-extremal graphs will be $\frac{1}{4}$, large extremal graphs will not be elementary. This enables us to limit the search for extremal elementary graphs.

Corollary 9.25. *Fix $p \geq 2$. If G is an elementary graph with n vertices, p perfect matchings, and $\frac{n^2}{4} + c_p$ edges, then $n^2 - 6n - 16p + 8c_p + 32 \leq 0$. Thus $n \leq 3 + \sqrt{16p - 8c_p - 23}$.*

Recall that $n_p = \min\{n\colon f(n,p) = \frac{n^2}{4} + c_p\}$. We can bound this threshold using the fact that all the chambers in a spire are elementary graphs.

Corollary 9.26. *For $p \geq 2$, let N_p be the largest even number bounded above by $3 + \sqrt{16p - 8c_p - 23}$. Every elementary graph in $\mathcal{F}_p$ has at most N_p vertices, and*

$$n_p \leq \max\left\{\sum_{i=0}^{k} N_{p_i}\colon \prod_i p_i = p\right\}.$$

Proof. By Corollary 9.25, all elementary graphs with n vertices and $\frac{n^2}{4} + c_p$ edges have at most $3 + \sqrt{16p - 8c_p - 23}$ vertices, and the number of vertices must be even.

Let $G \in \mathcal{F}_p$ be a spire generated by $H_0, \ldots, H_k$. Set $p_i = \Phi(H_i)$. We have observed that $p = \prod_{i=0}^{k} p_i$. Since each H_i is elementary, it has at most N_{p_0} vertices, so G has at most $\sum_{i=1}^{k} N_{p_i}$ vertices. Taking the maximum over all factorizations bounds n_p. □

The lower bound $c_p \geq -(p-1)(p-2)$ given by Dudek and Schmitt [38] implies $N_p \in O(p)$. The construction in Theorem 9.7 shows that c_p is nonnegative. Together with Corollary 9.26, this yields $N_p \in O(\sqrt{p})$. With N_q known for $q < p$, this reduces the determination of the exact value of c_p for a given p to a search over a finite set of graphs.

We close this section by summarizing the results of this and the previous section. The outcome is a systematic approach to classifying all graphs in $\mathcal{F}_p$.

Theorem 9.27. *For an n-vertex graph G in* $\mathcal{F}_p$*,*

1. *G is a spire with chambers* $H_0, \ldots, H_k$ *built over barriers* $Y_0, \ldots, Y_k$*.*
2. *Each* Y_i *is a barrier of maximum size in* H_i*.*
3. *If* $0 \leq i < j \leq k$*, then* $\frac{|Y_i|}{|V(H_i)|} \leq \frac{|Y_j|}{|V(H_j)|}$*.*
4. *Letting* $p_i = \Phi(H_i)$*, there are at most* N_{p_i} *vertices in* H_i*, and* $c(H_i) \leq c_{p_i}$*.*
5. $\Phi(G) = p = \prod_{i=0}^{k} p_i$ *and* $c(G) = c_p \leq \sum_{i=1}^{k} c(H_i)$*.*
6. *If* $p_i = 1$*, then* $H_i \cong K_2$*.*

9.5 Graphs with an Odd Number of Vertices

Since graphs with an odd number of vertices do not have perfect matchings, we generalize $f(n,p)$ to odd n using near-perfect matchings. In this section, n is odd.

Definition 9.28. An *near-perfect matching* in a graph is a matching that covers all but one vertex. Let $\tilde{\Phi}(G)$ denote the number of near-perfect matchings in G. Let $\tilde{f}(n,p)$ denote the maximum number of edges in an n-vertex graph with p near-perfect matchings.

The computation of $\tilde{f}(n,p)$ almost reduces to the computation of $f(n,p)$.

Theorem 9.29. *If n is odd and larger than* n_p*, then*

$$\tilde{f}(n,p) = f(n-1,p) = \frac{(n-1)^2}{4} + c_p.$$

Proof. Since $n > n_p$, we may choose $G \in \mathcal{F}_p$ with $n-1$ vertices. Adding an isolated vertex to G produces a graph with p near-perfect matchings and $f(n-1,p)$ edges. Thus $f(n-1,p) \le \tilde{f}(n,p)$.

Let H be an n-vertex graph having $\tilde{f}(n,p)$ edges and p near-perfect matchings. Adding a new vertex adjacent to every vertex in H produces a graph H' having p perfect matchings and $\tilde{f}(n,p)+n$ edges (there is a one-to-one correspondence between near-perfect matchings in H and perfect matchings in H').

Thus $\tilde{f}(n,p) = |E(H')| - n \le f(n+1,p) - n$. By Theorem 9.2, $n > n_p$ implies $f(n+1,p) = f(n-1,p) + n$. We conclude that $\tilde{f}(n,p) \le f(n-1,p)$, so equality holds. $\square$

Not only is the numerical value of $\tilde{f}(n,p)$ determined by the even case, but also the extremal graphs correspond to extremal graphs in the even case, using the bijection in the proof of Theorem 9.29.

Definition 9.30. Let $\tilde{\mathcal{F}}_p$ be the set of graphs G having $\frac{(|V(G)|-1)^2}{4} + c_p$ edges and exactly p near-perfect matchings.

Corollary 9.31. *For each graph $H \in \tilde{\mathcal{F}}_p$, there is a graph $G \in \mathcal{F}_p$ and a vertex $u \in V(G)$ such that u is adjacent to $V(G) - \{u\}$ and $H \cong G - u$.*

Not every graph in $\mathcal{F}_p$ has a dominating vertex, so there are n-vertex graphs in $\mathcal{F}_p$ that do not arise in this simple way from $(n-1)$-vertex graphs in $\tilde{\mathcal{F}}_p$. The graph $\overline{3K_2}$ has eight perfect matchings (each of the 12 edges appears in two perfect matchings, and each perfect matching has three edges). With $n = 6$, we have $n^2/4 + 3$ edges. We will see that $c_8 = 3$, so $\overline{3K_2} \in \mathcal{F}_p$, but the graph has no dominating vertex. On the other hand, when $n > n_p$, Hetyei-extension of an n-vertex graph in $\mathcal{F}_p$ yields a graph in $\mathcal{F}_p$ with $n+2$ vertices that does have a dominating vertex.

9.6 Constructive Lower Bounds

In this section, we refine the binary expansion construction $B(p)$ of Theorem 9.7 to give improved lower bounds for c_p. Because the barrier is large in $B(p)$, it can be used to increase the excess while multiplying the number of perfect matchings. Recall that $w(m)$ is the number of 1s in the binary expansion of m.

Proposition 9.32. *If p_1 and p_2 are integers with $p_1, p_2 \geq 2$, then $c_{p_1 p_2} \geq c_{p_1} + w(p_2 - 1)$.*

Proof. Let G be a n-vertex graph having $\frac{n^2}{4} + c_{p_1}$ edges and exactly p_1 perfect matchings. Let $H = B(p_2)$, in which the clique is a barrier containing exactly half of the vertices. Let G' be the saturated graph formed by making G a tower above this barrier in H.

By Lemma 9.18, $c(G') = c(G) + c(H) = c_1 + w(p - 1)$. By Lemma 9.9, G' has $p_1 \cdot p_2$ distinct perfect matchings. Therefore, $c_{p_1 p_2} \geq c_{p_1} + w(p_2 - 1)$. □

Corollary 9.33. *If p properly divides p', then $c_{p'} > c_p$.*

The binary expansion construction yields $c_p \geq \log_2 p$ when p is a power of 2. However, when $p - 1$ is a power of 2, it yields only $c_p \geq 1$. To combat this deficiency, we develop further lower bounds using graphs where $|E(G)|$ and $\Phi(G)$ are easy to compute. These constructions properly contain the Hetyei graphs, so the excess is positive. Unfortunately, not every p can be realized as $\Phi(G)$ using these constructions.

Definition 9.34. A *Hetyei list* is a nondecreasing list $d_1, \ldots, d_k$ of positive integers such that $d_i \geq i$ for all i and $d_k = k$. The *nested-degree graph* generated by a Hetyei list $(d_1, \ldots, d_k)$, denoted $\mathrm{Deg}(d_1, \ldots, d_k)$, is the supergraph of the Hetyei graph of order $2k$ in which the edge $\ell_i r_j$ exists if and only if $i \leq d_j$.

Theorem 9.35. *If $G = \mathrm{Deg}(d_1, \ldots, d_k)$ for a Hetyei list $d_1, \ldots, d_k$, then G has a barrier of size k and $\Phi(G) = \prod_{i=1}^{k}(d_i + 1 - i)$,*

Proof. Since $\{r_1, \ldots, r_k\}$ is an independent set, every perfect matching pairs its vertices with $\{\ell_1, \ldots, \ell_k\}$. Also, $\{\ell_1, \ldots, \ell_k\}$ is a barrier of size k in G.

To compute $\Phi(G)$, choose edges to cover vertices in the order $r_1, \ldots, r_k$. When covering r_i, there are $i - 1$ previously matched vertices in $\{\ell_1, \ldots, \ell_k\}$. Since

$$\bigcup_{j=1}^{i-1} N(r_i) \subseteq N(r_i),$$

there are $d_i - i + 1$ choices for the edge to cover r_i. Since $d_i \geq i$ for all i, the process completes a perfect matching in $\prod_{i=1}^{k}(d_i + 1 - i)$ ways. □

When a graph G has a barrier B with half its vertices, the edges in perfect matchings form a bipartite graph with partite sets B and $V(G) - B$, and $G - B$ has no edges. Ostrand [101] proved that if a bipartite graph G has a perfect matching, and $d_1, \ldots, d_k$ is the nondecreasing list of degrees of the vertices in one partite set, then $\Phi(G) \geq \prod_{i=1}^{k} \max\{1, d_i - i + 1\}$ (Hwang [66] gave a simple proof). When the list d is a Hetyei list, the corresponding nested-degree graphs achieve equality in the lower bound.

Example 9.36. If $G = \mathrm{Deg}(k, \ldots, k)$, then $\Phi(G) = k!$ and $c(G) = \binom{k}{2}$. By Stirling's approximation, $c_p \geq \Omega\left(\left(\frac{\ln p}{\ln \ln p}\right)^2\right)$ when $p = k!$.

Definition 9.37. Let $(d_1, \ldots, d_k)$ be a Hetyei list and $\{e_1, \ldots, e_m\}$ be a set of disjoint pairs in $\{1, \ldots, k\}$. The resulting *generalized nested-degree graph*, denoted

$$\mathrm{Gen}\left(d_1, \ldots, d_k; e_1, \ldots, e_m\right),$$

consists of the nested-degree graph $\mathrm{Deg}(d_1, \ldots, d_k)$ plus each edge r_ir_j such that $\{i,j\} = e_t$ for some t.

The *double factorial* of an integer n, denoted $n!!$, is the product of the integers in $\{1, \ldots, n\}$ with the same parity as n. As an empty product, by convention $(-1)!!$ equals 1.

Theorem 9.38. *For a set $\{e_1, \ldots, e_m\}$ of disjoint pairs in $\{1, \ldots, k\}$, let $\mathcal{P}$ denote the family of all subsets of $\{r_ir_j\colon\ \{i,j\} \in \{e_1, \ldots, e_m\}\}$. If $G = \mathrm{Gen}\left(d_1, \ldots, d_k; e_1, \ldots, e_m\right)$, then*

$$\Phi(G) = \sum_{M \in \mathcal{P}} (2|M|-1)!! \prod_{r_i \notin V(M)} (d_i - |\{j < i\colon\ r_j \notin V(M)\}|).$$

Also, if $m \geq 1$, then G has no barrier of size k.

Proof. Every perfect matching in G contains some subset M, where

$$M \subseteq \{r_ir_j\colon\ \{i,j\} \in \{e_1, \ldots, e_m\}\}.$$

To complete a matching, cover the remaining vertices in $\{r_1, \ldots, r_k\}$ in increasing order of subscripts by selecting neighbors in $\{\ell_1, \ldots, \ell_k\}$. The number of ways to do this is $\prod_{r_i \notin V(M)} (d_i - |\{r_j \notin V(M)\colon\ j < i\}|)$, as in the proof of Theorem 9.35. Finally, the $2|M|$ remaining unmatched vertices form a clique and can be matched in $(2|M|-1)!!$ ways. □

Theorem 9.35 is the special case of Theorem 9.38 for $m = 0$. When m is small, there are not many subsets of $\{e_1, \ldots, e_m\}$, and computing $\Phi(G)$ is feasible.

Example 9.39. When $m = \binom{k}{2}$ and $d_i = k$ for all i, the generalized nested-degree graph is K_{2k}, with $(2k-1)!!$ perfect matchings. Thus $c_{(2k-1)!!} \geq k^2 - k$. This yields the lower bound $c_p \geq \Omega\left(\left(\frac{\ln p}{\ln \ln p}\right)^2\right)$ when $p = (2m-1)!!$ for some m.

Examples 9.36 and 9.39 provide our best asymptotic lower bounds but apply only for special values. The generalized nested-degree construction is our most efficient method for finding lower bounds when k and m are small In Section 9.8, we discuss the results of computer search over small cases of these constructions to find explicit lower bounds on c_p when p is small.

9.7 A Conjectured Upper Bound

Dudek and Schmitt conjectured that the complete graph K_{2t} is p-extremal for $p = (2t-1)!!$, giving $c_p = t^2 - t$. We generalize this to conjecture an upper bound for all p. First, a lemma provides motivation. In light of the proof, we call it the "Star-Removal Lemma".

Lemma 9.40. *If $p, k, t \in \mathbb{N}$ satisfy $k \leq 2t$ and $p = k(2t-1)!!$, then $c_p \geq t^2 - t + k - 1$.*

Proof. Let G be the graph obtained from K_{2t+2} by removing $2t+1-k$ edges with a common endpoint x. The vertex x has k neighbors; after choosing one, the rest of the graph is isomorphic to K_{2t}. Thus $\Phi(G) = k(2t-1)!!$. The number of edges in G is $\binom{2t+2}{2} - (2t+1-k)$, which equals $\frac{(2t+2)^2}{4} + t^2 - t + k - 1$. Hence $c_p \geq t^2 - t + k - 1$. □

To reduce the number of perfect matchings from $(2t+1)!!$ to $k(2t-1)!!$, only $2t-1-k$ edges were removed; with each edge deleted, $(2t-1)!!$ perfect matchings were lost. This seems to be the most edge-efficient way to remove perfect matchings, which suggests a conjecture.

Conjecture 9.41. *For $p \in \mathbb{N}$, if integers k and t are defined uniquely by $k(2t-1)!! \leq p < (k+1)(2t-1)!!$ with $k \leq 2t$, then $c_p \leq C_p$, where $C_p = t^2 - t + k - 1$.*

The conjecture matches the lower bound in Lemma 9.40 when $p = k(2t-1)!!$. It also matches the value of c_p for $p \leq 6$ as computed in [38]. In Section 9.8, we verify that C_p also equals c_p for $7 \leq p \leq 10$, and we give empirical evidence that the bound holds for all p.

9.8 Exact Values for Small p

To confirm the values of c_p for $p \leq 6$, we used McKay's `geng` program [92, 93] to generate all graphs on 10 vertices. We checked that none of these graphs have exactly p perfect matchings while achieving larger excess. This yields a proof, since $N_p \leq 10$ for $p \leq 6$ and the smallest graph in $\mathcal{F}_p$ has at most N_p vertices.

For $p \leq 10$, we have $N_p \leq 12$. Generating all graphs on 12 vertices presently is infeasible for us; instead, we use the following lemma.

Lemma 9.42 (Dudek–Schmitt [38, Lemma 2.4]). *If $p \geq 2$, then $c_p \leq 1 + \max_{q<p} c_q$,*

If we know all previous values of c_p, and we construct an n-vertex graph G with $\Phi(G) = p$ and $|E(G)| = \frac{n^2}{4} + C$, where $C = \max\{c_q \colon q < p\}$, then we only need to check graphs with $\frac{n^2}{4} + C + 1$ edges to see whether one has exactly p perfect matchings. Thus our proof of the next theorem is by computer search. It yields the values in Table 9.1.

p	1	2	3	4	5	6	7	8	9	10
c_p	0	1	2	2	2	3	3	3	4	4
n_p	2	4	4	6	6	6	6	6	6	6
N_p		4	6	8	8	10	10	12	12	12
	[38]						Theorem 9.43			

Table 9.1: Excess c_p (at n_p), bound N_p on extremal chambers.

Theorem 9.43. *$c_7 = 3$, $c_8 = 3$, $c_9 = 4$, and $c_{10} = 4$.*

Proof. Explicit constructions in Fig. 9.5 give the lower bounds; we will subsequently describe how these constructions arise.

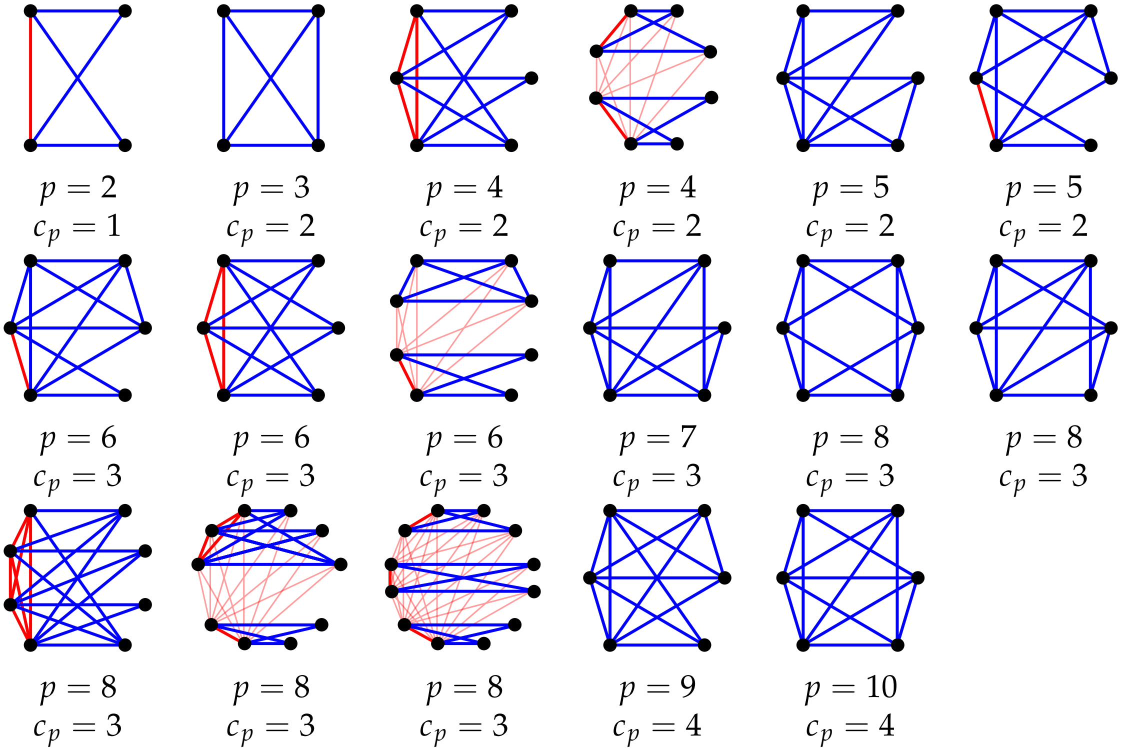
For the first two upper bounds, Lemma 9.42 yields (a) $c_7 \leq 4$ and (b) if $c_7 = 3$, then $c_8 \leq 4$. Thus to show $c_7 = c_8 = 3$ it suffices to examine graphs with 12 vertices and $\frac{12^2}{4} + 4$ edges. Using `geng`, we generated these and found none with exactly seven or eight perfect matchings, so $c_7 = c_8 = 3$.

By Lemma 9.42, $c_9 \leq 4$, and then similarly $c_{10} \leq 5$. To test equality, it suffices to study graphs with 12 vertices and $\frac{12^2}{4} + 5$ edges. Using `geng`, we enumerated these and found no graph with exactly ten perfect matchings, so $c_{10} \leq 4$. □

For small p, the chambers in the p-extremal graphs are instances of the general constructions we have provided in earlier sections. Below we characterize all p-extremal graphs for $p \leq 10$. Fig. 9.5 shows the smallest instances of the classes of graphs in these characterizations. The edge-colorings indicate the decomposition into chambers. Blue edges are extendable; when the subgraph of blue edges is connected, the graph is elementary. Red edges indicate the maximal barriers in chambers. Faint edges join these barriers to the spires over them when the graph is not elementary, in which case the factorization of p should be apparent; recall that $\Phi(G)$ is the product of $\Phi(H_i)$ when the chambers of G are $H_0, \ldots, H_k$.

Hetyei characterized the 1-extremal graphs. Dudek and Schmitt determined c_p for $p \leq 6$ but provided proof only for $p \leq 4$ and characterized the extremal graphs only for $p \leq 3$. We restate these characterizations in the language of the Cathedral Theorem. The "top" of a spire is the chamber last in the list of chambers describing the cathedral decomposition; it is at the other end from the foundation. When G is edge-transitive, G^- denotes the graph obtained by deleting one edge from G.

Theorem 9.44 (Hetyei)**.** *For even n with $n \geq 2$, the unique 1-extremal graph has $\frac{n^2}{4}$*



$p = 2$, $c_p = 1$; $p = 3$, $c_p = 2$; $p = 4$, $c_p = 2$; $p = 4$, $c_p = 2$; $p = 5$, $c_p = 2$; $p = 5$, $c_p = 2$

$p = 6$, $c_p = 3$; $p = 6$, $c_p = 3$; $p = 6$, $c_p = 3$; $p = 7$, $c_p = 3$; $p = 8$, $c_p = 3$; $p = 8$, $c_p = 3$

$p = 8$, $c_p = 3$; $p = 8$, $c_p = 3$; $p = 8$, $c_p = 3$; $p = 9$, $c_p = 4$; $p = 10$, $c_p = 4$

Figure 9.5: The smallest p-extremal configurations, for $2 \leq p \leq 10$

edges and is a spire whose chambers all equal K_2.

Theorem 9.45 ([38])**.** *For even n with $n \geq 4$, the 2-extremal graphs have $\frac{n^2}{4} + 1$ edges and are spires whose chambers are $(n-4)/2$ copies of K_2 and one copy of K_4^-, taken in any order.*

Theorem 9.46 ([38])**.** *For even n with $n \geq 4$, the unique 3-extremal graph has $\frac{n^2}{4} + 2$ edges and is a spire whose chambers are $(n-4)/2$ copies of K_2 and one copy of K_4 at the top.*

Note that K_2 and K_4^- have a barrier containing half of the vertices, while K_4 does not; hence the chambers for $p = 2$ appear in any order, while K_4 is at the top when $p = 3$.

The characterizations of the p-extremal graphs for $4 \leq p \leq 10$ all use the same method and involve the computer search used to prove Theorem 9.43. Instead of repeating the observations for each proof, we outline them here and just state the resulting characterizations.

Outline of Characterization Proofs. A p-extremal graph G is a spire of chambers $H_0, \ldots, H_k$ (Lemma 9.16), and $c(G) \leq \sum_i c(H_i)$ (Lemma 9.18). The number of perfect matchings in G equals $\prod_i \Phi(H_i)$ (Lemma 9.9). Hence to know the p-extremal graphs it suffices to know the p_j-extremal chambers for all p_j that are factors of p and compare the numbers of edges in the spires corresponding to factorizations of p.

The chambers in spires are elementary graphs. Every p-extremal elementary graph has at most N_p vertices, where N_p is the largest even number bounded by $3 + \sqrt{16p - 8c_p - 23}$ (Corollary 9.26). A p-extremal elementary graph with fewer than N_p vertices extends to a p-extremal graph with N_p vertices by Hetyei-extension (repeatedly adding K_2 as a chamber at the beginning of the spire), so the p-extremal chambers are found within the graphs on N_p vertices. The q-extremal chambers for $q < p$ are already known from previous searches.

When searching graphs with N_p vertices for p-extremal chambers, we limit the search to specific numbers of edges. A p-extremal graph with N_p vertices has $\frac{1}{4}N_p^2 + c_p$ edges. By Lemma 9.42, $c_p \leq C + 1$, where $C = \max_{q<p} c_q$. Hence we begin by searching graphs with N_p vertices and excess $C + 1$, looking for those having exactly p perfect matchings. The search moves to excess C if none are found with excess $C + 1$. In the results for $p \leq 10$, graphs with N_p vertices and p perfect matchings were always found having excess C or $C + 1$, so there was no need to search further.

At this point the q-extremal chambers are known for all factors q of p, and hence the complete description of p-extremal graphs can be given. The chambers in a p-extremal spire are q_i-extremal elementary graphs, where $\prod q_i = p$. However, a spire with q_i-extremal chambers may have too few edges to be $\prod q_i$-extremal (for example, the spire with chambers K_4 and K_4 has nine perfect matchings but is not 9-extremal).

The order of chambers in a spire does not affect the number of perfect matchings, but it does affect the number of edges. To have the most edges, the chambers must be listed in decreasing order of the fractions of their vertices occupied by their largest barrier (Lemma 9.17). Spires for which these fractions are equal (such as K_2 and K_4^- having barriers with half their vertices) may be listed in any order. □

See Fig. 9.5 for the smallest instances of the classes of graphs in these characterizations.

Theorem 9.47. *For even n with $n \geq 6$, the 4-extremal graphs have $\frac{n^2}{4} + 2$ edges and are spires whose chambers are*

a) $\frac{n-6}{2}$ copies of K_2 and one copy of $B(4)$ in any order, or

b) $\frac{n-8}{2}$ copies of K_2 and two copies of K_4^- in any order.

Theorem 9.48. *For even n with $n \geq 6$, the 5-extremal graphs have $\frac{n^2}{4} + 2$ edges and are spires whose chambers are $\frac{n-6}{2}$ copies of K_2 plus one 6-vertex graph at the top that is* $\mathrm{Gen}\,(2,2,3;\{1,2\})$ *or* $\mathrm{Gen}\,(2,3,3;\{2,3\})) - \ell_2 r_2$.

Theorem 9.49. *For even n with $n \geq 6$, the 6-extremal graphs have $\frac{n^2}{4} + 3$ edges and are spires whose chambers are*

a) $\frac{n-6}{2}$ copies of K_2 and one copy of $\mathrm{Gen}\,(2,3,3;\{2,3\})$ *at the top,*

b) $\frac{n-6}{2}$ copies of K_2 and one copy of $\mathrm{Deg}(3,3,3)$ *in any order, or*

c) $\frac{n-8}{2}$ copies of K_2 and one copy of K_4^- in any order, plus one copy of K_4 at the top.

Theorem 9.50. *For even n with $n \geq 6$, the unique* 7*-extremal graph has* $\frac{n^2}{4} + 3$ *edges and is a spire whose chambers are* $\frac{n-6}{2}$ *copies of* K_2 *and one* $\mathrm{Gen}\,(2,2,3;\{1,2\},\{1,3\})$ *at the top.*

Theorem 9.51. *For even n with $n \geq 6$, the* 8*-extremal graphs have* $\frac{n^2}{4} + 3$ *edges and are spires whose chambers are a)* $\frac{n-6}{2}$ *copies of* K_2*, plus one copy of* $\overline{3K_2}$ *at the top,*
b) $\frac{n-6}{2}$ *copies of* K_2*, plus one copy of* $\mathrm{Gen}\,(1,3,3;\{1,2\},\{1,3\},\{2,3\})$ *at the top,*
c) $\frac{n-8}{2}$ *copies of* K_2 *and one copy of* $B(8)$ *in any order,*
d) $\frac{n-10}{2}$ *copies of* K_2*, one copy of* K_4^-*, and one copy of* $B(8)$ *in any order, or*
e) $\frac{n-12}{2}$ *copies of* K_2 *and three copies of* K_4^- *in any order.*

Theorem 9.52. *For even n with $n \geq 6$, the unique* 9*-extremal graph has* $\frac{n^2}{4} + 4$ *edges and is a spire whose chambers are* $\frac{n-6}{2}$ *copies of* K_2 *and one* $\mathrm{Gen}\,(3,3,3;\{2,3\})$ *at the top.*

Theorem 9.53. *For even n with $n \geq 6$, the unique* 10*-extremal graph has* $\frac{n^2}{4} + 4$ *edges and is a spire whose chambers are* $\frac{n-6}{2}$ *copies of* K_2 *and one* $\mathrm{Gen}\,(2,3,3;\{1,2\},\{1,3\})$ *at the top.*

Moving beyond $p = 11$, note that $N_{11} = 14$. Unfortunately, the number of graphs with 14 vertices and suitable number of edges is beyond the capacity of our computer resources to determine c_{11} by this method.

In Figure 9.6, we present the lower bounds on c_p found by searching all graphs of order 10 to find chambers and forming spires from these chambers and chambers arising from the generalized nested degree construction on 12, 14, and 16 vertices with $k \in \{5,6,7,8\}$ and $m \in \{4,3,2,1\}$. The upper line is the conjectured upper bound C_p from Conjecture 9.41, defined as $t^2 - t + k - 1$, where t and k are determined by $k(2t-1)!! \leq p < (k+1)(2t-1)!!$ with $k \leq 2t$. As the plot shows, we have found no construction that violates the upper bound, and sometimes it equals the excess of the best construction found so far.

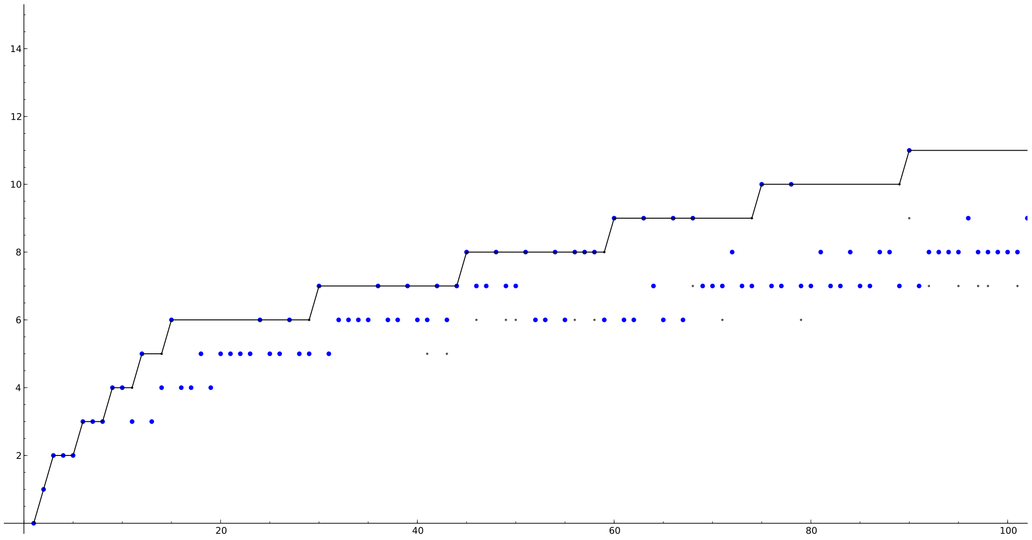


Figure 9.6: Lower bounds on c_p and conjectured upper bound C_p.

9.9 Connection with 2-Connected Graphs

Since it is intractable to check all graphs of order 14, simply using *geng* to check all graphs will not exhaustively find all p-extremal chambers for $p \geq 11$. In order to find these graphs, we will exploit the structure of the edges in perfect matchings, which form 2-connected graphs.

Proposition 9.54. *If H is 1-extendable with $\Phi(H) \geq 2$, then H is 2-connected.*

Proof. Since $\Phi(H) \geq 2$, there are at least four vertices in H. Suppose H was not 2-connected. Then, there exists a vertex $x \in V(H)$ so that $H - x$ has multiple components. Since H has an even number of vertices, at least one component of $H - x$ must have an odd number of vertices. Since H has perfect matchings, Tutte's Theorem implies exactly one such component C has an odd number of vertices. Moreover, in every perfect matching of H, x is matched to some vertex in C. Hence, the edges from x to the other components never appear in perfect matchings, contradicting that H was 1-extendable. □

2-connected graphs are characterized by *ear decompositions*. An *ear* is a path given by vertices $x_0, x_1, \ldots, x_k$ so that x_0 and x_k have degree at least three and x_i has degree exactly two for all $i \in \{1, \ldots, k-1\}$. The vertices x_0 and x_k are *branch vertices* while $x_1, \ldots, x_{k-1}$ are *internal vertices*. In the case of a cycle, the entire graph is considered to be an ear. For an ear ε, the *length* of ε is the number of edges between the endpoints and its *order* is the number of internal vertices between the endpoints. We will focus on the order of an ear. An ear of order 0 (length 1) is a single edge, called a *trivial* ear.

An *ear augmentation* is the addition of a path between two vertices of the graph. This process is invertible: an *ear deletion* takes an ear $x_0, x_1, \ldots, x_k$ in a graph and deletes all vertices $x_1, \ldots, x_{k-1}$ (or the edge x_0x_1 if $k = 1$). For a graph H, an ear augmentation is denoted $H + \varepsilon$ while an ear deletion is denoted $H - \varepsilon$. Every 2-connected graph H has a sequence of graphs $H_1 \subset \cdots \subset H_\ell = H$ so that H_1 is a cycle and for all $i \in \{1, \ldots, \ell - 1\}$, $H^{(i+1)} = H^{(i)} + \varepsilon_i$ for some ear ε_i [147].

Lovász's Two Ear Theorem gives the structural decomposition of 1-extendable graphs using a very restricted type of ear decomposition. A sequence $H_0 \subset H_1 \subset H_2 \subset \cdots \subset H_k$ of ear augmentations is a *graded ear decomposition* if each $H^{(i)}$ is 1-extendable. The decomposition is *non-refinable* if for all $i < k$, there is no 1-extendable graph H' so that $H^{(i)} \subset H' \subset H^{(i+1)}$ is a graded ear decomposition.

Theorem 9.55 (Two Ear Theorem [85]; See also [87, 133]). *If H is 1-extendable, then there is a non-refinable graded ear decomposition $H_0 \subset H_1 \subset \cdots \subset H_k$ so that $H_0 \cong C_{2\ell}$*

for some ℓ and each ear augmentation $H^{(i)} \subset H^{(i+1)}$ uses one or two new ears, each with an even number of internal vertices.

We will consider making single-ear augmentations to build 1-extendable graphs, so we classify the graphs which appear after the first ear of a two-ear augmentation. A graph H is *almost 1-extendable* if the free edges of H appear in a single ear of H. The following corollary is a restatement of the Two Ear Theorem using almost 1-extendable graphs.

Corollary 9.56. *If H is 1-extendable, then there is an ear decomposition $H_0 \subset H_1 \subset \cdots \subset H_k$ so that $H_0 \cong C_{2\ell}$ for some ℓ, each ear augmentation $H^{(i)} \subset H^{(i+1)}$ uses a single ear of even order, each $H^{(i)}$ is either 1-extendable or almost 1-extendable, and if $H^{(i)}$ is almost 1-extendable then H_{i-1} and $H^{(i+1)}$ are 1-extendable.*

An important property of graded ear decompositions is that $\Phi(H^{(i)})$ is strictly smaller than $\Phi(H^{(i+1)})$, since the perfect matchings in $H^{(i)}$ extend to perfect matchings of $H^{(i+1)}$ using alternating paths within the augmented ear(s) and the other edges must appear in a previously uncounted perfect matching.

We use this theorem to develop our search space for the canonical deletion technique, forming the first stage of the search. The second stage adds free edges to a 1-extendable graph with p perfect matchings. The structure of free edges is even more restricted, as shown in the following proposition.

Proposition 9.57 (Theorems 5.2.2 & 5.3.4 [87])**.** *Let G be an elementary graph. An edge e is free if and only if the endpoints are in the same barrier. If adding any missing edge to G increases the number of perfect matchings, then every barrier in G of size at least two is a clique of free edges.*

In Section 9.11, we describe a technique for adding free edges to a 1-extendable graph. In order to better understand the first stage, we investigate what types of ear augmentations are allowed in a non-refinable graded ear decomposition.

Lemma 9.58. *Let $H \subset H + \varepsilon$ be a one-ear augmentation between 1-extendable graphs H and $H + \varepsilon$. The endpoints of ε are in disjoint maximal barriers.*

Proof. If the endpoints of ε were not in disjoint maximal barriers, then they are contained in the same maximal barrier. If an edge were added between these vertices, Proposition 9.57 states that this edge would be free. Since ε is an even subdivision of such an edge, the edges incident to the endpoints are not extendable, making $H + \varepsilon$ not 1-extendable. □

Lemma 9.59. *Let $H \subset H + \varepsilon_1 + \varepsilon_2$ be a non-refinable two ear augmentation between 1-extendable graphs.*

1. *The endpoints of ε_1 are within a maximal barrier of H.*
2. *The endpoints of ε_2 are within a different maximal barrier of H.*

Proof. (1) If the endpoints a, b of ε_1 span two different maximal barriers, adding the edge ab would add an extendable edge by Proposition 9.57. The perfect matchings of $H + ab$ and $H + \varepsilon_1$ would be in bijection depending on if ab was used: if a perfect matching M in $H + ab$ does not contain ab, M extends to a perfect matching in $H + \varepsilon_1$ by taking alternating edges within ε_1, with the edges incident to a and b not used; if M used ab, the alternating edges along ε_1 would use the edges incident to a and b. Hence, $H + \varepsilon_1$ is 1-extendable and this is a refinable graded ear decomposition. This contradiction shows that ε_1 spans vertices within a single maximal barrier.

(2) The endpoints x, y of ε_2 must be within a single maximal barrier by the same proof as (2), since otherwise $H + \varepsilon_2$ would be 1-extendable and the augmentation is refinable. However, if both ε_1 and ε_2 span the same maximal barrier, $H + \varepsilon_1 + \varepsilon_2$ is not 1-extendable. By Proposition 9.57, edges within a barrier are free. Hence, the perfect matchings of $H + \varepsilon_1 + \varepsilon_2$ do not use the internal edges of ε_1 and ε_2 which are incident to their endpoints. This contradicts 1-extendability, so the endpoints of ε_2 are in a different maximal barrier than the endpoints of ε_1. □

9.10 Searching for p-extremal elementary graphs

Given p and c, we aim to generate all elementary graphs G with $\Phi(G) = p$ and $c(G) \geq c$. If $c \leq c_p$, Corollary 9.26 implies $n(G) \leq N_p \leq 3 + \sqrt{16p - 8c - 23}$. In order to discover these graphs, we use the isomorph-free generation algorithm from Chapter 7 to generate 1-extendable graphs with up to p perfect matchings and up to N_p vertices. This algorithm is based on Brendan McKay's canonical deletion technique [92] and generates graphs using ear augmentations while visiting each unlabeled graph only once. This technique will generate 1-extendable graphs and almost 1-extendable graphs. Let $\mathcal{M}^p$ be the set of 2-connected graphs G with $\Phi(G) \in \{2, \dots, p\}$ that are either 1-extendable or almost 1-extendable. $\mathcal{M}^p_{N_p}$ is the set of graphs in $\mathcal{M}^p$ with at most N_p vertices.

The following lemma is immediate from Corollary 9.56.

Lemma 9.60. *For each graph $H \in \mathcal{M}^p$, either H is an even cycle or there exists an ear ε so that $H - \varepsilon$ is in $\mathcal{M}^p$.*

With this property, all graphs in $\mathcal{M}^p_{N_p}$ can be generated by a recursive process: Begin at an even cycle $H_0 = C_{2\ell}$. For each $H^{(i)}$, try adding each all ears ε of order r

to all pairs of vertices in $H^{(i)}$ where $0 \leq r \leq N_p - n(H^i)$ to form $H^{(i+1)} + \varepsilon$. If $H^{(i+1)}$ is 1-extendable or $H^{(i)}$ is 1-extendable and $H^{(i+1)}$ is almost 1-extendable, recurse on $H^{(i+1)}$ until $\Phi(H^{(i+1)}) > p$. While this technique will generate all graphs in $\mathcal{M}^p_{N_p}$, it will generate each *unlabeled* graph several times. In fact, the number of times an unlabeled $H \in \mathcal{M}^p_{N_p}$ appears is *at least* the number of ear decompositions $H_0 \subset \cdots \subset H_k \subset H$ which match the conditions of Corollary 9.56.

We will remove these redundancies in two ways. First, we will augment using pair orbits of vertices in $H^{(i)}$. Second, we will *reject* some augmentations if they do not correspond with a “canonical" ear decomposition of the larger graph. This technique is described in detail in Chapter 7.

Let $\text{del}(H)$ be a function which takes a graph $H \in \mathcal{M}^p$ and returns an ear ε in H so that $H - \varepsilon$ is in $\mathcal{M}^p$. This function $\text{del}(H)$ is a *canonical deletion* if for any two $H_1, H_2 \in \mathcal{M}^p$ so that $H_1 \cong H_2$, there exists an isomorphism $\sigma : H_1 \to H_2$ that maps $\text{del}(H_1)$ to $\text{del}(H_2)$.

Given a canonical deletion $\text{del}(H)$, the *canonical ear decomposition* at H is given by the following iterative construction: (i) Set $H_0 = H$ and $i = 0$. (ii) While $H^{(i)}$ is not a cycle, define $H_{i-1} = H^{(i)} - \text{del}(H^{(i)})$ and decrement i. When this process terminates, what results is an ear decomposition $H_{-k} \subset H_{-(k-1)} \subset \cdots \subset H_{-1} \subset H_0$ where H_{-k} is isomorphic to a cycle and $H_0 = H$.

A simple consequence of this definition is that if $H_{-1} = H - \text{del}(H)$, then the canonical ear decomposition of H begins with the canonical ear decomposition of H_{-1} then proceeds with the augmentation $H_{-1} \subset H_{-1} + \text{del}(H) = H$. Applying isomorph-free generation algorithm of Corollary 9.56 will generate all unlabeled graphs in $\mathcal{M}^p$ without duplication by generating ear decompositions using all possible ear augmentations and rejecting any augmentations which are not isomorphic to the canonical deletion.

In order to guarantee the canonical deletion $\text{del}(H)$ satisfies the isomorphism requirement, the choice will depend on a canonical labeling. A function $\text{lab}(H)$ which takes a labeled graph H and outputs a bijection $\sigma_H : V(H) \to \{1, \ldots, n(H)\}$ is a *canonical labeling* if for all $H_1 \cong H_2$ the map $\pi : V(H_1) \to V(H_2)$ defined as $\pi(x) = \sigma_{H_2}^{-1}(\sigma_{H_1}(x))$ is an isomorphism. The canonical labeling $\sigma_H = \text{lab}(H)$ on the vertex set induces a label γ_H on the ears of H. Given an ear ε of order r between endpoints x and y, let $\gamma_H(\varepsilon) = (r, \min\{\sigma_H(x), \sigma_H(y)\}, \max\{\sigma_H(x), \sigma_H(y)\})$. These labels have a natural lexicographic ordering which minimizes the order of an ear and then minimizes the pair of canonical labels of the endpoints. In this work, the canonical labeling $\text{lab}(H)$ is computed using McKay's `nauty` library [93, 61]. We now describe the canonical deletion $\text{del}(H)$ which will generate a canonical ear decomposition matching Corollary 9.56 whenever given a graph $H \in \mathcal{M}^p$.

By the proof of Lemma 9.60, we need all almost 1-extendable graphs H to have $H - \varepsilon$ be 1-extendable, but 1-extendable graphs H may have $H - \varepsilon$ be 1-extendable or almost 1-extendable, depending on availability. Also, since we are only augmenting by ears of even order, we must select the deletion to have this parity.

The following sequence of choices describe the method for selecting a canonical ear to delete from a graph H in $\mathcal{M}^p_{N_p}$:

1. If H is almost 1-extendable, select an ear ε so that $H - \varepsilon$ is 1-extendable. By the definition of almost 1-extendable graphs, there is a unique such choice.

2. If H is 1-extendable and there exists an ear ε so that $H - \varepsilon$ is 1-extendable, then select such an ear with minimum value $\gamma_H(\varepsilon)$.

3. If H is 1-extendable and no single ear ε has the deletion $H - \varepsilon$ 1-extendable, then select an even-order ear ε so that there is a disjoint even-order ear ε' so

that $H - \varepsilon$ is almost 1-extendable and $H - \varepsilon - \varepsilon'$ is 1-extendable. Out of these choices for ε, select ε with minimum value $\gamma_H(\varepsilon)$.

See Chapter 7 for a more detailed description of the isomorph-free properties of the canonical deletion strategy.

The full generation algorithm, including augmentations, checking canonical deletions, as well as some optimizations and pruning techniques, is described in Section 9.14. We now investigate how to find p-extremal elementary graphs using 1-extendable graphs in $\mathcal{M}^p$. In the following section, we discuss how to fill a 1-extendable graph H with free edges without increasing the number of perfect matchings.

9.11 Structure of Free Subgraphs

By Proposition 9.57, the free edges within an elementary graph have endpoints within a common barrier. This implies that the structure of the free edges is coupled with the structure of barriers in G. In this section, we demonstrate that the structure of the free subgraph of a p-extremal elementary graph depends entirely on the structure of the barriers in the extendable subgraph. This leads to a method to generate all maximal sets of free edges that can be added to a 1-extendable graph. Section 9.12 describes a method for quickly computing the list of barriers of a 1-extendable graph using an ear decomposition. In particular, this provides an on-line algorithm which is implemented along with the generation of canonical ear decompositions. Finally, Section 9.13 combines the results of these sections into a very strict condition which is used to prune the search tree.

Let G be an elementary graph. The *barrier set* $\mathcal{B}(G)$ is the set of all barriers in G. The *barrier partition* $\mathcal{P}(G)$ is the set of all maximal barriers in G. The following lemmas give some properties of $\mathcal{P}(G)$ and $\mathcal{B}(G)$ when G is elementary.

Lemma 9.61 (Lemma 5.2.1 [87]). *For an elementary graph G, the set $\mathcal{P}(G)$ of maximal barriers in G is a partition of $V(G)$.*

Lemma 9.62 (Theorem 5.1.6 [87]). *For an elementary graph G and $B \in \mathcal{B}(G)$, $B \neq \emptyset$, all components of $G - B$ have odd order.*

Given an elementary graph H, let $\mathcal{E}(H)$ be the set of elementary supergraphs with the same extendable subgraph:

$$\mathcal{E}(H) = \{G \supseteq H : V(G) = V(H), \Phi(G) = \Phi(H)\}.$$

We will refer to maximal elements of $\mathcal{E}(H)$ using the subgraph relation $\subseteq$, giving a poset $(\mathcal{E}(H), \subseteq)$. Note that $(\mathcal{E}(H), \subseteq)$ has a unique minimal element, H.

Proposition 9.63. *Let H be a 1-extendable graph. If G is a maximal element in $\mathcal{E}(H)$, then every barrier in $\mathcal{P}(G)$ is a clique of free edges in G.*

Proof. If some maximal barrier X in $\mathcal{P}(G)$ is not a clique, then there is a missing edge e between vertices x, y of X. Since $|X| = \text{odd}(G - X)$, all perfect matchings of $G + e$ must match at least one vertex of each odd component to some vertex in X, saturating X. This means that e is not extendable in $G + e$, and $G + e \in \mathcal{E}(H)$. This contradicts that G was maximal in $\mathcal{E}(H)$.

By Proposition 9.57, the edges within the barriers are free. □

Lemma 9.64. *Let H be a 1-extendable graph and $G \in \mathcal{E}(H)$ be an elementary supergraph of H. Every barrier B of G is also a barrier of H.*

Proof. Each odd component of $G - B$ is a combination of components of $H - B$, an odd number of which are odd components, giving $\text{odd}(H - B) \geq \text{odd}(G - B)$.

There are no new vertices in G, so the components of $G - B$ partition $V(H) - B$ so that the partition of components of $H - B$ is a refinement of $G - B$.

Since B is a barrier of G, $\text{odd}(G - B) = |B|$. Since H is matchable, Tutte's Theorem implies $\text{odd}(H - B) \leq |B|$. Thus $|B| = \text{odd}(G - B) \leq \text{odd}(H - B) \leq |B|$ and equality holds, making B a barrier of H. $\square$

Given a 1-extendable graph H, barriers B_1 and B_2 *conflict* if (a) $B_1 \cap B_2 \neq \emptyset$, (b) B_1 spans multiple components of $H - B_2$, or (c) B_2 spans multiple components of $H - B_1$. A set $\mathcal{I}$ of barriers in $\mathcal{B}(H)$ is a *cover set* if each pair B_1, B_2 of barriers in $\mathcal{I}$ are non-conflicting and $\mathcal{I}$ is a partition of $V(H)$. Let $\mathcal{C}(H)$ be the family of cover sets in $\mathcal{B}(H)$. If $\mathcal{I}_1, \mathcal{I}_2 \in \mathcal{C}(H)$ are cover sets, let the relation $\mathcal{I}_1 \preceq \mathcal{I}_2$ hold if for each set $B_1 \in \mathcal{I}_1$ there exists a set $B_2 \in \mathcal{I}_2$ so that $B_1 \subseteq B_2$. This defines a partial order on $\mathcal{C}(H)$ and the poset $(\mathcal{C}(H), \preceq)$ has a unique minimal element given by the partition of $V(H)$ into singletons.

Theorem 9.65. *Let H be a 1-extendable graph. A graph $G \in \mathcal{E}(H)$ is maximal in $(\mathcal{E}(H), \subseteq)$ if and only if each $B \in \mathcal{P}(G)$ is a clique, $\mathcal{P}(G)$ is a cover set, and $\mathcal{P}(G)$ is maximal in $(\mathcal{C}(H), \preceq)$.*

Proof. We define a bijection between $\mathcal{C}(H)$ and a subset of elementary supergraphs in $\mathcal{E}(H)$, as given in the following claim.

Claim 9.66. *Cover sets $\mathcal{I} \in \mathcal{C}(H)$ are in bijective correspondence with elementary graphs $G \in \mathcal{E}(H)$ where the free subgraph of G is a disjoint union of cliques, each of which is a (not necessarily maximal) barrier of G.*

Let G be a graph in $\mathcal{E}(H)$ where the free subgraph of G is a disjoint union of cliques $X_1, X_2, \dots, X_k$, where each X_i is a barrier of G. Then, let $\mathcal{I} = \{X_1, \dots, X_k\}$ be the set of barriers. Note that $\mathcal{I}$ is a partition of $V(H)$, each part of which is a barrier of G which is a barrier of H by Lemma 9.64. Consider two barriers $B_1, B_2 \in \mathcal{I}$. Since we selected $\mathcal{I}$ to be a partition, $B_1 \cap B_2 = \emptyset$. If B_2 spans multiple components of $H - B_1$, then the vertices from these components are a single component in $G - B_1$, where B_2 is a clique of edges. However, Lemma 9.62 gives that all components of $H - B_1$ and $G - B_1$ are odd, since B_1 is a barrier. This implies that $|B_1| = \text{odd}(H - B_1) > \text{odd}(G - B_1) = |B_1|$, a contradiction. Hence, B_2 is contained within a single component of $H - B_1$, so B_1 and B_2 do not conflict in H. This gives that $\mathcal{I}$ is a cover set in $\mathcal{C}(H)$.

This map from $G \in \mathcal{E}(H)$ to $\mathcal{I} \in \mathcal{C}(H)$ is invertible by taking a cover set $\mathcal{I} \in \mathcal{C}(H)$ and filling each barrier $B \in \mathcal{I}$ with edges, forming a graph $H_{\mathcal{I}}$. Since each pair of barriers B_1, B_2 in $\mathcal{I}$ are non-conflicting, the components of $H - B_1$ do not change by adding edges between vertices in B_2. Therefore, each set $B \in \mathcal{I}$ is also a barrier in $H_{\mathcal{I}}$. By Proposition 9.57, the edges within each barrier of $H_{\mathcal{I}}$ are free, so all extendable edges of $H_{\mathcal{I}}$ are exactly those in H. This gives that $\Phi(H_{\mathcal{I}}) = \Phi(H)$ and $H_{\mathcal{I}} \in \mathcal{E}(H)$. The map from $\mathcal{I}$ to $H_{\mathcal{I}}$ is the inverse of the earlier map from $G \in \mathcal{E}(H)$ with free edges forming disjoint cliques to $\mathcal{I} \in \mathcal{C}(H)$. Hence, this is a bijection, proving the claim.

An important point in the previous claim is that the free edges formed cliques which are barriers, but those cliques were not necessarily *maximal* barriers. We now show that the above bijection maps edge-maximal graphs in $\mathcal{E}(H)$ to maximal cover sets in $\mathcal{C}(H)$.

Claim 9.67. *Let $\mathcal{I}$ be a cover set in $\mathcal{C}(H)$. The following are equivalent:*

(i) *$\mathcal{I}$ is maximal in $(\mathcal{C}(H), \preceq)$.*

(ii) *$H_{\mathcal{I}}$ is maximal in $(\mathcal{E}(H), \subseteq)$.*

(iii) *$\mathcal{P}(H_{\mathcal{I}}) = \mathcal{I}$.*

(ii) $\Rightarrow$ (iii) This is immediate from Proposition 9.63.

(iii) $\Rightarrow$ (ii) If $\mathcal{P}(H_{\mathcal{I}})$, then any edge $e \notin E(H_{\mathcal{I}})$ must span two maximal barriers. By Proposition 9.57, e is allowable in $H_{\mathcal{I}} + e$, so $H_{\mathcal{I}}$ is maximal in $(\mathcal{E}(H), \subseteq)$.

(i) $\Rightarrow$ (ii) Let $\mathcal{I}$ be a maximal cover set of barriers in $\mathcal{B}(H)$ and $H_{\mathcal{I}}$ the corresponding elementary supergraph in $\mathcal{E}(H)$. Suppose there exists a supergraph $H' \supset H_{\mathcal{I}}$ in $\mathcal{E}(H)$. Then, there is an edge e in $E(H') \setminus E(H_{\mathcal{I}})$ so that e is free in $H_{\mathcal{I}} + e$. This implies that e spans vertices within the same barrier B of $H_{\mathcal{I}} + e$ (by Proposition 9.57), and B is also a barrier of $H_{\mathcal{I}}$. However, B is split into k barriers $B_1, \ldots, B_k$ in $\mathcal{I}$, for some $k \geq 2$. Therefore, the set $\mathcal{I}' = (\mathcal{I} \setminus \{B_1, \ldots, B_k\}) \cup \{B\}$ is a refinement of $\mathcal{I}$.

We now show that $\mathcal{I}'$ is a cover set in $\mathcal{C}(H)$. Note that any two barriers $X_1, X_2 \in \mathcal{I}'$ where neither is equal to B is still non-conflicting. For any barrier $X \neq B$ in $\mathcal{I}'$, notice that X does not span more than one component of $H - B$, since B is a barrier in $H_{\mathcal{I}}$ and $H_{\mathcal{I}'}$. Also, if B spanned multiple components of $H - X$, then those components would be combined in $H_{\mathcal{I}'} - X$, but since X is a barrier, $|X| = \text{odd}(H_{\mathcal{I}'} - X) \leq \text{odd}(H - X) = |X|$. Therefore, B does not conflict with any other barrier X in $\mathcal{I}'$ giving $\mathcal{I}'$ is a cover set and $\mathcal{I} \preceq \mathcal{I}'$. This contradicts maximality of $\mathcal{I}$, so $H_{\mathcal{I}}$ is maximal.

(ii) $\Rightarrow$ (i) By (iii), $\mathcal{I} = \mathcal{P}(H_{\mathcal{I}})$. Let $\mathcal{I}'$ be a cover set so that $\mathcal{I} \preceq \mathcal{I}'$. $\mathcal{I}'$ also partitions $V(H)$, so $\mathcal{P}(H_{\mathcal{I}})$ is a refinement of $\mathcal{I}'$. Then, the graph $H_{\mathcal{I}'}$ is a proper supergraph of G. By the maximality of G, $H_{\mathcal{I}'}$ must not be an elementary supergraph in $\mathcal{E}(H)$. By the bijection of Claim 9.66, $\mathcal{I}'$ must not be a cover set of H. Therefore, $\mathcal{I}$ is a maximal covering set in $\mathcal{C}(H)$.

This proves the claim and the theorem follows. $\square$

The previous theorem provides a method to search for the maximum elements of $\mathcal{E}(H)$ by generating all cover sets $\{B_1, \dots, B_k\}$ in $\mathcal{C}(H)$ and maximizing the sum $\sum_{i=1}^{k} \binom{|B_i|}{2}$.

The naïve independent set generation algorithm runs with an exponential blow-up on the number of barriers. This can be remedied in two ways. First, we notice empirically that the number of barriers frequently drops as more edges and ears are added, especially for dense extendable graphs. Second, the number of barriers is largest when the graph is bipartite, as there are exactly two maximal barriers each containing half of the vertices, with many subsets which are possibly barriers. We directly adress the case when H is bipartite as there are exactly two maximum elements of $\mathcal{E}(H)$.

Lemma 9.68 (Corollary 5.2 [63])**.** *The maximum number of free edges in an elementary graph with n vertices is* $\binom{n/2}{2}$.

Not only is this a general bound, but it is attainable for bipartite graphs. In a bipartite graph H, there are exactly two graphs in $\mathcal{E}(H)$ which attain this number of free edges.

Lemma 9.69. *If H is a bipartite 1-extendable graph, then there are exactly two maximal barriers, X_1 and X_2. Also, there are exactly two maximum elements G_1, G_2 of $\mathcal{E}(H)$. Each graph G_i is given by adding all possible edges within X_i.*

Proof. Let X_1 and X_2 be the two sides of the bipartition of H. Since H is matchable, $|X_1| = |X_2|$ and $V(H - X_1) = X_2$ and $V(H - X_2) = X_1$. Thus X_1 and X_2 are both barriers which partition $V(H)$ and by Lemma 9.61 these must be the maximal barriers of H.

The sets $\mathcal{I}_1 = \{X_1\} \cup \{\{v\} : v \in X_2\}$ and $\mathcal{I}_2 = \{X_2\} \cup \{\{v\} : v \in X_1\}$ are maximal cover sets in $\mathcal{C}(H)$. Using the bijection of Theorem 9.65, $\mathcal{I}_1$ corresponds with the maximal elementary graph G_1 in $\mathcal{E}(H)$ where all possible edges are added to X_1. Similarly, $\mathcal{I}_2$ corresponds to adding all possible edges to X_2, producing G_2. Each of these graphs has $\binom{n(H)/2}{2}$ free edges, the maximum possible for graphs in $\mathcal{E}(H)$ by Lemma 9.68.

We must show that any other graph G in $\mathcal{E}(H)$ has fewer free edges. We again use the bijection of Theorem 9.65 in order to obtain a maximal cover set $\mathcal{I}$ in $\mathcal{B}(H)$ which are filled with free edges in G. Then, the number of free edges in G is given by $s(\mathcal{I}) = \sum_{B\in\mathcal{I}} \binom{|B|}{2}$.

Without loss of generality, the barrier A of largest size within $\mathcal{I}$ is a subset of X_1. For convenience, we use $m = n(H)/2$ to be the size of each part X_1, X_2 and $k = |A|$, with $1 \leq k < m$. Note that in $H_{\mathcal{I}}$, no free edges have endpoints in both A and $X_1 \setminus A$, leaving at least $k(m-k) = mk - k^2$ fewer free edges within X_1 in G than in G_1. If $H_{\mathcal{I}}$ has $\binom{n(H)/2}{2}$ edges, then the barriers in X_2 add at least $mk - k^2$ free edges.

The problem of maximizing $s(\mathcal{I})$ over all maximal cover sets can be relaxed to a linear program with quadratic optimization function as follows: First, fix the barriers of $\mathcal{I}$ within X_1, including the largest barrier, A. Then, fix the number of barriers of $\mathcal{I}$ within X_2 to be some integer ℓ. Then, let $\{B_1, \ldots, B_\ell\}$ be the list of barriers in X_2. Now, create variables $x_i = |B_i|$ for all $i \in \{1, \ldots, \ell\}$. The barriers in X_1 are fixed, so to maximize $s(\mathcal{I})$, we must maximize $\sum_{i=1}^{\ell} \binom{x_i}{2}$.

We now set some constraints on the x_i. Since the barriers B_i are not empty, we require $x_i \geq 1$. Since B_i does not conflict with A, each B_i is within a single component of $H - A$. Since there are $|A|$ such components, there are at least $|A| - 1$ other vertices in X_2 that are not in B_i, giving $x_i \leq m - k + 1$. Also, since A is the largest barrier, $x_i \leq k$. Finally, the barriers B_i partition X_2, giving $\sum_{i=1}^{\ell} x_i = m$ and that there are is at least one barrier per component, giving $\ell \geq k$.

Since for $x < y$, $\binom{x-1}{2} + \binom{y+1}{2} > \binom{x}{2} + \binom{y}{2}$, optimium solutions to this linear program have maximum value when the maximum number of variables have maximum feasible value. Suppose $1 \leq x_i \leq t$ are the tightest bounds on the variables $x_1, \ldots, x_\ell$. Then $\frac{m-\ell}{t-1}\binom{t}{2}$ is an upper bound on the value of the system.

Case 1: Suppose $k \geq m-k+1$. Now, the useful constraints are $\sum_{i=1}^{\ell} x_i = m, 1 \leq x_i \leq m-k+1$ and we are trying to maximize $\sum_{i=1}^{\ell}\binom{x_i}{2}$. The optimum value is bounded by $\frac{m-\ell}{m-k}\binom{m-k+1}{2}$. As a function of ℓ, this bound is maximized by the smallest feasible value of ℓ, being $\ell = k$. Hence, we have an optimum value at most $\frac{m-k}{m-k}\frac{(m-k+1)(m-k)}{2} = \frac{1}{2}m\,(m+1) - \left(m+\frac{1}{2}\right)k - \frac{1}{2}k^2$. Since $k \geq m-k+1$, the inequality $k \geq \frac{1}{2}(m+1)$ holds, and the optimum value of this program is at most

$$\frac{1}{2}m\left(m+\frac{1}{2}\right) - \left(m+\frac{1}{2}\right)k - \frac{1}{2}k^2 \leq \underbrace{mk}_{k\geq\frac{1}{2}(m+1)} - \underbrace{k^2}_{k\leq m} - \frac{1}{2}k^2 < mk - k^2.$$

Therefore, $H_{\mathcal{I}}$ must not have $\binom{n(H)/2}{2}$ free edges.

Case 2: Suppose $k < m-k+1$. The constraints are now $\sum_{i=1}^{\ell} x_i = m, 1 \leq x_i \leq k$ while maximizing $\sum_{i=1}^{\ell}\binom{x_i}{2}$. This program has optimum value bounded above by $\frac{m-\ell}{k-1}\binom{k}{2}$, which is maximized by the smallest feasible value of ℓ. If $m/k > k$ and $\ell < m/k$, the program is not even feasible, as a sum of ℓ integers at most k could not sum to m. Hence, $\ell \geq \max\{k, m/k\}$.

Case 2.i: Suppose $k \geq m/k$. Setting $\ell = k$ gives a bound of $\frac{m-k}{k-1}\binom{k}{2} = \frac{1}{2}(mk - k^2)$. This is clearly below $mk - k^2$, so $H_{\mathcal{I}}$ does not have $\binom{n(H)/2}{2}$ free edges.

Case 2.ii: Suppose $k < m/k$. Setting $\ell = \lceil m/k \rceil$ gives a bound of $\frac{m-\lceil m/k \rceil}{k-1}\binom{k}{2} = \frac{1}{2}(mk - m)$. Since $k < m/k$, $k^2 < m$ and $\frac{1}{2}(mk-m) \leq mk - k^2$. Hence, $H_{\mathcal{I}}$

does not have $\binom{n(H)/2}{2}$ free edges. $\square$

Experimentation over the graphs used during the generation algorithm for p-extremal graphs shows that a naïve generation of cover sets in $\mathcal{C}(H)$ is sufficiently fast to compute the maximum excess in $\mathcal{E}(H)$ when the list of barriers $\mathcal{B}(H)$ is known. The following section describes a method for computing $\mathcal{B}(H)$ very quickly using the canonical ear decomposition.

9.12 The Evolution of Barriers

In this section, we describe a method to efficiently compute the barrier list $\mathcal{B}(H)$ of a 1-extendable graph H utilizing a graded ear decomposition. Consider a non-refinable graded ear decomposition $H_0 \subset H_1 \subset H_2 \subset \cdots \subset H_k = H$ of a 1-extendable graph H starting at a cycle $C_{2\ell} = H_0$. Not only are the maximal barriers of $C_{2\ell}$ easy to compute (the sets X, Y forming the bipartition) but also the barrier list (every non-empty subset of X and Y is a barrier).

Lemma 9.70. *Let $H^{(i)} \subset H^{(i+1)}$ be a non-refinable ear decomposition of a 1-extendable graph $H^{(i+1)}$ from a 1-extendable graph $H^{(i)}$ using one or two ears. If B' is a barrier in $H^{(i+1)}$, then $B = B' \cap V(H)$ is a barrier in $H^{(i)}$.*

Proof. There are $|B'|$ odd components in $H^{(i+1)} - B'$. There are at most $|B|$ odd components in $H^{(i)} - B$, which may combine when the ear(s) are added to make $H^{(i+1)}$.

Let $x_1, x_2, \ldots, x_r$ be the vertices in $B' \setminus B$. Each x_i is not in $V(H^{(i)})$ so it is an internal vertex of an augmented ear. Therefore, x_i has degree two in $H^{(i+1)}$, so removing x_i from $H^{(i+1)} - (B \cup \{x_1, \ldots, x_{i-1}\})$ increases the number of odd components by at most one. Hence, the number of odd components of $H^{(i+1)} - B'$ is at most the number of odd components of $H^{(i)} - B$ plus the number of vertices in $B' \setminus B$. These combine to form the inequalities

$$\begin{aligned} |B'| &= \operatorname{odd}(H^{(i+1)} - B') \\ &\leq \operatorname{odd}(H^{(i+1)} - B) + |B' \setminus B| \\ &\leq \operatorname{odd}(H^{(i)} - B) + |B' \setminus B| \\ &\leq |B| + |B' \setminus B| = |B'|. \end{aligned}$$

Equality holds above, so B is a barrier in $H^{(i)}$. □

As one-ear augmentations and two-ear augmentations are applied to each $H^{(i)}$, we update the list $\mathcal{B}(H^{(i+1)})$ of barriers in $H^{(i+1)}$ using the list $\mathcal{B}(H^{(i)})$ of barriers in $H^{(i)}$.

Lemma 9.71. *Let B be a barrier of a 1-extendable graph $H^{(i)}$. Let $H^{(i)} \subset H^{(i+1)}$ be a 1-extendable ear augmentation of $H^{(i)}$ using one (ε_1) or two ($\varepsilon_1, \varepsilon_2$) ears.*

1. *If any augmenting ear connects vertices from different components of $H^{(i)} - B$, then B is not a barrier in $H^{(i+1)}$, and neither is any $B' \supset B$ where $B = B' \cap V(H^{(i)})$.*
2. *Otherwise, if B does not contain any endpoint of the augmented ear(s), then B is a barrier of $H^{(i+1)}$, but $B \cup S$ for any non-empty subset $S \subseteq V(H^{(i+1)}) \setminus V(H^{(i)})$ is not a barrier of $H^{(i+1)}$.*
3. *If B contains both endpoints of some ear ε_i, then B is not a barrier in $H^{(i+1)}$ and neither is any $B' \supset B$.*
4. *If B contains exactly one endpoint (x) of one of the augmented ears (ε_j), then*
    a) *B is a barrier of $H^{(i+1)}$.*
    b) *For $S \subseteq V(H^{(i+1)}) \setminus V(H^{(i)})$, $B \cup S$ is a barrier of $H^{(i+1)}$ if and only if S contains only internal vertices of ε_j of even distance from x along ε_j.*

5. *If $B = \emptyset$, then for any subset $S \subseteq V(H^{(i+1)}) \setminus V(H^{(i)})$ $B \cup S$ is a barrier of $H^{(i+1)}$ if and only if the vertices in S are on a single ear ε_j and the pairwise distances along ε_j are even.*

Proof. Let B' be a barrier in $H^{(i+1)}$. Lemma 9.70 gives $B = B' \cap V(H^{(i)})$ is a barrier of $H^{(i)}$, and $H^{(i)} - B$ has $|B|$ odd connected components. Thus, the barriers of $H^{(i+1)}$ are built from barriers B in $H^{(i)}$ and adding edges from the new ear(s).

Case 1: If an ear ε_j spans two components of $H^{(i)} - B$, then the number of components in $H^{(i+1)} - B$ is at most $|B| - 2$. Any vertices from ε_j added to B can only increase the number of odd components by at most one at a time, but also increases the size of B by one. Hence, vertices in $V(H^{(i+1)}) \setminus V(H^{(i)})$ can be added to B to form a barrier in $H^{(i+1)}$.

Case 2: If each ear ε_j spans points in the same components of $H^{(i)} - B$, then the number of odd connected components in $H^{(i+1)} - B$ is the same as in $H^{(i)} - B$, which is $|B|$. Hence, B is a barrier of $H^{(i+1)}$. However, adding a single vertex from any e_i does not separate any component of $H^{(i+1)} - B$, but adds a count of one to $|B|$. Adding any other vertices from ε_j to B can only increase the number of components by one but increases $|B|$ by one. Hence, no non-empty set of vertices from the augmented ears can be added to B to form a barrier of $H^{(i+1)}$.

Case 3: Suppose B contains both endpoints of an ear ε_j. If ε_j is a trivial ear, then it is an extendable edge. If $B' \supseteq B$ is a barrier in $H^{(i+1)}$, this violates Proposition 9.57 which states edges within barriers are free edges. If ε_j has internal vertices, they form an even component in $H^{(i+1)} - B$. By Lemma 9.62, this implies that B is not a barrier. Any addition of internal vertices from ε_j to form $B' \supset B$ will add at most one odd component each, but leave an even component in $H^{(i+1)} - B'$. It follows that no such B' is a barrier in $H^{(i+1)}$.

Case 4: Note that If an ear ε_j has an endpoint within B, then in $H^{(i+1)} - B$, the internal vertices of ε_j are attached to the odd component of $H^{(i+1)} - B$ containing the opposite endpoint. Since there are an even number of internal vertices on ε_j, then $H^{(i+1)} - B$ has the same number of odd connected components as $H^{(i)} - B$, which is $|B|$. Hence, B is a barrier in $H^{(i+1)}$.

Let the ear ε_j be given as a path of vertices $x_0x_1x_2\dots x_k$, where $x_0 = x$ and x_k is the other endpoint of ε_j. Let S be a subset of $\{x_1, \dots, x_{k-1}\}$, the internal vertices of ε_j. The number of components given by removing S from the path $x_1x_2\cdots x_{k-1}x_k$ is equal to the number of *gaps* in S: the values a so that x_a is in S and x_{a+1} is not in S. These components are all odd if and only if for each x_a and $x_{a'}$ in S, $|a - a'|$ is even. Thus, $B \cup S$ is a barrier in $H^{(i+1)}$ if and only if S is a subset of the internal vertices which are an even distance from x_0. □

Lemma 9.71 describes all the ways a barrier $B \in \mathcal{B}(H)$ can extend to a barrier $B' \in \mathcal{B}(H + \varepsilon_1)$ or $B' \in \mathcal{B}(H + \varepsilon_1 + \varepsilon_2)$. Note that the barriers B' which use the internal vertices of ε_1 are independent of those which use the internal vertices of ε_2, unless one of the ears spans multiple components of $H + \varepsilon_1 + \varepsilon_2 - B'$. This allows us to define a *pseudo-barrier list* $\mathcal{B}(H + \varepsilon)$ for almost 1-extendable graphs $H + \varepsilon$, where H is 1-extendable. During the generation algorithm, we consider a single-ear augmentation $H^{(i)} \subset H^{(i)} + \varepsilon_i = H^{(i+1)}$. Regardless of if $H^{(i)}$ or $H^{(i+1)}$ is almost 1-extendable, we can update $\mathcal{B}(H^{(i+1)})$ by taking each $B \in \mathcal{B}(H^{(i)})$ and adding each $B \cup S$ that satisfies Lemma 9.71 to $\mathcal{B}(H^{(i+1)})$. This process generates all barriers $B' \in \mathcal{B}(H^{(i+1)})$ so that $B' \cap V(H^{(i)}) = B$, so each barrier is generated exactly once.

In addition to updating the barrier list in an ear augmentation $H^{(i)} \subset H^{(i+1)}$, we determine the conflicts between these barriers.

Lemma 9.72. *Let $H^{(i)} \subset H^{(i+1)}$ be a 1-extendable ear augmentation using one (ε_1) or two $(\varepsilon_1, \varepsilon_2)$ ears. Suppose B_1' and B_2' are barriers in $H^{(i+1)}$ with barriers $B_1 = V(H^{(i)}) \cap B_1'$ and $B_2 = V(H^{(i)}) \cap B_2'$ of $H^{(i)}$. The barriers B_1' and B_2' conflict in $H^{(i+1)}$ if and only if one of the following holds: (1) $B_1' \cap B_2' \neq \emptyset$, (2) B_1 and B_2 conflict in $H^{(i)}$, or (3) B_1' and B_2' share vertices in some ear (ε_j), with vertices $x_0x_1x_2\ldots x_k$, and there exist indices $0 \leq a_1 < a_2 < a_3 < a_4 \leq k$ so that $x_{a_1}, x_{a_3} \in B_1'$ and $x_{a_2}, x_{a_4} \in B_2'$.*

Proof. Note that by definition, if $B_1' \cap B_2' \neq \emptyset$, then B_1' and B_2' conflict. We now assume that $B_1' \cap B_2' = \emptyset$.

If B_1 or B_2 conflict in $H^{(i)}$, then without loss of generality, B_2 has vertices in multiple components of $H^{(i)} - B_1$. Since B_1' is a barrier in $H^{(i+1)}$, Lemma 9.71 gives that no ear ε_j spans multiple components of $H^{(i)} - B_1$, and the components of $H^{(i)} - B_1$ correspond to components of $H^{(i+1)} - B_1$. Hence, B_2 also spans multiple components of $H^{(i+1)} - B_1$ and B_1' and B_2' conflict in $H^{(i+1)}$.

Now, consider the case that the disjoint barriers B_1 and B_2 did not conflict in $H^{(i)}$. Since B_1 and B_2 are barriers of $H^{(i)}$, then the vertices in $B_1' \setminus B_1$ are limited to one ear ε_{j_1} of the augmentation, and similarly the vertices of $B_2' \setminus B_2$ are within a single ear ε_{j_2}. Since B_1 and B_2 do not conflict, all of the vertices within B_2 lie in a single component of $H^{(i)} - B_1$: the component containing the ear ε_{j_1}. Similarly, the vertices of B_1 are contained in the component of $H^{(i)} - B_2$ that contains the endpoints of ε_{j_2}.

The components of $H^{(i+1)} - B_1$ are components in $H^{(i+1)} - B_1'$ except the component containing the ear ε_{j_1} is cut into smaller components for each vertex in ε_{j_1} and B_1'. In order to span these new components, B_2' must have a vertex within ε_{j_1}.

Therefore, the ears ε_{j_1} and ε_{j_2} are the same ear, given by vertices $x_0, x_1, \ldots, x_k$.

Suppose there exist indices $0 \le a_1 < a_2 < a_3 < a_4 \le k$ so that x_{a_1} and x_{a_3} are in B_1' and x_{a_2} and x_{a_4} are in B_2'. Then, the vertices x_{a_1} and x_{a_3} of B_1' are in different components of $H^{(i+1)} - B_2'$, since every path from x_{a_3} to x_{a_1} in $H^{(i+1)}$ passes through one of the vertices x_{a_2} or x_{a_4}. Hence, B_1' and B_2' conflict.

If B_1' and B_2' do not admit such indices $a_1, \ldots, a_4$, then listing the vertices x_0, x_1, x_2, …, x_k in order will visit those in B_1' and B_2' in two contiguous blocks. In $H^{(i+1)} - B_1'$, the block containing the vertices in B_2' remain connected to the endpoint closest to the block, and hence B_2' will not span more than one component of $H^{(i+1)} - B_1'$. Similarly, B_1' will not span more than one component of $H^{(i+1)} - B_2'$. B_1' and B_2' do not conflict in this case. □

The following corollary is crucial to the bound in Lemma 9.74.

Corollary 9.73. *Let $H^{(i)} \subset H^{(i+1)}$ be a 1-extendable ear augmentation using one (ε_1) or two $(\varepsilon_1, \varepsilon_2)$ ears. Let $\mathcal{I}$ be a maximal cover set in $\mathcal{C}(H^{(i+1)})$ and S be the set of internal vertices x of an ear ε_j such that the barrier in $\mathcal{I}$ containing x has at least one vertex in $V(H^{(i)})$. Then, S contains at most half of the internal vertices of ε_j.*

Proof. Let $A' \subset \mathcal{I}$ be the set of barriers containing a vertex x in ε_j and a vertex y in $V(H^{(i)})$. For a barrier B to contain an internal vertex of ε_j and a vertex in $V(H^{(i)})$, Lemma 9.71 states that B must contain at least one of the endpoints of the ear ε_j. Since each barrier in A' contains and endpoint of ε_j and non-conflicting barriers are non-intersecting, there are at most two barriers in A'.

If there is exactly one barrier B in A', by Lemma 9.71 it must contain vertices an even distance away from the endpoint contained in B, and hence at most half of the internal vertices of ε_j are contained in B.

If there are two non-conflicting barriers B_1 and B_2 in A', then by Lemma 9.72 the vertices of B_1 and B_2 within ε_j come in two consecutive blocks along ε_j. Since each barrier includes only vertices of even distance apart, B_1 contains at most half of the vertices in one block and B_2 contains at most half of the vertices in the other block. Hence, there are at most half of the internal vertices of ε_j in S. □

9.13 Bounding the maximum reachable excess

In order to prune search nodes, we wish to detect when it is impossible to extend the current 1-extendable graph H with q perfect matchings to a 1-extendable graph H' with p perfect matchings so that H' has an elementary supergraph $G' \in \mathcal{E}(H')$ with excess $c(G') \geq c$. The following lemma gives a method for bounding $c(G')$ using the maximum excess $c(G)$ over all elementary supergraphs G in $\mathcal{E}(H)$.

Lemma 9.74. *Let H be a 1-extendable graph on n vertices with $\Phi(H) = q$. Let H' be a 1-extendable supergraph of H built from H by a graded ear decomposition. Let $\Phi(H') = p > q$ and $N = n(H')$. Choose $G \in \mathcal{E}(H)$ and $G' \in \mathcal{E}(H')$ with the maximum number of edges in each set. Then,*

$$c(G') \leq c(G) + 2(p-q) - \frac{1}{4}(N-n)(n-2).$$

Proof. Let

$$H = H^{(0)} \subset H^{(1)} \subset \cdots \subset H^{(k-1)} \subset H^{(k)} = H'$$

be a non-refinable graded ear decomposition as in Theorem 9.55. For each $i \in \{0, 1, \ldots, k\}$, let $G^{(i)} \in \mathcal{E}(H^{(i)})$ be of maximum size. Without loss of generality, assume $G^{(0)} = G$ and $G^{(k)} = G'$. The following claims bound the excess $c(G^{(i)})$ in terms of $c(G^{(i-1)})$ using the ear augmentation $H^{(i-1)} \subset H^{(i)}$.

Claim 9.75. *If $H^{(i-1)} \subset H^{(i)}$ is a single ear augmentation $H^{(i)} = H^{(i-1)} + \varepsilon_1$ where ε_1 has order $\ell^{(i)}$, then*

$$c(G^{(i)}) \leq c(G^{(i-1)}) + 1 + \frac{3}{4}\ell^{(i)} - \frac{1}{8}(\ell^{(i)})^2 - \frac{1}{4}\ell^{(i)} n(H^{(i-1)}).$$

By Lemma 9.58, ε_1 spans two maximal barriers $X, Y \in \mathcal{P}(H^{(i)})$. $H^{(i)}$ has $\ell^{(i)} + 1$ more extendable edges than $H^{(i-1)}$.

We now bound the number of free edges $G^{(i)}$ has compared to the number of free edges in $G^{(i-1)}$. The elementary supergraph $G^{(i)}$ has a clique partition of free edges given by a maximal cover set $\mathcal{I}$ in $\mathcal{C}(H^{(i)})$. For each barrier $B \in \mathcal{I}$, the set $B \cap V(H^{(i-1)})$ is also a barrier of $H^{(i-1)}$, by Lemma 9.70. Through this transformation, the maximal cover set $\mathcal{I}$ admits an cover set $\mathcal{I}' = \{B \cap V(H^{(i-1)}) : B \in \mathcal{I}\}$ in $\mathcal{C}(H^{(i-1)})$. This cover set $\mathcal{I}'$ generates an elementary supergraph $G_*^{(i-1)} \in \mathcal{E}(H^{(i-1)})$ through the bijection in Claim 9.66. The free edges in $G^{(i)}$ which span endpoints within $V(H^{(i-1)})$ are exactly the free edges of $G_*^{(i-1)}$. By the selection of $G^{(i-1)}$, $e(G_*^{(i-1)}) \leq e(G^{(i-1)})$.

When $\ell^{(i)} > 0$, the $\ell^{(i)}$ internal vertices of ε_1 may be incident to free edges whose other endpoints lie in the barriers X and Y. By Corollary 9.73, at most half of the vertices in ε_1 have free edges to vertices in X and Y. Since the barriers X and Y are in $H^{(i-1)}$, they have size at most $\frac{n(H^{(i-1)})}{2}$. So, there are at most $\frac{\ell^{(i)}}{2}\frac{n(H^{(i-1)})}{2}$ free edges between these internal vertices and the rest of the graph. Also, there are at most $\binom{\ell^{(i)}/2}{2} = \frac{1}{8}(\ell^{(i)})^2 - \frac{1}{4}\ell^{(i)}$ free edges between the internal vertices themselves.

Combining these edge counts leads to the following inequalities:

$$\begin{aligned}
c(G^{(i)}) &= e(G^{(i)}) - \frac{(n(H^{(i-1)}) + \ell^{(i)})^2}{4} \\
&\leq \left[e(G_*^{(i-1)}) + \left(1 + \ell^{(i)}\right) + \frac{n(H^{(i-1)})\ell^{(i)}}{2} + \frac{1}{8}(\ell^{(i)})^2 - \frac{1}{4}\ell^{(i)} \right] \\
&\qquad - \left[\frac{n(H^{(i-1)})^2}{4} + \frac{n(H^{(i-1)})\ell^{(i)}}{2} + \frac{(\ell^{(i)})^2}{4} \right] \\
&\leq e(G^{(i-1)}) + 1 + \frac{3}{4}\ell^{(i)} - \frac{n(H^{(i-1)})^2}{4} - \frac{1}{8}(\ell^{(i)})^2 \\
&= c(G^{(i-1)}) + 1 + \frac{3}{4}\ell^{(i)} - \frac{1}{8}(\ell^{(i)})^2 - \frac{1}{4}\ell^{(i)} n_{i-1}.
\end{aligned}$$

This proves Claim 9.75. We now investigate a similar bound for two-ear autmentations.

Claim 9.76. *Let $H^{(i-1)} \subset H^{(i)}$ be a two-ear augmentation $H^{(i)} = H^{(i-1)} + \varepsilon_1 + \varepsilon_2$ where the ears ε_1 and ε_2 have $\ell_1^{(i)}$ and $\ell_2^{(i)}$ internal vertices, respectively. Set $\ell^{(i)} = \ell_1^{(i)} + \ell_2^{(i)}$. Then,*

$$c(G^{(i)}) \leq c(G^{(i)}) + 2 + \frac{3}{4}\ell^{(i)} - \frac{1}{8}(\ell^{(i)})^2 - \frac{1}{4}\ell_1^{(i)}\ell_2^{(i)} - \frac{1}{4}\ell^{(i)} n(H^{(i-1)}.$$

By Lemma 9.59, the first ear spans endpoints x_1, x_2 in a maximal barrier $X \in \mathcal{P}(H^{(i)})$ and the second ear spans endpoints y_1, y_2 in a different maximal barrier $Y \in \mathcal{P}(H^{(i)})$. Note that after these augmentations, x_1 and x_2 are not in the same barrier, and neither are y_1 and y_2, by Lemma 9.71.

The graph $G^{(i)}$ is an elementary supergraph of $H^{(i)}$ given by adding cliques of free edges corresponding to a maximal cover set $\mathcal{I}$ in $\mathcal{C}(H^{(i)})$. By Lemma 9.70, each barrier $B \in \mathcal{I}$ generates the barrier $B \cap V(H^{(i-1)})$ in $V(H^{(i-1)})$. This induces a cover set $\mathcal{I}' = \{B \cap V(H^{(i-1)}) : B \in \mathcal{I}\}$ in $\mathcal{C}(H^{(i-1)})$ which in turn defines an elementary supergraph $G_*^{(i-1)}$ through the bijection in Claim 9.66. By the choice of $G^{(i-1)}$, $e(G_*^{(i-1)}) \leq e(G^{(i-1)})$.

Consider the number of free edges in $G^{(i)}$ compared to the free edges in $G_*^{(i-1)}$. First, the number of edges between the $\ell_1^{(i)} + \ell_2^{(i)}$ new vertices and the $n(H^{(i-1)})$ original vertices is at most $\left(\frac{\ell_1^{(i)}}{2} + \frac{\ell_2^{(i)}}{2}\right)\frac{n(H^{(i-1)})}{2}$, since the additions must occur within barriers, at most half of the internal vertices of each ear can be used (by Corollary 9.73), and barriers in $H^{(i-1)}$ have at most $\frac{n(H^{(i-1)})}{2}$ vertices. Second, consider the number of free edges within the $\ell_1^{(i)} + \ell_2^{(i)}$ vertices. Note that no free edges can be added between ε_1 and ε_2 since the internal vertices of ε_1 and ε_2 are in different maximal barriers of $H^{(i)}$. Thus, there are at most $\binom{\ell_1^{(i)}/2}{2} + \binom{\ell_2^{(i)}/2}{2}$ free edges between the internal vertices. Since $\binom{\ell_1^{(i)}/2}{2} + \binom{\ell_2^{(i)}/2}{2} = \frac{1}{8}(\ell_1^{(i)} + \ell_2^{(i)})^2 - \frac{1}{4}(\ell_1^{(i)} + \ell_2^{(i)} + \ell_1^{(i)}\ell_2^{(i)})$, we have

$$
\begin{aligned}
c(G^{(i)}) &= e(G^{(i)}) - \frac{(n_{i-1}) + \ell_1^{(i)} + \ell_2^{(i)})^2}{4} \\
&\leq e(G_*^{(i-1)}) + \left(1 + \ell_1^{(i)} + 1 + \ell_2^{(i)}\right) + \frac{n(H^{(i-1)})(\ell_1^{(i)} + \ell_2^{(i)})}{4} \\
&\qquad + \frac{1}{8}\left(\ell_1^{(i)} + \ell_2^{(i)}\right)^2 - \frac{1}{4}\left(\ell_1^{(i)} + \ell_2^{(i)} + \ell_1^{(i)}\ell_2^{(i)}\right) \\
&\qquad - \left[\frac{n(H^{(i-1)})^2}{4} + \frac{n(H^{(i-1)})(\ell_1^{(i)} + \ell_2^{(i)})}{2} + \frac{(\ell_1^{(i)} + \ell_2^{(i)})^2}{4}\right] \\
&\leq e(G^{(i-1)}) - \frac{n(H^{(i-1)})^2}{4} + \left(2 + \ell_1^{(i)} + \ell_2^{(i)}\right) \\
&\qquad - \frac{1}{4}\left(\ell_1^{(i)} + \ell_2^{(i)}\right) - \frac{1}{8}\left(\ell_1^{(i)} + \ell_2^{(i)}\right)^2 - \frac{1}{4}\ell_1^{(i)}\ell_2^{(i)} - \frac{1}{4}n(H^{(i-1)})\ell^{(i)} \\
&= c(G^{(i-1)}) + 2 + \frac{3}{4}\left(\ell^{(i)}\right) - \frac{(\ell^{(i)})^2}{8} - \frac{1}{4}\ell_1^{(i)}\ell_2^{(i)} - \frac{1}{4}n(H^{(i-1)})\ell^{(i)}.
\end{aligned}
$$

We have now proven Claim 9.76. We now combine a sequence of these bounds to show the global bound.

Since each ear augmentation forces $\Phi(H^{(i)}) > \Phi(H^{(i-1)})$, there are at most

$p-q$ augmentations. Moreover, the increase in $c(G^{(i)})$ at each step is bounded by $1+\frac{3}{4}\ell^{(i)}-\frac{1}{8}(\ell^{(i)})^2-\frac{1}{4}\ell^{(i)}n(H^{(i-1)})$ in a single ear augmentation and $2+\frac{3}{4}\ell^{(i)}-\frac{1}{8}(\ell^{(i)})^2-\frac{1}{4}\ell_1^{(i)}\ell_2^{(i)}-\frac{1}{4}\ell^{(i)}n(H^{(i-1)})$ in a double ear augmentation. Independent of the number of ears,

$$c(G^{(i)})-c(G^{(i-1)})\leq 2+\ell^{(i)}-\frac{1}{8}(\ell^{(i)})^2-\frac{1}{4}\ell^{(i)}n(H^{(i-1)}).$$

Note also that if $\ell^{(i)}$ is positive, then it is at least two. Combining those inequalities gives

$$\begin{aligned}
c(G') &\leq c(G)+\sum_{i=1}^{k}\left(2+\frac{3}{4}\ell^{(i)}-\frac{1}{8}(\ell^{(i)})^2-\frac{1}{4}\ell^{(i)}n(H^{(i-1)})\right)\\
&\leq c(G)+\sum_{i=1}^{k}2+\frac{3}{4}\sum_{i=1}^{k}\ell^{(i)}-\frac{1}{8}\sum_{i=1}^{k}(\ell^{(i)})^2-\frac{1}{4}\sum_{i=1}^{k}\ell^{(i)}n(H^{(i-1)})\\
&\leq c(G)+2k+\frac{3}{4}(N-n)-\frac{1}{8}\sum_{i=1}^{k}2\ell^{(i)}-\frac{1}{4}\sum_{i=1}^{k}\ell^{(i)}n\\
&\leq c(G)+2(p-q)-\frac{1}{4}(N-n)(n-2).
\end{aligned}$$

This proves the result. □

Corollary 9.77. *Let $p, c \geq 1$ be integers. If H is a 1-extendable graph with $q=\Phi(H)$, c' is the maximum excess $c(G)$ over all graphs $G\in\mathcal{E}(H)$, and $c'+2(p-q)<c$, then there is no 1-extendable graph $H\subset H'$ reachable from H by a graded ear decomposition so that $\Phi(H')=p$ and there exits a graph $G'\in\mathcal{E}(H')$ with excess $c(G')\geq c$.*

Corollary 9.77 gives the condition to test if we can prune the current node, since there does not exist a sequence of ear augmentations that lead to a graph with excess at least our known lower bound on c_p. Moreover, Lemma 9.74 provides a dynamic bound on the number N of vertices that can be added to the current graph while maintaining the possibility of finding a graph with excess at least the known lower bound on c_p, by selecting N to be maximum so that $c' + 2(p - \Phi(H)) - \frac{1}{4}(N-n)(n-2) \geq c$.

9.14 Results and Data

The full algorithm to search for p-extremal elementary graphs combines three types of algorithms. First, the canonical deletion from Section 9.10 is used to enumerate the search space with no duplication of isomorphism classes. Second, the pruning procedure from Section 9.13 greatly reduces the number of generated graphs by backtracking when no dense graphs are reachable. Third, Section 9.11 provided a method for adding free edges to a 1-extendable graph with p perfect matchings to find maximal elementary supergraphs.

The recursive generation algorithm Search$(H^{(i)}, N, p, c)$ is given in Algorithm 9.1. Given a previously computed lower bound $c \leq c_p$, the full search Generate(p, c) (Algorithm 9.2) operates by running Search(C_{2k}, N_p, p, c) for each even cycle C_{2k} with $4 \leq 2k \leq N_p$. All elementary graphs G with $\Phi(G) = p$ and $c(G) \geq c$ are generated by this process.

Theorem 9.78. *Given p and $c \leq c_p$, Generate(p, c) (Algorithm 9.2) outputs all unlabeled elementary graphs with p perfect matchings and excess at least c.*

Proof. Given an unlabeled graph G with $\Phi(G) = p$ and $c(G) \geq c$, note that Corollary 9.26 implies $n(G) \leq 3 + \sqrt{16p - 8c - 23}$. With respect to the canonical deletion del(H), let $H^{(0)} \subset H^{(1)} \subset \cdots \subset H^{(k)}$ be the canonical ear decomposition of the extendable subgraph H in G. By the choice of canonical deletion, this decomposition takes the form of Corollary 9.56. Moreover, $H^{(0)}$ is an even cycle C_{2r} for

Algorithm 9.1 Search$(H^{(i)}, N^{(i)}, p, c)$

```
Check all pairs of vertices, up to symmetries
for all vertex-pair orbits $\mathcal{O}$ in $H^{(i)}$ do
    $\{x, y\} \leftarrow$ representative pair of $\mathcal{O}$
    Augment by ears of all even orders
    for all orders $r \in \{0, 2, \ldots, N^{(i)} - n(H^{(i)})\}$ do
        $H^{(i+1)} \leftarrow H^{(i)} + \mathrm{Ear}(x, y, r)$
        if $H^{(i)}$ is almost 1-extendable and $H^{(i+1)}$ is not 1-extendable then
            Skip $H^{(i+1)}$ (decomposition is not graded).
        else if $\Phi(H^{(i+1)}) > p$ then
            Skip $H^{(i+1)}$.
        else
            Check the canonical deletion
            $(x', y', r') \leftarrow \mathrm{del}(H^{(i+1)})$
            if $r = r'$ and $\{x', y'\} \in \mathcal{O}$ then
                This augmentation matches the canonical deletion
                $n^{(i+1)} \leftarrow n(H^{(i+1)})$.
                $p^{(i+1)} \leftarrow \Phi(H^{(i+1)})$.
                $c^{(i+1)} \leftarrow \max\{c\left(H^{(i+1)}_{\mathcal{I}}\right) : \mathcal{I} \in \mathcal{C}(H^{(i+1)})\}$.
                if $p^{(i+1)} = p$ and $c^{(i+1)} \geq c$ then
                    There are solutions within $\mathcal{E}(H^{(i+1)})$.
                    for all cover sets $\mathcal{I} \in \mathcal{C}(H^{(i+1)})$ do
                        if $c\left(H^{(i+1)}_{\mathcal{I}}\right) \geq c$ then
                            Output $H_{\mathcal{I}}$.
                        end if
                    end for
                else if $q < p$ and $c^{(i+1)} + 2(p - p^{(i+1)}) \geq c$ then
                    Use Lemma 9.74 to bound the size of future augmentations.
                    $N^{(i+1)} = \max\{N' : c^{(i+1)} + 2(p - p^{(i+1)})$
                                    $- \frac{1}{4}(N' - n^{(i+1)})(n^{(i+1)} - 2) \geq c\}$.
                    Search$(H^{(i+1)}, N^{(i+1)}, p, c)$.
                end if
            end if
        end if
    end for
end for
return
```

Algorithm 9.2 Generate(p, c)

```
$N \leftarrow \max\{2r : 2r \leq 3 + \sqrt{16p - 8c - 23}\}$.
for $r \in \{1, \ldots, N/2\}$ do
    Search$(C_{2r}, N, p, c)$
end for
return
```

some r. The Generate(p, c) method initializes Search(C_{2r}, N, p, c).

By the definition of canonical ear decomposition, the canonical ear $\varepsilon^{(i)}$ of $H^{(i)}$ is the ear used to augment from $H^{(i-1)}$ to $H^{(i)}$. Let $x^{(i)}, y^{(i)}$ be the endpoints of $\varepsilon^{(i)}$. When Search($H^{(i)}, N^{(i)}, p, c$) is called, the pair orbit $\mathcal{O}$ containing $\{x^{(i+1)}, y^{(i+1)}\}$ is visited and an ear ε of the same order as $\varepsilon^{(i+1)}$ is augmented to $H^{(i)}$ to form a graph $H_*^{(i+1)}$. Note that $H_*^{(i+1)} \cong H^{(i+1)}$ with an isomorphism mapping ε to $\varepsilon^{(i+1)}$. By the definition of the canonical deletion del(H), the algorithm accepts this augmentation.

For each i, let $G^{(i)}$ be a maximum-size elementary supergraph in $\mathcal{E}(H^{(i)})$. By Theorem 9.65, there exists a maximal cover set $\mathcal{I} \in \mathcal{C}(H^{(i)})$ so that $G^{(i)} = H_{\mathcal{I}}^{(i)}$. Since $c(G^{(k)}) = c(G) \geq c$, Lemma 9.74 gives $c(G) \leq c(G^{(i+1)}) + 2(p - p^{(i+1)}) - \frac{1}{4}(n(G) - n(H^{(i+1)}))(n(H^{(i+1)}) - 2)$, so the algorithm recurses with $N^{(i+1)} \geq n(G)$.

When $H^{(k)}$ is reached, the algorithm notices that $\Phi(H^{(k)}) = p$ and enumerates all cover sets $\mathcal{I} \in \mathcal{C}(H^{(k)})$ which generates the elementary supergraphs $H_{\mathcal{I}}^{(k)} \in \mathcal{E}(H^{(k)})$ with excess at least c. Since $H^{(k)}$ is the extendable subgraph of G and $c(G) \geq c$, this procedure will output G. □

The framework for this search was implemented within the *EarSearch* library. This software is detailed in Appendix E. This implementation was executed on the Open Science Grid [107] using the University of Nebraska Campus Grid [143]. The nodes available on the University of Nebraska Campus Grid consist of Xeon and Opteron processors with a speed range of between 2.0 and 2.8 GHz.

p	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
n_p	8	6	8	8	6	8	8	8	8	8	8	8	8	8	8	8	8
c_p	3	5	3	4	6	4	4	5	4	5	5	5	5	6	5	5	6
C_p	4	5	5	5	6	6	6	6	6	6	6	6	6	6	6	6	6

Table 9.2: New values of n_p and c_p. Conjecture 9.41 states that $c_p \leq C_p$.

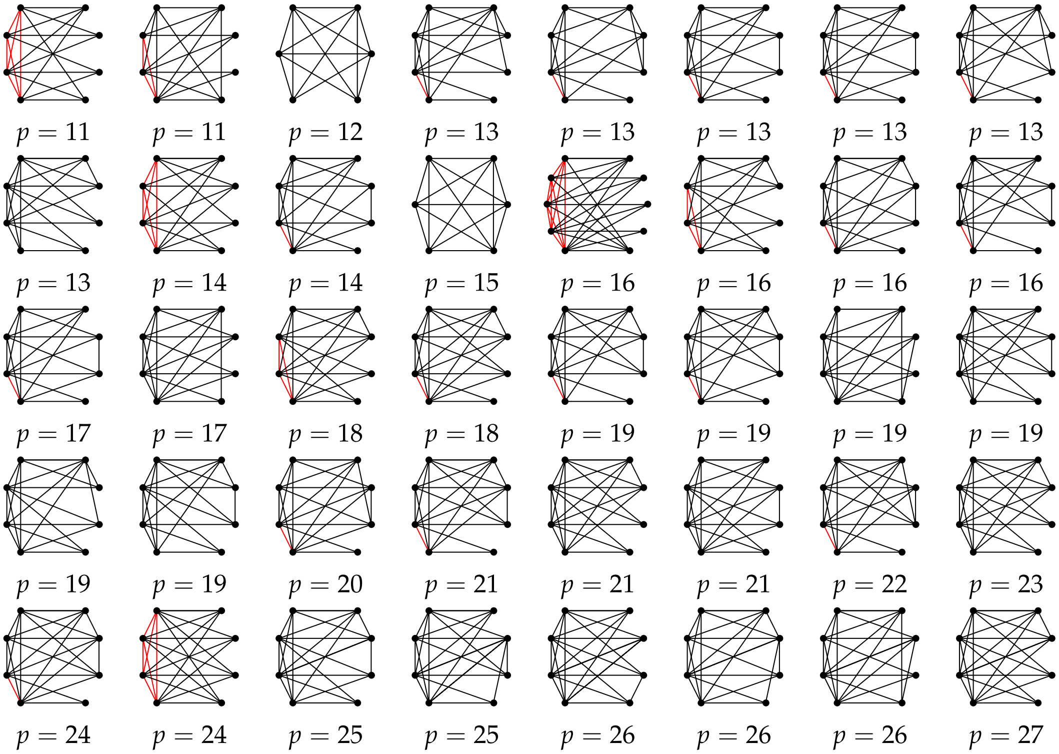
p	N_p	c_p	CPU Time
5	8	2	0.02s
6	10	3	0.04s
7	10	3	0.18s
8	12	3	0.72s
9	12	4	1.46s
10	12	4	5.95s
11	14	3	43.29s
12	14	5	44.01s
13	14	3	6.66m
14	16	4	12.17m
15	16	6	12.71m
			.

p	N_p	c_p	CPU Time
16	16	4	2.02h
17	16	4	6.77h
18	18	5	11.75h
19	18	4	2.71d
20	18	5	4.21d
21	18	5	13.71d
22	20	5	42.84d
23	20	5	118.32d
24	20	6	209.42d
25	20	5	2.52y
26	20	5	7.21y
27	22	6	10.68y

Table 9.3: Time analysis of the search for varying p values.

Combining this algorithm with known lower bounds on c_p for $p \in \{11, \ldots, 27\}$ provided a full enumeration of p-extremal graphs for this range of p. The resulting values of c_p and n_p are given in Table 9.2. The computation time for these values ranged from less than a minute to more than 10 years. Table 9.3 gives the full list of computation times. The resulting p-extremal elementary graphs for $11 \leq p \leq 27$ are given in Figure 9.7.

To describe the complete structural characterization of p-extremal graphs on n vertices for all even $n \geq n_p$, we apply Theorem 9.27. An important step in applying Theorem 9.27 is to consider every factorization $p = \prod p_i$ and to check which spires are generated by the p_i-extremal elementary graphs. We describe these structures based on the types of constructions given by these factorizations. It is necessary to consider the p-extremal elementary graphs for $1 \leq p \leq 10$, in Figure 9.8.



$p = 11$ $p = 11$ $p = 12$ $p = 13$ $p = 13$ $p = 13$ $p = 13$ $p = 13$

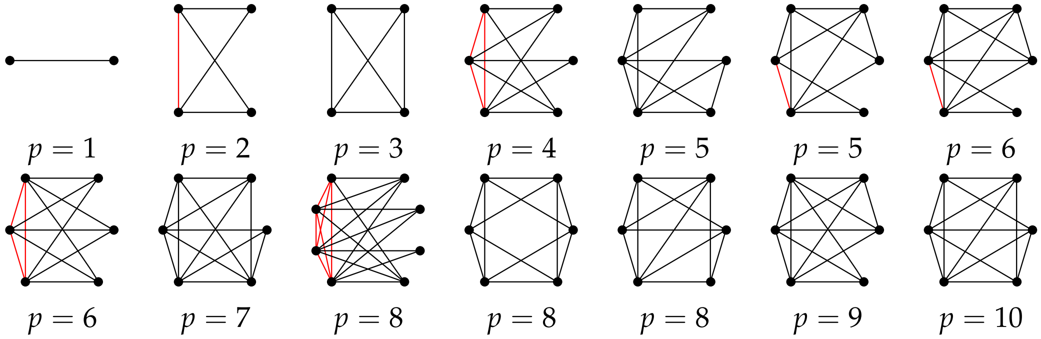
$p = 13$ $p = 14$ $p = 14$ $p = 15$ $p = 16$ $p = 16$ $p = 16$ $p = 16$

$p = 17$ $p = 17$ $p = 18$ $p = 18$ $p = 19$ $p = 19$ $p = 19$ $p = 19$

$p = 19$ $p = 19$ $p = 20$ $p = 21$ $p = 21$ $p = 21$ $p = 22$ $p = 23$

$p = 24$ $p = 24$ $p = 25$ $p = 25$ $p = 26$ $p = 26$ $p = 26$ $p = 27$

Figure 9.7: The p-extremal elementary graphs where $1 \leq p \leq 27$.



$p = 1$ $p = 2$ $p = 3$ $p = 4$ $p = 5$ $p = 5$ $p = 6$

$p = 6$ $p = 7$ $p = 8$ $p = 8$ $p = 8$ $p = 9$ $p = 10$

Figure 9.8: The p-extremal elementary graphs with $1 \leq p \leq 10$ [38, 63].

For $p \in \{11, 13, 17, 19, 23\}$, p is prime, and there is no non-trivial factorization of p. Hence, a p-extremal graph is a spire using exactly one p-extremal elementary graph with all other vertices within chambers isomorphic to K_2. In most cases, the p-extremal elementary graph must appear at the top of the spire. Only when $p = 11$ and the 11-extremal elementary graph chosen is the one with a barrier of size 4 can this chamber be positioned anywhere in the spire.

For each $p \in \{15, 22, 25, 26\}$, p has at least one non-trivial factorization $p = \prod p_i$, but the sum of c_{p_i} over the factors is strictly below c_p. Hence, no p-extremal spire could contain more than one non-trivial chamber. Also, all p-extremal elementary graphs have a barrier with relative size strictly below $\frac{1}{2}$, so the non-trivial chamber must appear at the top of the spire.

For each $p \in \{21, 27\}$, there exists at least one factorization $p = \prod p_i$, all with $\sum c_{p_i} \leq c_p$, and at least one factorization which reaches c_p with equality. For example, $21 = 3 \cdot 7$, and $c_3 + c_7 = 2 + 3 = 5 = c_{21}$. However, in these cases of equality, p_i-extremal elementary graphs with large barriers do not exist and it is impossible to achieve an excess of c_p over the entire spire using multiple non-trivial chambers. Hence, the p-extremal graphs for these values of p have exactly one non-trivial chamber with p perfect matchings and these chambers have small barriers, so they must appear at the top of the spire.

For each $p \in \{14, 18, 20, 24\}$, there is at least one factorization $p = \prod p_i$ so that $\sum c_{p_i} = c_p$ and there are p_i-extremal graphs with large enough barriers to admit a spire with excess c_p. These factorizations are $14 = 2 \cdot 7$, $18 = 3 \cdot 6 = 2 \cdot 9$, $20 = 2 \cdot 10$, and $24 = 2 \cdot 12$. There are also the p-extremal spires with exactly one non-trivial chamber, most of which must appear at the top of the spire. For $p \in \{14, 24\}$, there exists one p-extremal elementary graph with a large barrier that can appear anywhere in a p-extremal spire.

The case $p = 16$ is special: every factorization admits at least one configuration for a 16-extremal spire. The q-extremal elementary graphs for $q \in \{1, 2, 4, 8\}$ as found by Hartke, Stolee, West, and Yancey [63] are given in Figure 9.8. Note that for each such q, there exists at least one q-extremal graph with a barrier with relative size $\frac{1}{2}$. This allows any combination of values of q that have product 16 give a spire with 16 perfect matchings and excess equal to the sum of the excesses of the chambers, which always adds to $c_{16} = 4$. There are two 8-extremal elementary graphs and three 16-extremal elementary graphs which have small barriers and must appear at the top of a 16-extremal spire. All other chambers of a 16-extremal spire can take any order.

9.15 Discussion

The $O(\sqrt{p})$ bound N_p on the number of vertices in a p-extremal elementary graph was sufficient for the computational technique described in this work to significantly extend the known values of c_p. However, all of the elementary graphs we discovered to be p-extremal for $p \leq 27$ have at most 10 vertices, which could be generated using existing software, such as McKay's `geng` program [93]. With a smaller N_p value, the distributed search can also be improved. Computation time would still be exponential in p because the depth of the search is a function of p, but the branching factor at each level would be reduced. This delays the exponential behavior and potentially makes searches over larger values of p become tractable. This motivates the following conjecture.

Conjecture 9.79. *Every p-extremal elementary graph has at most* $2\log_2 p$ *vertices.*

This conjecture is tight for $p = 2^k$, with $k \in \{1, 2, 3, 4\}$ and holds for all $p \leq 27$. Note that $n_8 = 6$, but there is an 8-extremal elementary graph with eight vertices. Similarly, $n_{16} = 8$, but there is a 16-extremal elementary graph with ten vertices.

The structure theorem requires searching over all factorizations of p in order to determine which factorizations yield a spire with the largest excess. However, all known values of p admit p-extremal elementary graphs. Moreover, all composite values $p = p_1 p_2$ admit $c_p \geq c_{p_1} + c_{p_2}$. Does this always hold?

Conjecture 9.80. *For all $p \geq 1$, there exists a p-extremal elementary graph.*

Conjecture 9.81. *For all products $p = \prod_{i=1}^{k} p_i$ with $p \geq 1$, $c_p \geq \sum_{i=1}^{k} c_{p_i}$.*

The closest known bound to Conjecture 9.81 is $c_p \geq c_{p_1} + \sum_{i=2}^{k} w(p_i)$, where $w(n)$ is the number of 1's in the binary representation of n [63, Proposition 7.1].

The method we used to determine the graphs in $\mathcal{F}_p$ for $p \leq 10$ is feasible only for small p. Several natural questions arise from these computational results. The data suggest that there is always at least one p-extremal graph whose subgraph of extendable edges is connected. If this is true, then it could help to guide searches, since when $n \geq n_p$ some p-extremal graph would have a chamber other than K_2 only at the top (this must happen when p is prime).

Conjecture 9.82. *For $p \in \mathbb{N}$, there exist a p-extremal graph that is an elementary graph.*

For p-extremal spires that consist of copies of K_2 and one p-extremal chamber, the value of c_p may be small. The examples we have of $c_p < c_{p-1}$ occur when p is a power of a prime, at $p \in \{11, 13, 16, 19, 25\}$. With greater variety of factorizations available, there are more ways to form spires with exactly p perfect matchings and hence more ways for c_p to be large. The data suggest the following.

Conjecture 9.83. *For $p \in \mathbb{N}$, always $c_p \geq \max\{\sum c_i\{p_i\} \colon \prod p_i = p\}$.*

A different conjecture is suggested by consider the values of c_q after c_p. Let $m_p = \min\{c_q \colon q \geq p\}$; how does this sequence behave? We know only that $m_p \geq 1$, by Theorem 9.7. However, if there are finitely many Fermat primes (primes of the form $2^k + 1$, see [57]), then $m_p \geq 2$ for sufficiently large p, by Corollary 9.33. This is too difficult a task for such a small payoff, especially since a much stronger statement seems likely.

Conjecture 9.84. $\lim_{p\to\infty} m_p = \infty$.

Figure 9.6 provides strong support for this conjecture. If it holds, then what is the asymptotic behavior of m_p? Is it $\Omega\left(\left(\frac{\ln p}{\ln\ln p}\right)^2\right)$, matching the conjectured upper bound? Or, is it smaller, such as $\Omega(\log_2 p)$, the current lower bound when p is a power of 2?

Finally, it would be interesting to characterize $\mathcal{F}_p$ or at least compute c_p for some infinite family of values of p.

Bibliographic Notes

The work in this chapter is joint work with Stephen G. Hartke, Douglas B. West, and Matthew Yancey [63]. This project was initiated during the Combinatorics Research Experience for Graduate Students (REGS) at the University of Illinois, Summer 2010. We thank Garth Isaak for suggesting the extension to odd n.

Part III

Orbital Branching

Chapter 10

Orbital Branching

A property P on a combinatorial object can be expressed as a set $\{C_i\}$ of constraints on a set of $0/1$-valued variables. For example, graphs can be stored as indicator variables $x_{u,v}$ with value 1 if and only if uv is an edge. Then, a constraint $C_i(\mathbf{x})$ can be satisfied when the equality $\sum_{j=1}^{n} x_{i,j} = 4$ holds. If all constraints $\{C_i\}_{i=1}^{n}$, then the graph corresponding to these variables is 4-regular. Thus, a property P can be expressed as the conjunction of several constraints, where P holds if and only if all constraints hold.

Orbital branching is a technique to solve symmetric constraint systems using branch-and-bound techniques by selecting orbits of variables for the branch instead of a single variable. Orstrowski, Linderoth, Rossi, and Smriglio [102] introduced orbital branching and applied the technique to covering and packing problems. The authors implemented the branching procedure into the optimization framework MINTO [98]. This integration allowed interaction with existing heuristics such as clique cuts [8] and the commercial optimization software IBM ILOG CPLEX [35]. They also generalized the technique to include knowledge of subproblems [103] within the context of Steiner systems [104].

There are three main components to orbital branching. First, a description of the constraint system must be implemented. Tightly integrated subcomponents of the system include the symmetry model of the constraints and the presolver. The symmetry model consists of a graph G with vertices corresponding to the variables along with vertices encoding the structure of the system. The automorphism group of G is computed using `nauty` [93, 61]. The presolver reduces the number of variables and constraints by recognizing dependencies. Second, the constraint propagation algorithm takes existing choices from the branch-and-bound and fixes variables with forced values. Third, a *branching rule* decides which orbit of variables is selected for the branch.

10.1 Variable Assignments

We shall consider a constraint problem P where $P : \{0,1\}^m \to \{0,1\}$ is a function on m variables which take value in $\{0,1\}$. A vector $\mathbf{x} = (x_1, \dots, x_m) \in \{0,1,*\}^m$ is a *variable assignment*. If $x_i = *$, we say that variable is *unassigned*. We can compare two assignments $\mathbf{x}, \mathbf{y}$ with $\mathbf{x} \preceq \mathbf{y}$ if and only if for all $i \in \{1, \dots, m\}$ either $x_i = *$ or $x_i = y_i$.

Example. An (n, k, λ, μ) *strongly regular graph* is a k-regular graph on n vertices so that if uv is an edge, then u and v have λ common neighbors and if uv is not an edge, then u and v have μ common neighbors. Let $\mathbf{x}$ be a variable assignment on $m = \binom{n}{2}$ variables corresponding to $x_i = 1$ if and only if the ith unordered pair of vertices[1] is an edge.

---

[1] We assume some order on pairs of elements is fixed. The co-lexicographic order is particularly useful because it does not require knowledge of the number of elements in order to rank or unrank a pair. That is, consider the vertex set $\{v_0, v_1, v_2, \dots\}$. For vertices v_i, v_j with $i < j$, the rank of the pair $\{v_i, v_j\}$ can be given as $\binom{j-1}{2} + i$. Thus, all pairs from this infinite vertex set are mapped

Thus, a variable assignment $\mathbf{x}$ maps to a *trigraph* G which is given by a vertex set and the pairs are partitioned into three sets: *edges*, *nonedges*, and *unassigned pairs* (corresponding to $x_i = 1$, $x_i = 0$, and $x_i = *$ for the ith pair). Two trigraphs G and H can be compared as $G \preceq H$ if the corresponding variable assignments $\mathbf{x}$ and $\mathbf{y}$ have $\mathbf{x} \preceq \mathbf{y}$. Trigraphs and variable assignments are useful data structures when searching for strongly regular graphs because nonedges are fundamental to the structure of the graph.

Let $\mathsf{SRG}_{n,k}^{\lambda,\mu}(\mathbf{x})$ be the boolean function that encodes whether or not the given variable assignment corresponds to an (n, k, λ, μ) strongly regular graph. For later examples, we shall consider n, k, λ, and μ to be fixed and define $\mathsf{SRG}(\mathbf{x}) = \mathsf{SRG}_{n,k}^{\lambda,\mu}(\mathbf{x})$.

Our goal is to generate all possible vectors $\mathbf{x}$ so that $P(\mathbf{x}) = 1$. We shall build these vectors starting at the empty assignment $\mathbf{x} = (*, *, \dots, *)$ and assign values to the variables. A naïve approach would be to brute-force check every possible assignment. Instead, we shall employ constraint detection and constraint propagation techniques.

1. Let $\text{Detect}_P(\mathbf{x})$ be an algorithm that returns True only if for all $\mathbf{y} \in \{0,1\}^n$ where $\mathbf{x} \preceq \mathbf{y}$ we have $P(\mathbf{y}) = 0$.
2. Let $\text{Propagate}_P(\mathbf{x})$ be an algorithm that on input $\mathbf{x} \in \{0,1,*\}^n$ returns a variable assignment $\mathbf{y} \in \{0,1,*\}^n$ so that for all $\mathbf{z}$ where $\mathbf{x} \preceq \mathbf{z}$ so that $P(\mathbf{z}) = 1$, then $\mathbf{x} \preceq \mathbf{y} \preceq \mathbf{z}$.

The efficiency and strength of Detect_P and Propagate_P will depend on the property P (and the user's knowledge about the property P). At minimum, $\text{Detect}_P(\mathbf{x})$

---

bijectively to natural numbers so that for a fixed n the pairs using vertices $v_0, \dots, v_{n-1}$ map to the set $\{0, \dots, \binom{n}{2} - 1\}$. The unranking formula is efficient to compute.

could return True if and only if $\mathbf{x}$ has no unassigned variables and $P(\mathbf{x}) = 0$. Further, $\text{Propagate}_P(\mathbf{x})$ could simply return $\mathbf{x}$. Using some structure of the function P can allow more complicated (and helpful) functions.

Example. When searching for an (n, k, λ, μ) strongly regular graph, we can guarantee a few properties. Let $\mathbf{x}$ be a variable assignment and $G_{\mathbf{x}}$ be the associated trigraph. Note that the maximum degree of a graph ($\Delta(G)$) and neighborhoods ($N(v_i)$) are monotone functions on the edge set of trigraphs with respect to $\preceq$. The following implications are then immediate:

1. If $\mathsf{SRG}(G) = 1$, then $\Delta(G) = k$. Therefore, if $\Delta(G_{\mathbf{x}}) > k$, then $G_{\mathbf{x}}$ cannot extend to a strongly regular graph.
2. If $\mathsf{SRG}(G) = 1$, then $\Delta(\overline{G}) = n - k - 1$. Therefore, if $\Delta(\overline{G_{\mathbf{x}}}) > n - k - 1$, then $G_{\mathbf{x}}$ cannot extend to a strongly regular graph.
3. If $\mathsf{SRG}(G) = 1$, then every pair $v_iv_j \in E(G)$ has $|N(v_i) \cap N(v_j)| = \lambda$. Therefore, if v_iv_j is an edge in $G_{\mathbf{x}}$ and $|N(v_i) \cap N(v_j)| > \lambda$ then $G_{\mathbf{x}}$ cannot extend to a strongly regular graph.
4. If $\mathsf{SRG}(G) = 1$, then every pair $v_iv_j \in E(\overline{G})$ has $|N(v_i) \cap N(v_j)| = \mu$. Therefore, if v_iv_j is a nonedge in $G_{\mathbf{x}}$ and $|N(v_i) \cap N(v_j)| > \mu$ then $G_{\mathbf{x}}$ cannot extend to a strongly regular graph.

Thus, a possible algorithm for $\text{Detect}_{\mathsf{SRG}}(\mathbf{x})$ is to return 1 whenever $\Delta(G_{\mathbf{x}}) > k$, $\Delta(\overline{G_{\mathbf{x}}}) > n - k - 1$, or $|N(v_i) \cap N(v_j)| > \max\{\lambda, \mu\}$ for some pair $v_i, v_j \in V(G_{\mathbf{x}})$. Further, a possible algorithm for $\text{Propagate}_{\mathsf{SRG}}(\mathbf{x})$ is to place assignments on unassigned variables whenever these constraints are sharp:

1. If v_i has degree k, then for any vertex v_j where the pair v_iv_j is unassigned, make v_iv_j be a nonedge.

2. If v_i has non-degree[2] $n-k-1$, then for any vertex v_j where the pair v_iv_j is unassigned, make v_iv_j be an edge.

3. If v_iv_j is an edge in $G_{\mathbf{x}}$ and $|N(v_i) \cap N(v_j)| = \lambda$ then whenever v_ℓ is a vertex with v_iv_ℓ an edge and $v_\ell v_j$ unassigned, make $v_\ell v_j$ be a nonedge. Similarly, if v_iv_ℓ is unassigned and $v_\ell v_j$ is an edge, make v_iv_ℓ be a nonedge.

4. If v_iv_j is a nonedge in $G_{\mathbf{x}}$ and $|N(v_i) \cap N(v_j)| = \mu$ then whenever v_ℓ is a vertex with v_iv_ℓ an edge and $v_\ell v_j$ unassigned, make $v_\ell v_j$ be a nonedge. Similarly, if v_iv_ℓ is unassigned and $v_\ell v_j$ is an edge, make v_iv_ℓ be a nonedge.

Given algorithms for $\text{Detect}_P(\mathbf{x})$ and $\text{Propagate}_P(\mathbf{x})$, we can build a full search algorithm using *branch-and-bound*[3]. One missing ingredient is the $\text{Unassigned}_P(\mathbf{x})$ function which selects an index i of an unassigned variable ($x_i = *$). Such a selection could be arbitrary, but it could also be carefully selected. This algorithm allows for *dynamic variable ordering*, which changes which variable we shall assign next in hopes that certain constraints will become sharp and the $\text{Propagate}_P(\mathbf{x})$ algorithm can assign several new values in one step. The full branch-and-bound algorithm is given as $\text{BranchAndBound}_P(\mathbf{x})$ in Algorithm 10.1.

By initializing this recursive algorithm on the empty assignment $\mathbf{x} = (*, \dots, *)$, all possible feasible solutions $\mathbf{x}$ with $P(\mathbf{x}) = 1$ will be written to output.

While branch-and-bound is a complete algorithm, it has no concern for the symmetries of the objects represented by the variable assignments. That is, perhaps the property P is invariant under a certain set of permutations and so we should only generate variable assignments up to isomorphism. The next section defines these symmetries.

---

[2] The *non-degree* of a vertex v_i is the number of nonedge pairs containing v_i.

[3] Also called *backtrack search*, we shall not use the optimization methods used in a typical branch-and-bound algorithm. So, you could also call this method simply "branch."

Algorithm 10.1 BranchAndBound$_P(\mathbf{x})$

```
if Detect_P(x) then
  return
end if
y ← Propagate_P(x)
if y ∈ {0,1}^m then
  if P(y) = 1 then
    Output y
  end if
  return
end if
i ← Unassigned_P(x)
y_i ← 0
call BranchAndBound_P(y)
y_i ← 1
call BranchAndBound_P(y)
return
```

10.2 Constraint Symmetries

Let $\sigma \in S_n$ be a permutation of order m. Given a vector $\mathbf{x} = (x_1, \dots, x_m)$, the vector $\mathbf{x}_\sigma$ has value $\mathbf{x}_\sigma = (x_{\sigma(1)}, x_{\sigma(2)}, \dots, x_{\sigma(m)})$. That is, σ acts on the variables.

An *automorphism* of a constraint problem P is a permutation $\sigma \in S_m$ so that for every vector $\mathbf{x} \in \{0,1\}^m$, $P(\mathbf{x}) = P(\mathbf{x}_\sigma)$. The set of automorphisms of P form a group, denoted $\mathrm{Aut}(P)$.

Example. When the variable assignment $\mathbf{x}$ corresponds to a trigraph $G_\mathbf{x}$ (as in the case of strongly regular graphs), we consider a permutation $\tau \in S_n$ to be a *colored isomorphism* between two trigraphs G and H if for all pairs v_iv_j, the color of $v_{\tau(i)}v_{\tau(j)}$ in H matches the color of v_iv_j in G. A property P is *invariant* under colored isomorphsims if $P(G) = P(H)$ for any two isomorphic trigraphs. Therefore, for a property P that is invariant under colored isomorphisms,

$$\mathrm{Aut}(P) \supseteq \{\sigma_\tau \in S_{\binom{n}{2}} : \forall \tau \in S_n\},$$

where σ_τ maps a pair v_iv_j to $v_{\tau(i),\tau(j)}$. That is, any permutation of the vertices corresponds to a map of the pairs and P is invariant under these permutations.

Since we shall be making changes to the variable assignment $\mathbf{x}$, we want to track the symmetries of the system with respect to the current variable assignment. Thus, let $\mathrm{Aut}_{\mathbf{x}}(P)$ be the set of permutations of the variable set so that P is invariant under all extensions of the given variable assignment, and a variable x_i is mapped to a variable with the same value:

$$\mathrm{Aut}_{\mathbf{x}}(P) = \{\sigma \in \mathrm{Aut}(P) : \forall i \in \{1, \ldots, m\}, x_i = x_{\sigma(i)}\}.$$

Example. For a trigraph G, all extensions of this trigraph will convert some unassigned pairs into edges and nondedge, but all current edges and nonedges will be present in any trigraph H with $G \preceq H$. Therefore, we shall restrict the symmetries of the system to be colored automorphisms of G. Thus, the colored automorphism group $\mathrm{Aut}_{\mathbf{x}}(P)$ is

$$\mathrm{Aut}_{\mathbf{x}}(P) = \{\sigma_\tau : \tau \in S_n, \tau \text{ is a colored automorphism of } G_{\mathbf{x}}\}.$$

This automorphism group can be computed by defining a *layered* graph $L(G)$ which has vertex set $V(L(G)) = V(G) \times \{0,1\}$ and edges given by

1. $(u,0) \leftrightarrow (u,1)$ for all $u \in V(G)$,
2. $(u,0) \leftrightarrow (v,0)$ for all nonedges uv of G,
3. $(u,1) \leftrightarrow (v,1)$ for all edges uv of G.

By partitioning $V(L(G))$ by the second coordinate (one part is $V(G) \times \{0\}$ and the other is $V(G) \times \{1\}$) computing the automorphisms of $L(G)$ that stabilize these parts guarantees that for all $u \in V(G)$ and automorphisms σ, $\sigma((u,0)) = (v,0)$ for some $v \in V(G)$ and $\sigma((u,1)) = (v,1)$. Therefore, the automorphisms of $L(G)$ (as permutations of $V(G) \times \{0,1\}$) collapse to colored automorphisms of G (as permutations of $V(G)$). This is a standard method for computing colored automorphisms, as described in the *nauty* user guide [93].

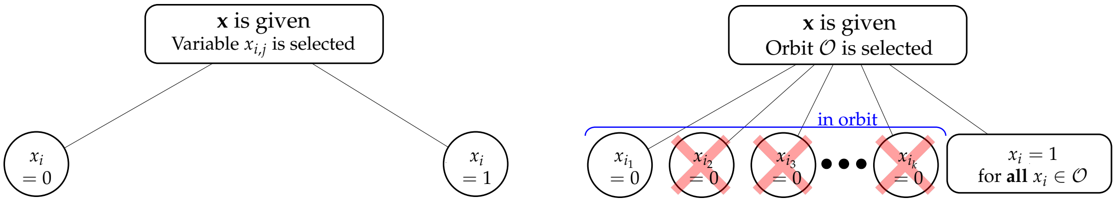
We shall focus on the action this group on the variables. The *orbit* of a variable x_i is the set $\mathcal{O}_i = \{x_j : \exists\sigma \in \mathrm{Aut}_{\mathbf{x}}(P), j = \sigma(i)\}$. Using these symmetries and considering orbits of unassigned variables instead of single variables, we can extend branch-and-bound to be symmetry-aware and reduce the number of isomorphic duplicates.

10.3 Orbital Branching

Given a partial variable assignment $\mathbf{x}$, we compute the colored automorphism group $\mathrm{Aut}_{\mathbf{x}}(P)$ and use this action on the variables to compute orbits. Orbital branching extends the standard branch-and-bound technique by selecting an orbit of unassigned variables. From a selected orbit $\mathcal{O}$, there are two branches:

1. Select a representative $i_1 \in \mathcal{O}$ and assign the variable $x_{i_1} = 0$.
2. For all representatives $i \in \mathcal{O}$, assigne the variable $x_i = 1$.

Notice that if there are non-trivial automorphisms, then there exists an orbit of size at least two. This makes the second branch assign more than one variable in a given time. The reason this process works is that assigning $x_i = 0$ for *any* representative $i \in \mathcal{O}$ leads to the same variable assignment up to isomorphism. Since trying any *one* variable a zero is the same, we can just try one and in the second branch we can set all of the variables to the other value. Figure 10.1 shows the difference between branch-and-bound and orbital branching.



(a) The branch-and-bound algorithm.

(b) The orbital branching algorithm.

Figure 10.1: Comparing Branch-and-Bound with Orbital Branching.

Algorithm 10.2 gives a full description of the orbital branching method. The algorithm follows a similar pattern to Algorithm 10.1 but uses a subroutine, denoted $\text{UnassignedOrbit}_P(\mathbf{x})$, to select an orbit of unassigned variables instead of just a single variable. The correctness of this algorithm is given by Theorem 10.1.

Algorithm 10.2 $\text{OrbitalBranching}_P(\mathbf{x})$

```
if $\text{Detect}_P(\mathbf{x})$ then
  return
end if
$\mathbf{y} \leftarrow \text{Propagate}_P(\mathbf{x})$
if $\mathbf{y} \in \{0,1\}^m$ then
  if $P(\mathbf{y}) = 1$ then
    Output $\mathbf{y}$
  end if
  return
end if
$\mathcal{O} \leftarrow \text{UnassignedOrbit}_P(\mathbf{x})$
$i_1 \leftarrow \min\{i \in \mathcal{O}\}$
$y_{i_1} \leftarrow 0$
call $\text{OrbitalBranching}_P(\mathbf{y})$
for all $i \in \mathcal{O}$ do
  $y_i \leftarrow 1$
end for
call $\text{OrbitalBranching}_P(\mathbf{y})$
return
```

Theorem 10.1 (Ostrowski, Linderoth, Rossi, Smriglio [102]). *The orbital branching algorithm outputs at least one solution* $\mathbf{x}$ *with* $P(\mathbf{x}) = 1$ *from every orbit of vectors under* $\mathrm{Aut}(P)$*.*

Proof. We shall prove that a call to the recursive algorithm $\text{OrbitalBranching}_P(\mathbf{x})$ for a given variable assignment $\mathbf{x}$ shall output at least one vector from every orbit of vectors $\mathbf{z} \in \{0,1\}^m$ (under the action of $\mathrm{Aut}(P)$) so that $\mathbf{x} \preceq \mathbf{z}$ and $P(\mathbf{z}) = 1$. We proceed by induction on the number of unassigned variables on the vector $\mathbf{y}$ given by $\text{Propagate}_P(\mathbf{x})$.

If $\mathbf{y}$ has no unassigned variables, then either $P(\mathbf{y}) = 1$ and $\mathbf{y}$ is output, or $P(\mathbf{y}) = 0$ and $\mathbf{y}$ is not output. By the assumptions on Propagate_P, $\mathbf{y}$ is the only extension of $\mathbf{x}$ with $P(\mathbf{y}) = 1$.

Suppose $\mathbf{y}$ has k unassigned variables and for any variable assignment $\mathbf{z}$ with at most $k-1$ unassigned variables, $\text{OrbitalBranching}_P(\mathbf{z})$ outputs at least one vector from every orbit of solutions to P that extends $\mathbf{z}$. Let $\mathcal{O}$ be the orbit from $\text{UnassignedOrbit}_P(\mathbf{y})$ and $i_1 < i_2 < \cdots < i_\ell$ be the indices in $\mathcal{O}$.

For any integer $j \in \{1,\dots,m\}$, let $\mathbf{z}^{(j)}$ have value $z_i^{(j)} = \begin{cases} 0 & i = j \\ y_i & \text{otherwise} \end{cases}$. By induction, $\text{OrbitalBranching}_P(\mathbf{z}^{(i_1)})$ outputs at least one vector from every orbit of extensions of $\mathbf{y}$ where the i_1th variable takes value 0.

Let $\mathbf{z}^{(\mathcal{O})}$ have value $z_i^{(\mathcal{O})} = \begin{cases} 1 & i \in \mathcal{O} \\ y_i & \text{otherwise} \end{cases}$. $\text{OrbitalBranching}_P(\mathbf{z}^{(\mathcal{O})})$ outputs at least one vector from every orbit of extensions of $\mathbf{y}$ where the variables with index in $\mathcal{O}$ take value 1.

Let $\mathbf{w}$ be an extension of $\mathbf{y}$ ($\mathbf{y} \preceq \mathbf{w}$) so that $P(\mathbf{w}) = 1$. If $\mathbf{z}^{(i_1)} \preceq \mathbf{w}$ or $\mathbf{z}^{(\mathcal{O})} \preceq \mathbf{w}$, then some vector from the orbit of $\mathbf{w}$ is output by these calls to $\text{OrbitalBranching}_P$.

Otherwise, since $\mathbf{z}^{(\mathcal{O})} \not\preceq \mathbf{w}$, there is some variable $i_j \in \mathcal{O}$ so that $w_{i_j} = 0$. Since $\mathbf{z}^{(i_1)} \not\preceq \mathbf{w}$, $j \neq 1$. By definition of $\mathcal{O}$, there must be a permutation $\sigma \in \text{Aut}_{\mathbf{y}}(P)$ so that $\sigma(i_j) = i_1$. Therefore, $\mathbf{w}_\sigma$ has the i_1th variable with value 0 and $\mathbf{z}^{(i_1)} \preceq \mathbf{w}_\sigma$. Hence, some vector in orbit with $\mathbf{w}_\sigma$ is output by the first call to $\text{OrbitalBranching}_P$. Since $\mathbf{w}_\sigma$ is in orbit with $\mathbf{w}$, the claim is satisfied. □

Example. In the case of variable assignments corresponding to trigraphs, the first branch in the orbital branching algorithm selects a pair from the orbit and assigns that pair to be a nonedge. No matter which representative pair is selected from the orbit, the resulting trigraph is isomorphic to any other choice of representative. Therefore, an arbitrary representative suffices. Also, any assignment of edges and nonedges to this trigraph so that one of the representatives becomes a nonedge is isomorphic to some trigraph where our selected representative is a nonedge. Therefore, after checking all extensions where this representative is a nonedge we may assume that every pair from the orbit is an edge for other extensions.

10.4 Branching Rules

In the previous description of the orbital branching algorithm, we left the algorithm for $\text{UnassignedOrbit}_P(\mathbf{x})$ to be an arbitrary algorithm. An instance of this algorithm is called a *branching rule*, as it dictates which variables are assigned in the current branch. In the original work by Ostrowski et al. [102], a lot of attention was given to these branching rules. One reason is that they were working with combinatorial optimization problems where the “bounding" part of branch-and-bound is particularly useful. Hence, several of the branching rules used a continuous relaxation of the problem as advice to choosing an orbit.

Since this thesis considers exhaustively generating all feasible solutions to a combinatorial problem where the constraints are rarely simple to describe in a continuous setting, we ignore the original branching rules. Instead, we focus on the example of searching over trigraphs for our rules.

Example. Suppose we wish to select an orbit of unassigned pairs from a trigraph G. Table 10.1 contains a few potential branching rules along with positive and negative effects.

10.5 Orbital Branching and Canonical Deletion

From our example of searching for strongly regular graphs, we see that orbital branching can be used to search for combinatorial objects. This brings the technique into the combinatorial realm, where it can compete with canonical deletion. Orbital branching may seem weaker than canonical deletion, since we have no guarantee that every object is visited at most once. Even worse, when using orbital branching and you generate an object with no automorphisms, the technique becomes no better than brute-force search. However, there are situations when orbital branching is significantly more efficient than canonical deletion.

One reason the orbital branching technique may work better than canonical deletion is that the techniques have strengths in two opposite areas. Canonical deletion focuses on removing all isomorphic duplicates, but there is no known method to incorporate constraint propagation with canonical deletion. Orbital branching is built to naturally allow constraint propagation, but it only reduces the number of isomorphic duplicates.

Another reason is that the augmentation step for orbital branching involves exactly one automorphism calculation, and two augmentations. In canonical deletion, every possible augmentation must be attempted (up to isomorphism) and checked to see if it is a canonical augmentation. Depending on the augmentation step, this can be a very costly operation.

In Chapter 11, we extend the orbital branching technique with a customized augmentation for a specific problem. This augmentation integrates well with orbital branching, but our attempt to integrate it with canonical deletion was less efficient. Our implementation with orbital branching is a very efficient algorithm; by executing the algorithm, we discovered several new graphs with the desired properties.

Table 10.1: List of example branching rules.

Rule	How it Works	Pros	Cons
LargestOrbit	Select an orbit of highest order.	Maximizes amount of change in second branch.	Does not concern constraints. Can quickly remove symmetry from the graph.
LargestConnOrbit	Maintain a set $S \subseteq V(G)$ of vertices where every vertex in S is contained in at least one assigned pair. Select an orbit where all pairs have both endpoints in S (if possible) or at least one endpoint in S (otherwise). From these orbits, select the one of largest order.	Builds graphs by filling in all pairs from a given set first. Attempts to maximize change in second branch.	Some of the orbits may be small when S is nearly full, resulting in low symmetry for later orbit calculations.
MaxDegreeSumOrbit	Select an orbit of pairs uv where the degree and non-degree sums of u and v are maximum. Break ties by orbit size.	Attempts to maximize tightness of constraints by fully specifying the edges and nonedges at vertices of high degree and nondegree.	The first branch leads to a single vertex with high nondegree, which leads to the orbit sizes decreasing rapidly.

Chapter 11

Uniquely K_r-Saturated Graphs

A graph G is *uniquely H-saturated* if there is no subgraph of G isomorphic to H, and for all edges e in the complement of G there is a unique subgraph in $G + e$ isomorphic to H[4]. Uniquely H-saturated graphs were introduced by Cooper, Lenz, LeSaulnier, Wenger, and West [34] where they classified uniquely C_k-saturated graphs for $k \in \{3, 4\}$; in each case there is a finite number of graphs. Wenger [144, 145] classified the uniquely C_5-saturated graphs and proved that there do not exist any uniquely C_k-saturated graphs for $k \in \{6, 7, 8\}$.

In this chapter, we focus on the case where $H = K_r$, the complete graph of order r. Usually K_r is the first graph considered for extremal and saturation problems. However, we find that classifying all uniquely K_r-saturated graphs is far from trivial, even in the case that $r = 4$.

Previously, few examples of uniquely K_r-saturated graphs were known, and little was known about their properties. We adapt the computational technique of orbital branching into the graph theory setting to search for uniquely K_r-saturated graphs. Orbital branching was originally introduced by Ostrowski, Linderoth,

---

[4] A technicality: for all $t < n(H)$, the complete graph K_t is trivially uniquely H-saturated. We adopt the convention that always $n(G) \geq n(H)$.

Rossi, and Smriglio [102] to solve symmetric integer programs. We further extend the technique to use augmentations which are customized to this problem. By executing this search, we found several new uniquely K_r-saturated graphs for $r \in \{4, 5, 6, 7\}$ and we provide constructions of these graphs to understand their structure. One of the graphs we discovered is a Cayley graph, which led us to design a search for Cayley graphs which are uniquely K_r-saturated. Motivated by these search results, we construct two new infinite families of uniquely K_r-saturated Cayley graphs.

Erdős, Hajnal, and Moon [43] studied the minimum number of edges in a K_r-saturated graph. They proved that the only extremal examples are the graphs formed by adding $r - 2$ dominating vertices to an independent set; these graphs are also uniquely K_r-saturated. However, if G is uniquely K_r-saturated and has a dominating vertex, then deleting that vertex results in a uniquely K_{r-1}-saturated graph. To avoid the issue of dominating vertices, we define a graph to be *r-primitive* if it is uniquely K_r-saturated and has no dominating vertex. Understanding which r-primitive graphs exist is fundamental to characterizing uniquely K_r-saturated graphs.

Since $K_3 \cong C_3$, the uniquely K_3-saturated graphs were proven by Cooper et al. [34] to be stars and Moore graphs of diameter two. While stars are uniquely K_3-saturated, they are not 3-primitive. The Moore graphs of diameter two are exactly the 3-primitive graphs; Hoffman and Singleton [64] proved there are a finite number of these graphs.

David Collins and Bill Kay discovered the only previously known infinite family of r-primitive graphs, that of complements of odd cycles: $\overline{C_{2r-1}}$ is r-primitive. Collins and Cooper discovered two more 4-primitive graphs of orders 10 and 12 [31]. These two graphs are described in detail in Section 11.4.

One feature of all previously known r-primitive graphs is that they are all regular. Since proving regularity has been instrumental in previous characterization proofs (such as [34, 64]), there was a hope that r-primitive graphs are regular. However, we present a counterexample: a 5-primitive graph on 16 vertices with minimum degree 8 and maximum degree 9.

The major open question in this area concerns the number of r-primitive graphs for a fixed r.

Conjecture 11.1 (Cooper [31]). *For each $r \geq 3$, there are a finite number of r-primitive graphs.*

This conjecture is true for $r = 3$ [64] and otherwise completely open. Before this work, it was not even known if there was more than one r-primitive graph for any $r \geq 5$. After we discovered the graphs in this work (which lack any common structure and sometimes appear very strange), we are unsure the conjecture holds even for $r = 4$.

In Section 11.1, we briefly summarize our results, including our computational method, the new sporadic r-primitive graphs, and our new algebraic constructions.

11.1 Summary of results

Our results have three main components. First, we develop a computational method for generating uniquely K_r-saturated graphs. Then, based on one of the generated examples, we construct two new infinite families of uniquely K_r-saturated graphs. Finally, we describe all known uniquely K_r-saturated graphs, including the nine new sporadic[1] graphs found using the computational method.

11.1.1 Computational method

In Section 11.2, we develop a new technique for exhaustively searching for uniquely K_r-saturated graphs on n vertices. The search is based on the technique of *orbital branching* originally developed for use in symmetric integer programs by Ostrowski, Linderoth, Rossi, and Smriglio [102, 103]. We focus on the case of constraint systems with variables taking value in $\{0, 1\}$. The orbital branching is based on the standard branch-and-bound technique where an unassigned variable is selected and the search branches into cases for each possible value for that variable. In a symmetric constraint system, the automorphisms of the variables which preserve the constraints and variable values generate orbits of variables. Orbital branching selects an orbit of variables and branches in two cases. The first branch selects an arbitrary representative variable is selected from the orbit and set to zero. The second branch sets all variables in the orbit to one.

We extend this technique to be effective to search for uniquely K_r-saturated graphs. We add an additional constraint to partial graphs: if a pair v_i, v_j is a non-edge in G, then there is a unique set $S_{i,j}$ containing $r-2$ vertices so that $S_{i,j}$ is a clique and every edge between $\{v_i, v_j\}$ and $S_{i,j}$ is included in G. This guarantees that there is at least one copy of K_r in $G + v_iv_j$ for all assignments of edges and non-edges to the remaining unassigned pairs. The orbital branching method is customized to enforce this constraint, which leads to multiple edges being added to the graph in every augmentation step. By executing this algorithm, we found 10 new r-primitive graphs.

---

[1] We call a graph *sporadic* if it has not yet been extended to an infinite family. Therefore, even though our search found 10 new graphs, one extended to an infinite family and so is not sporadic.

11.1.2 New r-primitive graphs

For $r \in \{4, 5, 6, 7, 8\}$, we used this method to exhaustively search for uniquely K_r-saturated graphs of order at most N_r, where $N_4 = 20$, $N_5 = N_6 = 16$, and $N_7 = N_8 = 17$. Table 11.1 lists the r-primitive graphs that were discovered in this search. Most graphs do not fit a short description and are labeled $G_N^{(i)}$, where N is the number of vertices and $i \in \{A, B, C\}$ distinguishes between graphs of the same order.

n	13	15	16	16	17	18
r	4	6	5	6	7	4
Graphs	G_{13}, Paley(13)	$G_{15}^{(A)}, G_{15}^{(B)}$	$G_{16}^{(A)}, G_{16}^{(B)}$	$G_{16}^{(C)}$	$\overline{C}(\mathbb{Z}_{17}, \{1, 4\})$	$G_{18}^{(A)}, G_{18}^{(B)}$

Table 11.1: Newly discovered r-primitive graphs.

In all, ten new graphs were discovered to be uniquely K_r-saturated by this search. Explicit constructions of these graphs are given in Section 11.4. Two graphs found by computer search are vertex-transitive and have a prime number of vertices. Recall by Proposition 2.11 that vertex-transitive graphs with a prime number of vertices are Cayley graphs. One vertex-transitive 4-primitive graph is the Paley graph of order 13 (see [105]). The other vertex-transitive graph is 7-primitive on 17 vertices and is 14 regular. However, it is easier to understand its complement, which is the Cayley graph for $\mathbb{Z}_{17}$ generated by 1 and 4. This graph is listed as $\overline{C}(\mathbb{Z}_{17}, \{1, 4\})$ in Table 11.1 and is the first example of our new infinite families, described below.

11.1.3 Algebraic Constructions

For a finite group Γ and a generating set $S \subseteq \Gamma$, let $C(\Gamma, S)$ be the *Cayley graph* for Γ generated by S: the vertex set is Γ and two elements $x, y \in \Gamma$ are adjacent if and only if there is a $z \in S$ where $x = yz$ or $x = yz^{-1}$. When $\Gamma \cong \mathbb{Z}_n$, the resulting graph is also called a *circulant graph*. The cycle C_n can be described as the Cayley graph of $\mathbb{Z}_n$ generated by 1. Since $\overline{C_{2r-1}}$ is r-primitive and we discovered a graph on 17 vertices whose complement is a Cayley graph with two generators, we searched for r-primitive graphs when restricted to complements of Cayley graphs with a small number of generators.

For a finite group Γ and a set $S \subseteq \Gamma$, the *Cayley complement* $\overline{C}(\Gamma, S)$ is the complement of the Cayley graph $C(\Gamma, S)$. We restrict to the case when $\Gamma = \mathbb{Z}_n$ for some n, and the use of the complement allows us to use a small number of generators while generating dense graphs.

We search for r-primitive Cayley complements by enumerating all small generator sets S, then iterate over n where $n \geq 2\max S + 1$ and build $\overline{C}(\mathbb{Z}_n, S)$. If $\overline{C}(\mathbb{Z}_n, S)$ is r-primitive for any r, it must be for $r = \omega(\overline{C}(\mathbb{Z}_n, S)) + 1$, so we compute this r using Niskanen and Östergård's *cliquer* library [100]. Also using *cliquer*, we count the number of r-cliques in $\overline{C}(\mathbb{Z}_n, S) + \{0, i\}$ for all $i \in S$. Since $\overline{C}(\mathbb{Z}_n, S)$ is vertex-transitive, this provides sufficient information to determine if $\overline{C}(\mathbb{Z}_n, S)$ is r-primitive. The successful parameters for r-primitive Cayley complements with g generators are given in Tables 11.1(a) ($g = 2$), 11.1(b) ($g = 3$), and 11.1(c) ($g \geq 4$).

For two and three generators, a pattern emerged in the generating sets and interpolating the values of n and r resulted in two infinite families of r-primitive graphs:

Theorem 11.2. *Let* $t \geq 2$ *and set* $n = 4t^2 + 1, r = 2t^2 - t + 1$. *Then,* $\overline{C}(\mathbb{Z}_n, \{1, 2t\})$ *is r-primitive.*

Theorem 11.3. *Let* $t \geq 2$ *and set* $n = 9t^2 - 3t + 1, r = 3t^2 - 2t + 1$. *Then,* $\overline{C}(\mathbb{Z}_n, \{1, 3t - 1, 3t\})$ *is r-primitive.*

(a) Two Generators

t	S	r	n
2	$\{1,4\}$	7	17
3	$\{1,6\}$	16	37
4	$\{1,8\}$	29	65
5	$\{1,10\}$	46	101
6	$\{1,12\}$	67	145

(b) Three Generators

t	S	r	n
2	$\{1,5,6\}$	9	31
3	$\{1,8,9\}$	22	73
4	$\{1,11,12\}$	41	133
5	$\{1,14,15\}$	66	211
6	$\{1,17,18\}$	97	307

(c) Sporadic Cayley Complements

g	S	r	n
3	$\{1,3,4\}$	4	13
4	$\{1,5,8,34\}$ $\{1,11,18,34\}$	28	89
5	$\{1,5,14,17,25\}$	19	71
5	$\{1,6,14,17,36\}$	27	101
6	$\{1,6,16,22,35,36\}$	21	97
6	$\{1,8,23,26,43,64\}$	54	185
7	$\{1,20,23,26,30,32,34\}$	15	71
8	$\{1,8,12,18,22,27,33,47\}$	20	97
9	$\{1,4,10,16,25,27,33,40,64\}$	28	133

Table 11.2: Cayley complement parameters for r-primitive graphs over $\mathbb{Z}_n$.

An important step to proving these Cayley complements are r-primitive is to compute the clique number. Computing the clique number or independence number of a Cayley graph is very difficult, as many papers study this question [52, 74], including in the special cases of circulant graphs [13, 23, 65, 148] and Paley graphs [11, 15, 22, 30]. Our enumerative approach to Theorem 11.2 and discharging approach to Theorem 11.3 provide a new perspective on computing these values.

It remains an open question if an infinite family of Cayley complements $\overline{C}(\mathbb{Z}_n, S)$ exist for a fixed number of generators $g = |S|$ where $g \geq 4$. For all known constructions with $g \neq 4$, observe that the generators are roots of unity in $\mathbb{Z}_n$ with $x^{2g} \equiv 1 \pmod{n}$ for each generator x. Being roots of unity is not a sufficient condition for the Cayley complement to be r-primitive, but this observation may lead to algebraic techniques to build more infinite families of Cayley complements.

Determining the maximum density of a clique and independent set for infinite Cayley graphs (i.e., $\overline{C}(\mathbb{Z}, S)$, where S is finite) would be useful for providing bounds on the finite graphs. Further, such bounds could be used by algorithms to find and count large cliques and independent sets in finite Cayley graphs.

11.2 Orbital branching using custom augmentations

In this section, we describe a computational method to search for uniquely K_r-saturated graphs. We shall build graphs piece-by-piece by selecting pairs of vertices to be edges or non-edges.

To store partial graphs, we use the notion of a *trigraph*, defined by Chudnovsky [29] and used by Martin and Smith [90]. A *trigraph* T is a set of n vertices $v_1, \dots, v_n$ where every pair v_iv_j is colored black, white, or gray. The black pairs represent edges, the white edges represent non-edges, and the gray edges are unassigned pairs. A graph G is a *realization* of a trigraph T if all black pairs of T are edges of G and all white pairs of T are non-edges of G. Essentially, a realization is formed by assigning the gray pairs to be edges or non-edges. In this way, we consider a graph to be a trigraph with no gray pairs.

Non-edges play a crucial role in the structure of uniquely K_r-saturated graphs. Given a trigraph T and a pair v_iv_j, a set S of $r-2$ vertices is a *K_r-completion* for v_iv_j if every pair in $S \cup \{v_i, v_j\}$ is a black edge, except for possibly v_iv_j. Observe that a K_r-free graph is uniquely K_r-saturated if and only if every non-edge has a unique K_r-completion.

We begin with a completely gray trigraph and build uniquely K_r-saturated graphs by adding black and white pairs. If we can detect that no realization of the current trigraph can be uniquely K_r-saturated, then we backtrack and attempt a different augmentation. The first two constraints we place on a trigraph T are:

(C1) There is no black r-clique in T.

(C2) Every vertex pair has at most one black K_r-completion.

It is clear that a trigraph failing either of these conditions will fail to have a uniquely K_r-saturated realization.

We use the symmetry of trigraphs to reduce the number of isomorphic duplicates. The *automorphism group* of a trigraph T is the set of permutations of the vertices that preserve the colors of the pairs. These automorphisms are computed with McKay's *nauty* library [61, 93] through the standard method of using a layered graph.

11.2.1 Orbital Branching

Ostrowski, Linderoth, Rossi, and Smriglio introduced the technique of *orbital branching* for symmetric integer programs with 0-1 variables [102] and for symmetric constraint systems [103]. Orbital branching extends the standard branch-and-bound strategy of combinatorial optimization by exploiting symmetry to reduce the search space. We adapt this technique to search for graphs by using trigraphs in place of variable assignments.

Given a trigraph T, compute the automorphism group and select an *orbit* $\mathcal{O}$ of gray pairs. Since every representative pair in $\mathcal{O}$ is identical in the current trigraph, assigning any representative to be a white pair leads to isomorphic trigraphs. Hence, we need only attempt assigning a single pair in $\mathcal{O}$ to be white. The natural complement of this operation is to assign all pairs in $\mathcal{O}$ to be black. Therefore, we branch on the following two options:

- *Branch 1*: Select any pair in $\mathcal{O}$ and assign it the color white.
- *Branch 2*: Assign all pairs in $\mathcal{O}$ the color black.

A visual representation of this branching process is presented in Figure 11.1(a).

An important part of this strategy is to select an appropriate orbit. The selection should attempt to maximize the size of the orbit (in order to exploit the number of pairs assigned in the second branch) while preserving as much symmetry as possible (in order to maintain large orbits in deeper stages of the search). It is difficult to determine the appropriate branching rule *a priori*, so it is beneficial to implement and compare the performance of several branching rules.

This use of orbital branching suffices to create a complete search of all uniquely K_r-saturated graphs, but is not very efficient. One significant drawback to this technique is the fact that the constraints (C1) and (C2) rely on black pairs forming cliques. In the next section, we create a custom augmentation step that is aimed at making these constraints trigger more frequently and thereby reducing the number of generated trigraphs.

11.2.2 Custom augmentations

We search for uniquely K_r-saturated graphs by enforcing at each step that every white pair has a unique K_r-completion. We place the following constraints on a trigraph:

(C3) If v_iv_j is a white edge, then there exists a unique K_r-completion $S \subseteq \{v_1, \dots, v_n\}$ for v_iv_j.

To enforce the constraint (C3), whenever we assign a white pair we shall also select a set of $r - 2$ vertices to be the K_r-completion and assign the appropriate pairs to be black. The orbital branching procedure was built to assign only one white pair in a given step, so we can attempt all possible K_r-completions for that pair. However, if we perform an automorphism calculation and only augment for one representative set from every orbit of these sets, we can reduce the number of isomorphic duplicates.

We follow a two-stage orbital branching procedure. In the first stage, we select an orbit $\mathcal{O}$ of gray pairs. Either we select a representative pair $v_{i'}v_{j'} \in \mathcal{O}$ to set to white or assign v_iv_j to be black for all pairs $v_iv_j \in \mathcal{O}$. In order to guarantee constraint (C3), the white pair must have a K_r-completion. We perform a second automorphism computation to find $\text{Stab}_{\{v_{i'},v_{j'}\}}(T)$, the set of automorphisms which set-wise stabilize the pair $v_{i'}v_{j'}$. Then, we compute all orbits of $(r-2)$-subsets S in $\{v_1, \dots, v_n\} \setminus \{v_i, v_j\}$ under the action of $\text{Stab}_{\{v_{i'},v_{j'}\}}(T)$. The second stage branches on each set-orbit $\mathcal{A}$, selects a single representative $S' \in \mathcal{A}$ and adds all necessary black pairs to make S' be a K_r-completion for $v_{i'}v_{j'}$. If at any point we attempt to assign a white pair to be black, that branch fails and we continue with the next set-orbit.

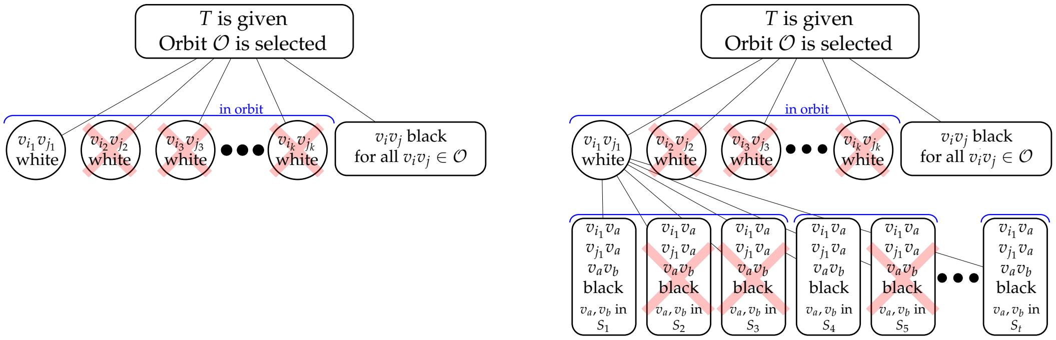
This branching process on a trigraph T is:

- *Branch 1*: Select any pair $v_{i_1}v_{j_1} \in \mathcal{O}$ to be white.
  - *Sub-Branch*: For *every* orbit $\mathcal{A}$ of $(r-2)$-subsets of $V(T) \setminus \{v_{i_1}, v_{i_2}\}$ under the action of $\text{Stab}_{\{v_{i_1},v_{j_1}\}}(T)$, select any set $S \in \mathcal{A}$, assign $v_{i_1}v_a$, $v_{j_1}v_a$, and v_av_b to be black for all $v_a, v_b \in S$.
- *Branch 2*: Set v_iv_j to be black for all pairs $v_iv_j \in \mathcal{O}$.

The full algorithm to output all uniquely K_r-saturated graphs on n vertices is given as the recursive method SaturatedSearch(n, r, T) in Algorithm 11.1, while

Algorithm 11.1 SaturatedSearch(n, r, T)

```
if $T$ contains a black $r$-clique then
    Constraint (C1) fails.
    return
else if there exists a pair $v_iv_j$ with two $K_r$-completions in $T$ then
    Constraint (C2) fails.
    return
else if there are no gray pairs then
    The trigraph $T$ is uniquely $K_r$-saturated.
    Output $T$.
    return
end if
Propagate under constraint (C1).
for all gray pairs $v_iv_j$ do
    if $v_iv_j$ has a $K_r$-completion in $T$ then
        Assign $v_iv_j$ to be white.
    end if
end for
Compute pair orbits $\mathcal{O}_1, \mathcal{O}_2, \dots,$ of gray pairs $\{i, j\}$.
Select an orbit $\mathcal{O}_k$ using the branching rule.
Branch 1.
Let $v_{i'}v_{j'}$ be a representative of $\mathcal{O}_k$.
Compute orbits $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_\ell$ of $(r-2)$-vertex sets in $\{v_1, \dots, v_n\} \setminus \{v_{i'}, v_{j'}\}$.
for $t \in \{1, \dots, \ell\}$ do
    Let $S$ be a representative of $\mathcal{A}_t$.
    if $v_{i'}v_a, v_{j'}v_a, v_av_b$ not white for all $a, b \in S$ then
        Sub-Branch: Create $T'$ from $T$ by assigning $v_{i'}v_a, v_{j'}v_a, v_av_b$ to be black for all $a, b \in S$.
        call SaturatedSearch($n, r, T'$)
    end if
end for
Branch 2: Create $T''$ from $T$ by assigning $v_iv_j$ to be black for all $v_iv_j \in \mathcal{O}_k$.
call SaturatedSearch($n, r, T''$)
return
```



(a) Standard orbital branching.

(b) Custom augmentations.

Figure 11.1: Visual description of the branching process.

the branching procedure is represented in Figure 11.1(b). The algorithm is initialized using the trigraph corresponding to a single white pair with a K_r-completion. The first step of every recursive call to SaturatedSearch(n, r, T) is to verify the constraints (C1) and (C2). If either constraint fails, no realization of the current trigraph can be uniquely K_r-saturated, so we return. After verifying the constraints, we perform a simple propagation step: If a gray pair $\{i, j\}$ has a K_r-completion we assign that pair to be white. We can assume that this pair is a white edge in order to avoid violation of (C1), and this assignment satisfies (C3).

The missing component of this algorithm is the *branching rule*: the algorithm that selects the orbit of unassigned pairs to use in the first stage of the branch. Based on experimentation, the most efficient branching rule we implemented only considers pairs where both vertices are contained in assigned pairs (if they exist) or pairs where one vertex is contained in an assigned pair (which must exist, otherwise), and selects from these pairs the orbit of largest size. This choice would guarantee the branching orbit has maximum interaction with currently assigned edges while maximizing the effect of assigning all representatives to be edges in the second branch.

n	$r = 4$	$r = 5$	$r = 6$	$r = 7$	$r = 8$
10	0.10 s	0.37 s	0.13 s	0.01 s	0.01 s
11	0.68 s	5.25 s	1.91 s	0.28 s	0.09 s
12	4.58 s	1.60 m	25.39 s	1.97 s	1.12 s
13	34.66 s	34.54 m	6.53 m	59.94 s	20.03 s
14	4.93 m	10.39 h	5.13 h	20.66 m	2.71 m
15	40.59 m	23.49 d	10.08 d	12.28 h	1.22 h
16	6.34 h	1.58 y	1.74 y	34.53 d	1.88 d
17	3.44 d			8.76 y	115.69 d
18	53.01 d				
19	2.01 y				
20	45.11 y				

Table 11.3: CPU times to search for uniquely K_r-saturated graphs of order n. Execution times from the Open Science Grid [107] using the University of Nebraska Campus Grid [143]. The nodes available on the University of Nebraska Campus Grid consist of Xeon and Opteron processors with a range of speed between 2.0 and 2.8 GHz.

11.2.3 Implementation, Timing, and Results

The full implementation is available as the *Saturation* project in the *SearchLib* software library[2]. More information for the implementation is given in the *Saturation* User Guide, available with the software. In particular, the user guide details the methods for verifying the constraints (C1), (C2), and (C3). When $r \in \{4, 5\}$, we monitored clique growth using a custom data structure, but when $r \geq 6$ an implementation using Niskanen and Östergård's *cliquer* library [100] was more efficient.

Our computational method is implemented using the *TreeSearch* library [122], which abstracts the search structure to allow for parallelization to a cluster or grid. Table 11.3 lists the CPU time taken by the search for each $r \in \{4, 5, 6, 7, 8\}$ and $10 \leq n \leq N_r$ (where $N_4 = 20$, $N_5 = N_6 = 16$, and $N_7 = N_8 = 17$) until the search became intractable for $n = N_r + 1$. Table 11.1 lists the r-primitive graphs of these

---

[2] *SearchLib* is available online at `http://www.math.unl.edu/~s-dstolee1/SearchLib/`

sizes. Constructions for the graphs are given in Section 11.4.

11.3 Infinite families of r-primitive graphs using Cayley graphs

In this section, we prove Theorems 11.2 and 11.3, which provide our two new infinite families of r-primitive graphs. We begin with some definitions that are common to both proofs.

Fix an integer n, a generator set $S \subseteq \mathbb{Z}_n$, and a Cayley complement $G = \overline{C}(\mathbb{Z}_n, S)$. For a set $X \subseteq \mathbb{Z}_n$ with $r = |X|$, list the elements of X as $0 \leq x_0 \leq x_1 \leq \cdots \leq x_{r-1} < n$. We shall assume that X is a clique in G (or in $G + e$ for some non-edge $e \in E(\overline{G})$).

Considering X as a subset of $\mathbb{Z}_n$, we let the kth *block* B_k be the elements of $\mathbb{Z}_n$ increasing from x_k (inclusive) to x_{k+1} (exclusive): $B_k = \{x_k, x_k + 1, \dots, x_{k+1} - 1\}$. Note that $|B_k| = x_{k+1} - x_k$; we call a block of size s an *s-block*. For an integer $t \geq 1$ and $j \in \{0, \dots, r-1\}$, the jth *frame* F_j is the collection of t consecutive blocks in increasing order starting from B_j: $F_j = \{B_j, B_{j+1}, \dots, B_{j+\ell-1}\}$. A *frame family* is a collection $\mathcal{F}$ of frames.

If F is a frame (or any set of blocks), define $\sigma(F) = \sum_{B_j \in F} |B_j|$, the number of elements covered by the blocks in F.

Observation 11.4. If X is a clique in $\overline{C}(\mathbb{Z}_n, S)$ and F is a set of consecutive blocks in X, then $\sigma(F) \notin S$.

11.3.1 Two Generators

Theorem 11.2. *Let $t \geq 1$, and set $n = 4t^2 + 1$, $r = 2t^2 - t + 1$. Then, $\overline{C}(\mathbb{Z}_n, \{1, 2t\})$ is r-primitive.*

Proof. Let $G = \overline{C}(\mathbb{Z}_n, \{1, 2t\})$. Note that G is regular of degree $n - 5$. If $t = 1$, then $n = 5$, G is an empty graph, and $r = 2$, and empty graphs are 2-primitive. Therefore, we consider $t \geq 2$.

Claim 11.5. *For a clique X, every frame F_j has at least one block of size at least three, and $\sigma(F_j) \geq 2t + 1$.*

All blocks B_j have at least two elements, since no pair of elements in X may be consecutive in $\mathbb{Z}_n$, so $\sigma(F_j) \geq 2t$. If for all $B_k \in F_j$ the block length $|B_k|$ is exactly two, then $\sigma(F_j) = 2t \in S$. Hence, there is some $B_k \in F_j$ so that $|B_k| \geq 3$ and $\sigma(F_j) \geq 2t + 1$.

We now prove there is no r-clique in G.

Claim 11.6. *$\omega(G) < r$.*

Suppose $X \subseteq \mathbb{Z}_n$ is a clique of order r in G. Let $\mathcal{F}$ be the frame family of all frames ($\mathcal{F} = \{F_j : j \in \{0, \dots, r-1\}\}$) and consider the sum $\sum_{j=0}^{r-1} \sigma(F_j)$. Using the bound $\sigma(F_j) \geq 2t + 1$, we have this sum is at least $(2t+1)r$. Each block length $|B_k|$ is counted in t evaluations of $\sigma(F_j)$ (for $j \in \{k - t + 1, k - t + 2, \dots, k\}$). This sum counts each element of $\mathbb{Z}_n$ exactly t times, giving value tn. This gives $tn = \sum_{j=0}^{r-1} \sigma(F_j) \geq (2t+1)r$, but $tn = 4t^3 + t < 4t^3 + t + 1 = (2t+1)r$, a contradiction. Hence, X does not exist, proving the claim.

To prove unique saturation, we consider only the non-edge $\{0, 1\}$ since G is vertex-transitive and the map $x \mapsto -2tx$ is an automorphism of G mapping the edge $\{0, 2t\}$ to $\{0, -4t^2\} \equiv \{0, 1\} \pmod{n}$.

Claim 11.7. *There is a unique r-clique in $G + \{0, 1\}$.*

We may assume $X = \{0, 1, x_2, \dots, x_{r-1}\}$ is an r-clique in $G + \{0, 1\}$. We use the frame family $\mathcal{F}$ defined as

$$\mathcal{F} = \left\{F_{jt+1} : j \in \{0, \dots, 2t-2\}\right\}.$$

Note that $\mathcal{F}$ contains $2t-1$ *disjoint* frames containing disjoint blocks, and the block $B_0 = \{x_0\}$ is not contained in any frame within $\mathcal{F}$. Hence, $n-1 = \sum_{F\in\mathcal{F}} \sigma(F)$. By Claim 11.5, we know that every frame $F \in \mathcal{F}$ has $\sigma(F) \geq 2t+1$. This lower bound gives $\sum_{F\in\mathcal{F}} \sigma(F) \geq (2t+1)(2t-1) = n-2$. Thus, considering $\sigma(F)$ as an integer variable for each $F \in \mathcal{F}$, all solutions to the integer program with constraints $\sigma(F) \geq 2t+1$ and $\sum_{F\in\mathcal{F}} \sigma(F) = n-1$ have $\sigma(F) = 2t+1$ for all $F \in \mathcal{F}$ except a unique $F' \in \mathcal{F}$ with $\sigma(F') = 2t+2$.

The frame F' has two possible ways to attain $\sigma(F') = 2t+2$: (a) have two blocks of size three, or (b) have one block of size four. However, if F' has a block of size four, then there is a 2-block $B_j \in F'$ on one end of F' where $\sigma(F' \setminus \{B_j\}) = 2t \in S$, a contradiction. Thus, F' has two blocks of size three. In addition, if F' has fewer than $t-2$ blocks of size two between the two blocks of size three, then there is a pair $x, y \in X$ with $y = x + 2t$. Therefore, F' has two blocks of size three and they are the first and last blocks of F'.

This frame family demonstrates the following properties of X. First, there are exactly $2t$ blocks of size three ($2t-2$ frames have exactly one and F' has exactly two). Second, there is no set of t consecutive blocks of size two. Finally, no two blocks of size three have fewer than $t-2$ blocks of size two between them.

Consider the position of a 3-block in the first frame, F_1. If there are two 3-blocks in F_1, they appear as the first and last blocks in F_1, but then the distance from x_0

to x_{t-1} is $2t$, a contradiction. Since there is exactly one 3-block, B_k, in F_1, suppose $k < t$. Then the distance from x_0 to x_{t-1} is $2t$. Hence, B_t is the 3-block in F_1. By symmetry, there must be $t-1$ 2-blocks between the 3-block in $F_{(2t-2)t+1}$ and x_0.

Let $B_{k_1}, B_{k_2}, \dots, B_{k_{2t}}$ be the 3-blocks in X with $k_1 < k_2 < \cdots < k_{2t}$. By the position of the 3-block in F_1, we have $k_1 = t$. By the position of the 3-block in $F_{(2t-2)t+1}$, we have $k_{2t} = (2t-2)t+1$. Since 3-blocks must be separated by at least $t-1$ 2-blocks, $k_{j+1} - k_j \geq t-1$ but since $k_{2t} = (2t-1)(t-1) + k_1$ we must have equality: $k_{j+1} - k_j = t-1$. Assuming X is an r-clique, it is uniquely defined by these properties. Indeed all vertices of this set are adjacent. □

11.3.2 Three Generators

Theorem 11.3. *Let* $t \geq 1$ *and set* $n = 9t^2 - 3t + 1, r = 3t^2 - 2t + 1$. *Then,* $\overline{C}(\mathbb{Z}_n, \{1, 3t-1, 3t\})$ *is r-primitive.*

Proof. Let $G = \overline{C}(\mathbb{Z}_n, \{1, 3t-1, 3t\})$. Observe that G is vertex-transitive and there are automorphisms mapping $\{0, 3t-1\}$ to $\{0,1\}$ or $\{0, 3t\}$ to $\{0,1\}$. Thus, we only need to verify that G has no r-clique and $G + \{0,1\}$ has a unique r-clique.

We prove that G is r-primitive in three steps. First, we show that there is no r-clique in G in Claim 11.11 using discharging. Second, assuming there are no 2-blocks in an r-clique of $G + \{0,1\}$, we prove in Claim 11.12 that there is a unique such clique. This proof uses a counting method similar to the proof of Claim 11.7. Finally, we show that any r-clique in $G + \{0,1\}$ cannot contain any 2-blocks. This step is broken into Claims 11.13 and 11.14, both of which slightly modify the discharging method from Claim 11.11 to handle the 1-block. Claim 11.14 requires a detailed case analysis.

We use several figures to aid the proof. Figure 11.2 shows examples of common features from these figures.

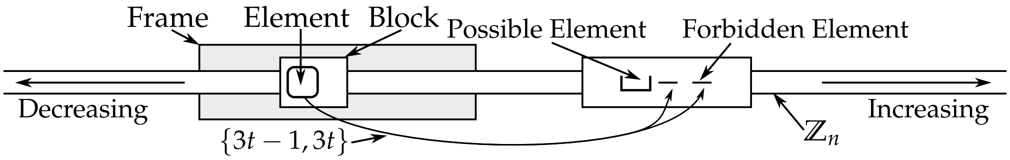


Figure 11.2: Key to later figures

We begin by showing some basic observations which are used frequently in the rest of the proof. These observations focus on interactions among blocks that are forced by the generators $3t-1$ and $3t$. In the observations below, we define functions φ_s and ψ_s which map s-blocks of X to other blocks of X. Always, φ_s maps blocks *forward* ($\varphi_s(B_k)$ has higher index than B_k) while ψ_s maps blocks *backward* ($\psi_s(B_k)$ has lower index than B_k).

It is intuitive that a maximum size clique uses as many small blocks as possible, to increase the density of the clique within G. However, Observation 11.8 shows that every 2-block induces a block of size at least five *in both directions*.

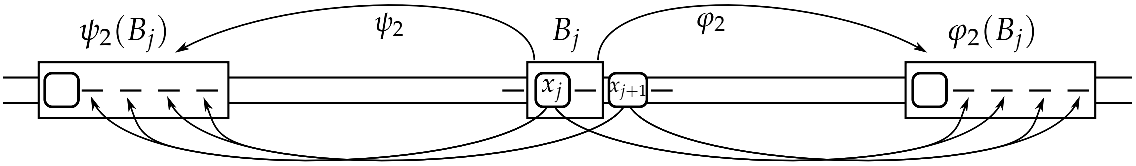


Figure 11.3: Observation 11.8 and a 2-block B_j.

Observation 11.8 (2-blocks)**.** Let B_j be a 2-block, so $x_{j+1} = x_j + 2$. The elements x_j and x_{j+1} along with generators $3t-1$ and $3t$ guarantee that the sets $\{x_j + 3t - 1, x_j + 3t, x_j + 3t + 1, x_j + 3t + 2\}$ and $\{x_j - 3t, x_j - 3t + 1, x_j - 3t + 2, x_j - 3t + 3\}$ do not intersect X. Since these sets contain consecutive elements, each set is contained within a single block of X. We will use $\varphi_2(B_j)$ to denote the block containing $x_j + 3t$

and $\psi_2(B_j)$ to denote the block containing $x_j - 3t$. Both $\varphi_2(B_j)$ and $\psi_2(B_j)$ have size at least five.

If in fact multiple 2-blocks induce the same big block, Observation 11.9 implies the big block has even larger size.

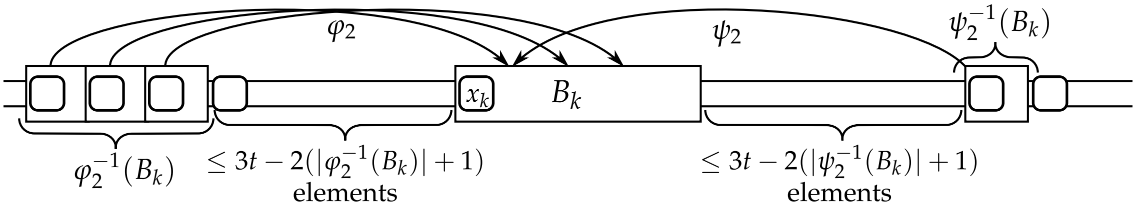


Figure 11.4: Observation 11.9 and a block B_k.

Observation 11.9 (Big blocks)**.** Let B_k be a block of size at least five. The set $\varphi_2^{-1}(B_k)$ is the set of 2-blocks B_j so that $\varphi_2(B_j) = B_k$. Similarly, $\psi_2^{-1}(B_k)$ is the set of 2-blocks B_j so that $\psi_2(B_j) = B_k$. Note that when $s = |\varphi_2^{-1}(B_k)|$, there are at least $s+1$ elements of X (s from the 2-blocks in $\varphi_2^{-1}(B_k)$ and one following the last 2-block in $\varphi_2^{-1}(B_k)$) which block $2(s+1)$ elements from containment in X using the generators $3t-1$ and $3t$. Therefore,

$$|B_k| \geq 2|\varphi_2^{-1}(B_k)| + 3, \quad \text{and} \quad |B_k| \geq 2|\psi_2^{-1}(B_k)| + 3.$$

Further, there are at most $3t - 2(|\varphi_2^{-1}(B_k)| + 1)$ elements between B_k and the last block of $\varphi_2^{-1}(B_k)$. Similarly, there are at most $3t - 2(|\psi_2^{-1}(B_k)| + 1)$ elements between B_k and the first block of $\psi_2^{-1}(B_k)$.

Observation 11.10 (4-blocks)**.** Let B_j be a 4-block, so $x_{j+1} = x_j + 4$. The elements $\{x_j + 3t - 1, x_j + 3t, x_j + 3t + 3, x_j + 3t + 4\}$ are not contained in X, so $X \cap \{x_j + 3t - 1, \ldots, x_j + 3t + 4\} \subseteq \{x_j + 3t + 1, x_j + 3t + 2\}$. In G, no two elements of X are consecutive elements of $\mathbb{Z}_n$, so there is at most one element in this range. If

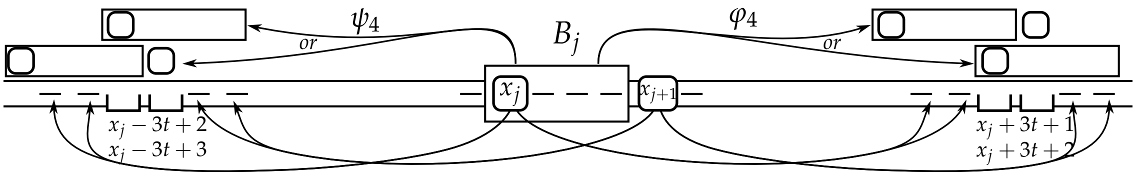


Figure 11.5: Observation 11.10 and a 4-block B_j.

there is no element of X in $\{x_j + 3t + 1, x_j + 3t + 2\}$, then there is a block of size at least seven that contains $x_j + 3t + 1$. Otherwise, there is a single element in $X \cap \{x_j + 3t + 1, x_j + 3t + 2\}$ and one of the adjacent blocks has size at least four. We use $\varphi_4(B_j)$ to denote one of these blocks of size at least four. By symmetry, we use $\psi_4(B_j)$ to denote a block of size at least four that contains or is adjacent to the block containing $x_j - 3t + 2$. In $G + \{0, 1\}$, the only elements of X that can be consecutive are 0 and 1, let $B_0 = \{0\}$ denote the first block of X. Thus, let $\varphi_4(B_j) = B_0$ if $x_j + 3t + 1 = 0$ and $\psi_4(B_j) = B_0$ if $x_j - 3t + 2 = 0$.

We now use a two-stage discharging method to prove that there is no r-clique X in G. In Stage 1, we assign charge to the blocks of X and discharge so that all blocks have non-negative charge. In Stage 2, we assign charge to the frames of X using the new charges on the blocks and then discharge among the frames.

$$\begin{tikzcd}
\textbf{Stage 1: } \textit{Blocks} & \mu(B_j) \arrow[r, "\text{discharge}"] & \mu^*(B_j) \arrow[d, "\text{defines}"] & \\
\textbf{Stage 2: } \textit{Frames} & & \nu^*(F_j) \arrow[r, "\text{discharge}"] & \nu'(F_j)
\end{tikzcd}$$

Figure 11.6: The two-stage discharging method.

We will use this framework three times, in Claims 11.11, 11.13, and 11.14, but we use a different set of rules for Stage 1 each time. Stage 2 will always use the same discharging rule.

Claim 11.11. *$\omega(G) < r$.*

Proof of Claim 11.11. Suppose X is an r-clique in G.

Let μ be a charge function on the blocks of X defined by $\mu(B_j) = |B_j| - 3$. All 2-blocks have charge -1, 3-blocks have charge 0, and all other blocks have positive charge. Moreover, the total charge on all blocks is

$$\sum_{j=0}^{r-1} \mu(B_j) = n - 3r = 3t - 2.$$

We shall discharge among the blocks to form a new charge function μ^*.

Stage 1α: Discharge by shifting one charge from $\varphi_2(B_j)$ to B_j for every 2-block B_j.

After Stage 1α, $\mu^*(B_j) = 0$ when $|B_j| \in \{2, 3\}$, $\mu^*(B_j) = 1$ when $|B_j| = 4$, and

$$\mu^*(B_j) = |B_j| - 3 - |\varphi_2^{-1}(B_j)| \geq |\varphi_2^{-1}(B_j)|$$

when $|B_j| \geq 5$. Note that if $|\varphi_2^{-1}(B_j)| = 0$ for a block B_j of size at least five, then $\mu^*(B_j) \geq 2$.

Now, μ^* is a non-negative function and $\sum_{j=0}^{r-1} \mu^*(B_j) = 3t - 2$.

For every frame F_j, define $\nu^*(F_j)$ as $\nu^*(F_j) = \sum_{B_{j+i} \in F_j} \mu^*(B_{j+i})$. Since every block is contained in exactly t frames, the total charge on all frames is

$$\sum_{j=0}^{r-1} \nu^*(F_j) = t \sum_{j=0}^{r-1} \mu^*(B_j) = t(3t - 2) = r - 1.$$

There must exist a frame with $\nu^*(F_j) = 0$, and hence contains only 2- and 3-blocks. If this frame contained only blocks of length three and at most one block of length two, then $\sigma(F_j) \in \{3t - 1, 3t\}$, contradicting that X is a clique. Thus, any frame with $\nu^*(F_j) = 0$ must contain at least two 2-blocks where all blocks between are 3-blocks.

For each pair $B_k, B_{k'}$ of 2-blocks that are separated only by 3-blocks, define $L_{k,k'}$ to be the set of frames containing both B_k and $B_{k'}$, and $R_{k,k'}$ to be the set of frames containing both $\varphi_2(B_k)$ and $\varphi_2(B_{k'})$. If $\varphi_2(B_k) = \varphi_2(B_{k'})$, then $|R_{k,k'}| = t \geq |L_{k,k'}|$. Otherwise, there are fewer elements between $\varphi_2(B_k)$ and $\varphi_2(B_{k'})$ than between B_k and $B_{k'}$, and every block between $\varphi_2(B_k)$ and $\varphi_2(B_{k'})$ has size at least three (a 2-block B_j between $\varphi_2(B_k)$ and $\varphi_2(B_{k'})$ would induce a large block $\psi_2(B_j)$ between B_k and B'_k). Hence, there are at least as many blocks between B_k and B'_k as there are between $\varphi_2(B_k)$ and $\varphi_2(B_{k'})$ and so $|L_{k,k'}| \leq |R_{k,k'}|$. Let $f_{k,k'} : L_{k,k'} \to R_{k,k'}$ be any injection where $f_{k,k'}(F_j) = F_j$ for all $F_j \in L_{k,k'} \cap R_{k,k'}$.

Using these injections, we discharge among the frames to form a new charge function ν'.

Stage 2: For every frame F_j and every pair $B_k, B_{k'}$ of 2-blocks in F_j separated by only 3-blocks, F_j pulls one charge from $f_{k,k'}(F_j)$.

Since every frame F_j with $\nu^*(F_j) = 0$ has at least one such pair $B_k, B_{k'}$ and does not contain $\varphi_2(B_i)$ for any 2-block B_i, F_j pulls at least one charge but does not have any charge removed. Thus, $\nu'(F_j) \geq 1$.

We will show that frames F_j with $\nu^*(F_j) \geq 1$ have strictly less than $\nu^*(F_j)$ charge pulled during the second stage. Let $\{(B_{k_i}, B_{k'_i}; F_{j_i}) : i \in \{1, \dots, \ell\}\}$ be the set of pairs $B_{k_i}, B_{k'_i}$ of 2-blocks and a common frame F_{j_i} where $f_{k_i,k'_i}(F_{j_i}) = F_j$. Since each map f_{k_i,k'_i} is an injection, the blocks B_{k_i} are distinct for all $i \in \{1, \dots, \ell\}$, and exactly ℓ charge was pulled from F_j. While $B_{k'_i}$ and $B_{k_{i+1}}$ may be the same block, $B_{k_1}, \dots, B_{k_\ell}, B_{k'_\ell}$ are $\ell + 1$ distinct 2-blocks. Every block B_{k_i} has $\varphi_2(B_{k_i}) \in F_j$ and $\varphi_2(B_{k'_\ell}) \in F_j$. Thus, $\nu^*(F_j) \geq \sum_{B_i \in F_j} |\varphi_2^{-1}(B_i)| \geq \ell + 1$ which implies $\nu'(F_j) \geq 1$.

Therefore, $\nu'(F_j) \geq 1$ for all frames F_j, and $r - 1 = \sum_{j=0}^{r-1} \nu'(F_j) \geq r$, a contradiction. Hence, there is no clique of size r in G, proving Claim 11.11. □

For the remaining claims, we assume X is an r-clique in $G + \{0, 1\}$ where X contains both 0 and 1. Then, B_0 is the block containing exactly $\{0\}$, and all other blocks from X have size at least two. Since 0 and 1 are in X, the sets $\{3t-1, 3t, 3t+1\}$ and $\{-3t-1, -3t, -3t+1\}$ of consecutive elements do not intersect X. Thus, there are two blocks B_{k_1} and B_{k_2} so that $\{3t-1, 3t, 3t+1\} \subset B_{k_1}$ and $\{-3t-1, -3t, -3t+1\} \subset B_{k_2}$. When B_{k_1} and B_{k_2} are 4-blocks, then $B_0 = \psi_4(B_{k_1}) = \varphi_4(B_{k_2})$ as in Observation 11.10.

With the assumption that there are no 2-blocks in X, uniqueness follows through an enumerative proof similar to Claim 11.7, given as Claim 11.12. After this claim, Claims 11.13 and 11.14 show that X has no 2-blocks, completing the proof.

Claim 11.12. *There is a unique r-clique in $G + \{0, 1\}$ with no 2-blocks.*

Proof of Claim 11.12. Consider the frame family $\mathcal{F} = \{F_{jt+1} : j \in \{0, \dots, 3t-2\}\}$ of $3t-1$ disjoint frames. Note that the block B_0 is not contained in any of these frames. Since there are no 2-blocks, $\sigma(F_{jt+1}) \geq 3t$, but $\sigma(F_{jt+1}) \neq 3t$ so $\sigma(F_{jt+1}) \geq 3t+1$. Thus,

$$n - 1 = \sum_{F_{jt+1} \in \mathcal{F}} \sigma(F_{jt+1}) \geq (3t-1)(3t+1) = n - 3.$$

From this inequality we have $\sigma(F_{jt+1}) = 3t+1$ for all frames except either one frame F_k with $\sigma(F_k) = 3t+3$ or two frames $F_k, F_{k'}$ with $\sigma(F_k) = \sigma(F_{k'}) = 3t+2$.

Suppose there is a frame F_k with $\sigma(F_k) = 3t+3$. Since $x_{k+t} = x_k + 3t + 3$, the elements

$$x_{k+t} - 3t = x_k + 3, \quad x_{k+t} - (3t-1) = x_k + 4,$$

$$x_k + 3t - 1 = x_{k+t} - 4, \quad \text{and} \quad x_k + 3t = x_{k+t} - 3,$$

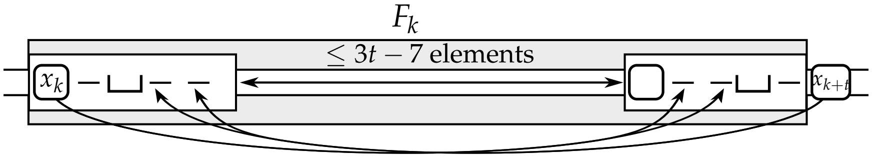


Figure 11.7: Claim 11.12, $\sigma(F_k) = 3t+3$.

are not contained in X. Since we have no 2-blocks, the elements x_k+2 and $x_{k+t}-2$ are not in X. Thus, there are two blocks of size at least five in F_k. This means there are $t-2$ blocks for the remaining $3t-7$ elements, but $t-2$ blocks of size at least three cover at least $3t-6$ elements. Hence, no frame has $\sigma(F_k) = 3t+3$.

Suppose we have exactly two frames $F_k, F_{k'} \in \mathcal{F}$ with $\sigma(F_k) = \sigma(F_{k'}) = 3t+2$. If a frame F_j contains a block of size at least six, then $\sigma(F_j) \geq 3t+3$, so F_k and $F_{k'}$ each contain either one 5-block or two 4-blocks. However, if the first or last block (denoted by B_j) of F_k (or $F_{k'}$) has size three, then $\sigma(F_k \setminus \{B_j\}) = 3t-1$, a contradiction. Thus, the first and last blocks of F_k and $F_{k'}$ are not 3-blocks and hence are both 4-blocks. Therefore, there are exactly two frames in $\mathcal{F}$ containing exactly two 4-blocks and the rest contain exactly one 4-block, for a total of $3t$ 4-blocks in X.

Let $\ell_1, \ell_2, \ldots, \ell_{3t}$ be the indices of the 4-blocks. Since each frame F_i has at least one 4-block, $\ell_j \leq \ell_{j-1} + t$. Also, if a frame F_i has exactly two 4-blocks, then the blocks appear as the first and last blocks in F_j, giving $\ell_j \geq \ell_{j-1} + t - 1$.

Consider the position of B_{ℓ_1}. If B_{ℓ_1} is strictly between B_0 and B_{k_1}, then the frame F_1 contains two 4-blocks B_{ℓ_1} and B_{k_1}, and so $B_{\ell_1} = B_1$ and $B_{k_1} = B_t$. But, there are $3t-3$ elements between B_0 and B_{k_1}, but at least $3t-2$ elements between B_0 and B_t. Therefore, $B_{\ell_1} = B_{k_1}$ and there are $t-1$ 3-blocks between B_0 and B_{ℓ_1}, so $\ell_1 = t-1$. Similarly, $B_{\ell_{3t}} = B_{k_2}$ and there are $t-1$ 3-blocks between $B_{\ell_{3t}}$ and B_0, so $\ell_{3t} = (r-1) - (t-1) = 3t^2 - 3t + 1$.

There is exactly one solution to the constraints $\ell_j \in \{\ell_{j-1}+t-1, \ell_{j-1}+t\}$ and $\ell_{3t}-\ell_1 = 3t^2-2t+1 = (3t-1)(t-1)$ given by $\ell_j = \ell_{j-1}+t-1$. This uniquely describes X as a clique in $G+\{0,1\}$. $\square$

We now aim to show that there are no 2-blocks in an r-clique X of G. This property can be quickly checked computationally for $t \leq 4$, so we now assume that $t \geq 5$.

The problem with applying the discharging method from Claim 11.11 is that B_0 starts with charge $\mu(B_0) = -2$ and there is no clear place from which to pull charge to make $\mu^*(B_0)$ positive. We define three values, a, b, and c, which quantify the *excess charge* from Stage 1α which can be redirected to B_0 while still guaranteeing that all frames end with positive charge. In Claim 11.13, we assume $a+b+c \geq 3$ and place all of this excess charge on B_0 in Stage 1β, giving $\mu^*(B_0) \geq 1$; an identical Stage 2 discharging leads to positive charge on all frames. In Claim 11.14, Stage 1γ pulls charge from B_{k_1} and B_{k_2} to result in $\mu^*(B_0) = 0$ and possibly $\mu^*(B_{k_1}) = 0$ or $\mu^*(B_{k_2}) = 0$. After Stage 1γ and Stage 2, there may be some frames with ν'-charge zero, but they must contain B_0, B_{k_1}, or B_{k_2}. By carefully analyzing this situation, we find a contradiction in that either X is not a clique or $a+b+c \geq 3$.

We now define the quantities a, b, and c.

If a block B_j has size at least five and $\varphi_2^{-1}(B_j)$ is empty, then no charge is removed from B_j in Stage 1α. If charge is pulled from frames containing B_j in Stage 2, there are other blocks that supply the charge required to stay positive. Therefore, we define a to be the excess μ-charge that can be removed and maintain positive μ^*-charge:

$$a = \sum_{B_j \in \mathcal{A}} \left[|B_j| - 4\right], \text{ where } \mathcal{A} \text{ is the set of blocks } B_j \text{ with } |B_j| \geq 5 \text{ and } \varphi_2^{-1}(B_j) = \emptyset.$$

If a block B_j has size at least five and $\varphi_2^{-1}(B_j)$ is not empty, charge is pulled from B_j in Stage 1α. However, if $|B_j| > 2|\varphi_2^{-1}(B_j)| + 3$, there is more charge left after Stage 1α than is required in Stage 2 to maintain a positive charge on frames containing B_j. We define b to be the excess charge left in this situation:

$$b = \sum_{B_j \in \mathcal{B}} \left[|B_j| - (2|\varphi_2^{-1}(B_j)| + 3) \right],$$

where $\mathcal{B}$ is the set of blocks B_j with $|B_j| \geq 5$ and $\varphi_2^{-1}(B_j) \neq \emptyset$.

If there is a frame F_j with three blocks $B_{\ell_0}, B_{\ell_1}, B_{\ell_2}$ where $|B_{\ell_i}| \geq 4$ for all $i \in \{0, 1, 2\}$ and $\varphi_2^{-1}(B_{\ell_1}) = \emptyset$, then let $c = 1$; otherwise $c = 0$. Since every frame containing B_{ℓ_1} also contains B_{ℓ_0} or B_{ℓ_2}, these frames are guaranteed a positive ν'-charge from B_{ℓ_0} or B_{ℓ_2}, so the single charge on B_{ℓ_1} that was not pulled from previous rules is free to pass to B_0.

Claim 11.13. *Suppose X is a set in $G + \{0, 1\}$ with $|X| = r$. If $a + b + c \geq 3$, then X is not a clique.*

Proof of Claim 11.13. We proceed by contradiction, assuming that $a + b + c \geq 3$ and X is an r-clique. We shall modify the two-stage discharging from Claim 11.11 with a more complicated discharging rule to handle B_0 so that the result is the same contradiction: that all r frames have positive charge, but the amount of charge over all the frames is $r - 1$.

Let μ be the charge function on the blocks of X defined by $\mu(B_j) = |B_j| - 3$. We discharge using Stage 1β to form the charge function μ^*.

Stage 1β: There are four discharging rules:

1. If $|B_k| = 2$, B_k pulls one charge from $\varphi_2(B_k)$.

2. B_0 pulls $|B_k| - 4$ charge from every block B_k with $|B_k| \geq 5$ and $\varphi_2^{-1}(B_k) = \emptyset$. (The total charge pulled by B_0 in this rule is a.)
3. B_0 pulls $|B_k| - (2|\varphi_2^{-1}(B_k)| + 3)$ charge from every block B_k with $|B_k| \geq 5$ and $\varphi_2^{-1}(B_k) \neq \emptyset$. (The total charge pulled by B_0 in this rule is b.)
4. If there is a frame F_j with three blocks $B_{\ell_0}, B_{\ell_1}, B_{\ell_2}$ where $|B_{\ell_i}| \geq 4$ for all $i \in \{0,1,2\}$ and $\varphi_2^{-1}(B_{\ell_1}) = \emptyset$, then B_0 pulls one charge from B_{ℓ_1}. (The amount of charge pulled by B_0 in this rule is c.)

Since $a + b + c \geq 3$, B_0 pulls at least 3 charge, so $\mu^*(B_0) \geq 1$. Blocks of size two and three have μ^*-charge zero. If a block B_k has size four or has size at least five and $\varphi_2^{-1}(B_k) = \emptyset$, then $\mu^*(B_k) = 1$ except B_{ℓ_1} where $\mu^*(B_{\ell_1}) = 0$. Similarly, a block B_k of size at least five with $\varphi_2^{-1}(B_k) \neq \emptyset$ has charge $\mu^*(B_k) = |\varphi_2^{-1}(B_k)|$.

For every frame F_j, define $\nu^*(F_j) = \sum_{B_{j+i} \in F_j} \mu^*(B_{j+i})$. Note that if the charge $\nu^*(F_j)$ is zero, every block in F_j has zero charge since $\mu^*(B_k) \geq 0$ for all blocks.

Stage 2: For every frame F_j and every pair $B_k, B_{k'}$ of 2-blocks in F_j separated by only 3-blocks, F_j pulls one charge from $f_{k,k'}(F_j)$.

If $\nu^*(F_j) = 0$, then F_j contains only blocks B_k with $\mu^*(B_k) = 0$. These blocks are 2-blocks, 3-blocks, and B_{ℓ_1}. However, any frame which contains B_{ℓ_1} also contains B_{ℓ_0} or B_{ℓ_2} which have positive charge. Thus, frames F_j with $\nu^*(F_j) = 0$ contain only 2- and 3-blocks. Since $\sigma(F_j) \notin \{3t, 3t-1\}$, F_j must contain at least two 2-blocks $B_k, B_{k'}$, so F_j pulls at least one charge in the second stage and loses no charge, so $\nu'(F_j) \geq 1$.

If $\nu^*(F_j) \geq 1$, the amount of charge pulled from F_j in Stage 2 is the number of 2-block pairs $B_k, B_{k'}$ separated by 3-blocks so that $\varphi_2(B_k), \varphi_2(B_{k'}) \in F_j$. Observe $\mu^*(B_i) = |\varphi_2^{-1}(B_i)|$ for all blocks B_i with $\varphi_2^{-1}(B_i) \neq \emptyset$, so $\nu^*(F_j) = \sum_{B_i \in F_j} \mu^*(B_i) \geq \sum_{B_i \in F_j} |\varphi_2^{-1}(B_i)|$. If there are ℓ pairs $B_k, B_{k'}$ that pull one charge from F_j in Stage 2, then there are at least $\ell+1$ 2-blocks in $\cup_{B_i\in F_j}\varphi_2^{-1}(B_i)$, and $\nu^*(F_j)\geq \ell+1$.

Therefore, $\nu'(F_j)\geq 1$ for all $j\in\{0,\dots,r-1\}$, but since

$$r\leq \sum_{j=0}^{r-1}\nu'(F_j)=\sum_{j=0}^{r-1}\nu^*(F_j)=t\sum_{j=0}^{r-1}\mu^*(B_j)=t\sum_{j=0}^{r-1}\mu(B_j)=t(n-3r)=r-1,$$

we have a contradiction, and so X is not a clique. □

Claim 11.14. *If X is an r-clique in $G+\{0,1\}$ that contains a 2-block, then $a+b+c\geq 3$.*

Proof of Claim 11.14. We shall repeat the two-stage discharging from Claim 11.11 with a simpler rule for discharging to B_0 than in Claim 11.13. After this discharging is complete, we will investigate the configuration of blocks surrounding one of the 2-blocks and show that the sum $a+b+c$ has value at least three.

Let μ be the charge function on the blocks of X defined by $\mu(B_j)=|B_j|-3$. We use Stage 1γ to discharge among the blocks and form a charge function μ^*.

Stage 1γ: We have two discharging rules:

1. If $|B_j|=2$, B_j pulls one charge from $\varphi_2(B_j)$.
2. B_0 pulls one charge from B_{k_1} and one charge from B_{k_2}.

After the first rule within Stage 1γ there is at least one charge on all blocks of size at least four. Thus, removing one more charge from each of B_{k_1} and B_{k_2} in the second rule of Stage 1γ maintains that $\mu^*(B_{k_1})$ and $\mu^*(B_{k_2})$ are non-negative. Since B_0 receives two charge and every 2-block receives one charge, $\mu^*(B_j)$ is non-negative after Stage 1γ for all blocks B_j.

Define the charge function $\nu^*(F_j)=\sum_{B_i\in F_j}\mu^*(B_i)$.

Stage 2: For every frame F_j and every pair $B_k, B_{k'}$ of 2-blocks in F_j separated by only 3-blocks, F_j pulls one charge from $f_{k,k'}(F_j)$.

Again, $\sum_{j=0}^{r-1} \nu'(F_j) = r-1$. Also, $\nu'(F_j) > 0$ whenever F_j contains a block of order at least four that is not B_{k_1} or B_{k_2}, or F_j contains two 2-blocks separated only by 3-blocks. Since one charge was removed from B_{k_1} and B_{k_2} in Stage 1γ, the frames containing B_{k_1} or B_{k_2} are no longer guaranteed to have positive charge, but still have non-negative charge. In order to complete the proof of Claim 11.14, we must more closely analyze the charge function ν'.

Definition 11.15 (Pull sets)**.** A *pull set* is a set of blocks, $\mathcal{P} = \{B_{i_1}, \dots, B_{i_p}\}$, where $|B_{i_j}| \geq 5$ for all $j \in \{1, \dots, p\}$ and all blocks between B_{i_j} and $B_{i_{j+1}}$ are 3-blocks. Let $\varphi_2^{-1}(\mathcal{P}) = \cup_{B_i \in \mathcal{P}} \varphi_2^{-1}(B_i)$. A pull set $\mathcal{P}$ is *perfect* if all blocks $B_i \in \mathcal{P}$ have $|B_i| = 2|\varphi_2^{-1}(B_i)| + 3$. Otherwise, a pull set $\mathcal{P}$ contains a block $B_i \in \mathcal{P}$ with $|B_i| \geq 2|\varphi_2^{-1}(B_i)| + 4$ and $\mathcal{P}$ is *imperfect*. Given a pull set $\mathcal{P}$, the *defect* of $\mathcal{P}$ is $\delta(\mathcal{P}) = \sum_{B_i \in \mathcal{P}} \left[\mu^*(B_i) - |\varphi_2^{-1}(B_i)|\right] - 1$.

The defect $\delta(\mathcal{P})$ measures the amount of excess charge (more than one charge) the pull set $\mathcal{P}$ contributes to the ν'-charge of any frame containing $\mathcal{P}$. Note that pull sets $\mathcal{P}$ with $B_{k_1}, B_{k_2} \notin \mathcal{P}$ have defect $\delta(\mathcal{P}) \geq 0$, with equality if and only if $\mathcal{P}$ is perfect. Perfect pull sets $\mathcal{P}$ containing B_{k_1} or B_{k_2} have defect $\delta(\mathcal{P}) = -1$. For a block $B_i \in \mathcal{P}$, if $d \leq \mu^*(B_i) - |\varphi_2^{-1}(B_i)|$ then we say B_i *contributes* d to the defect of $\mathcal{P}$.

Consider a pull set $\mathcal{P} = \{B_{i_1}, \dots, B_{i_p}\}$. Since there are at most $3t-4$ elements between $\varphi_2^{-1}(B_{i_p})$ and B_{i_p} and all blocks from B_{i_1} to B_{i_p} have order at least three, there exists a frame that contains all blocks of $\mathcal{P}$. Therefore, every pull set is contained within *some* frame.

If B_i is a block with $|B_i| \geq 5$, then $\mathcal{P} = \{B_i\}$ is a (not necessarily maximal) pull set, and $\{B_i\}$ is a subset of each frame containing B_i. For every frame F_j and block $B_i \in F_j$ with $|B_i| \geq 5$ there is a unique maximal pull set $\mathcal{P} \subseteq F_j$ containing B_i. Thus, if there are multiple maximal pull sets within a frame F_j, then they are disjoint.

Observation 11.16. Let X be an r-clique and ν' be the charge function on frames of X after Stage 1γ and Stage 2. Then, for a frame F_j, $\nu'(F_j)$ is at least the sum of

1. the number of distinct pairs $B_k, B_{k'}$ of 2-blocks in F_j separated only by 3-blocks,
2. the number of 4-blocks in F_j not equal to B_{k_1}, B_{k_2},
3. $1 + \delta(\mathcal{P})$ for every maximal pull set $\mathcal{P} \subseteq F_j$.

In Claim 11.14.4, we prove there exists a special block B_* in a frame F_z with $\nu'(F_z) = 0$. The proof of Claim 11.14.4 reduces to three special cases which are handled in Claims 11.14.1-11.14.3.

Recall $\sum_{j=0}^{r-1} \nu'(F_j) = r - 1$. Let Z be the number of frames F with $\nu'(F) = 0$. Then,

$$\sum_{j:\nu'(F_j)>0} \left[\nu'(F_j) - 1\right] = \sum_{j=0}^{r-1} \left[\nu'(F_j) - 1\right] + Z = (r-1) - r + Z = Z - 1.$$

Therefore, if there are at most $t+1$ frames with ν'-charge zero ($\nu'(F_j) = 0$), then the sum $\sum_{j:\nu'(F_j)>0}[\nu'(F_j) - 1]$ is bounded above by t. The proof of Claim 11.14.4 frequently reduces to a contradiction with this bound. Claims 11.14.1-11.14.3 provide some situations which guarantee this sum has value at least $t+1$.

Claim 11.14.1. *Let $\mathcal{P}$ be a pull set containing a block B_j. If $|\varphi_2^{-1}(\mathcal{P})| \geq 2$ and $x_{k_1} + 6t^2 \leq x_j \leq x_{k_2}$, then there is a set $\mathcal{H}$ of frames with $\sum_{F_j \in \mathcal{H}}(\nu'(F_j) - 1) \geq t + 1$.*

Proof of Claim 11.14.1. Starting with $\mathcal{P}^{(0)} = \mathcal{P}$, we construct a sequence $\mathcal{P}^{(0)}$, $\mathcal{P}^{(1)}$, $\ldots$, $\mathcal{P}^{(\ell)}$ of pull sets with $\ell \leq \lceil \frac{t+1}{2} \rceil + 1$. We build $\mathcal{P}^{(k)}$ by following the map ψ_2 from $\varphi_2^{-1}(\mathcal{P}^{(k-1)})$. This process will continue until one of the sets is not a pull set, one of the sets is an imperfect pull set, or we reach $\lceil \frac{t+1}{2} \rceil$ pull sets. In either case, we find a set $\mathcal{H}$ of frames that satisfies the claim.

We initialize $\mathcal{P}^{(0)}$ to be $\mathcal{P}$, which contains B_j. Note that it is possible that $B_j = B_{k_2}$, but otherwise B_j precedes B_{k_2}. There will be at most $6t$ elements covered by the blocks starting at $\mathcal{P}^{(k)}$ to the blocks preceding $\mathcal{P}^{(k-1)}$. Note that since $x_j - x_{k_1} \geq 6t^2$, $\mathcal{P}^{(k)}$ will not contain B_{k_1} or B_{k_2} for any $k \in \{1, \ldots, \lceil \frac{t+2}{2} \rceil\}$.

Let $k \geq 1$ be so that $\mathcal{P}^{(k-1)}$ is a perfect pull set with $|\varphi_2^{-1}(\mathcal{P}^{(k-1)})| \geq 2$. For every block $B_i \in \mathcal{P}^{(k-1)}$, let B_ℓ be a 2-block in $\varphi_2^{-1}(B_i)$ and place $\psi_2(B_\ell)$ in $\mathcal{P}^{(k)}$. Then, place any block of size at least five that is positioned between to blocks of $\mathcal{P}^{(k)}$ into $\mathcal{P}^{(k)}$.

If $\mathcal{P}^{(k)}$ is always perfect for all $k \leq \lceil \frac{t+1}{2} \rceil$, then we have pull sets $\mathcal{P}^{(0)}, \ldots, \mathcal{P}^{(k)}$ and frames $F_{j_0}, F_{j'_0}, \ldots, F_{j_{k-1}}, F_{j'_{k-1}}$, where $k = \lceil \frac{t+1}{2} \rceil$. Thus, let $\mathcal{H} = \{F_{j_\ell}, F_{j'_\ell} : \ell \in \{1, \ldots, k\}\}$ and $\sum_{F \in \mathcal{H}} [\nu'(F) - 1] \geq t + 1$, proving the claim. It remains to show that such a set $\mathcal{H}$ exists if some $\mathcal{P}^{(k)}$ is imperfect.

If $\mathcal{P}^{(k)}$ is a perfect pull set with $|\varphi_2^{-1}(\mathcal{P}^{(k)})| \geq 2$, then let F_{j_k} be the frame that starts at the last block of $\mathcal{P}^{(k)}$ and $F_{j'_k}$ be the frame that ends at the first block of $\mathcal{P}^{(k)}$. We claim that F_{j_k} and $F_{j'_k}$ have ν'-charge at least two. There are at most $3t - 4$ elements between the last block in $\mathcal{P}^{(k)}$ and the last 2-block in $\psi_2^{-1}(\mathcal{P}^{(k)})$. If there is at most one 2-block in F_{j_k}, then $\sigma(F_{j_k}) \geq 2 + 3(t-2) + 5 = 3t + 3$ and F_{j_k} contains all 2-blocks in $\psi_2^{-1}(\mathcal{P}^{(k)})$, a contradiction. Therefore, the frame F_{j_k} contains at least two 2-blocks. If those 2-blocks are separated by three blocks, they pull at least one charge in Stage 2. If those 2-blocks are not separated by three blocks, then either they are separated by a 4-block (which contributes at least one charge) or a second maximal pull set (which contributes at least one charge). Thus, $\nu'(F_{j_k}) \geq 2$. By a symmetric argument, $F_{j'_k}$ contains two 2-blocks and has $\nu'(F_{j'_k}) \geq 2$. Figure 11.9 shows how the frames F_{j_k} and $F_{j'_k}$ are placed among the pull sets $\mathcal{P}^{(k-1)}$ and $\mathcal{P}^{(k)}$.

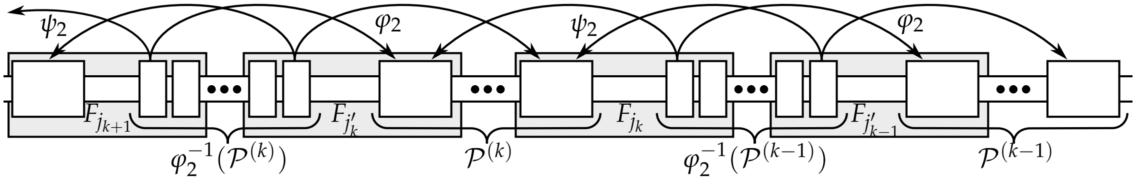


Figure 11.8: Claim 11.14.1, building $\mathcal{P}^{(k)}$ and frames $F_{j_k}, F_{j'_k}$.

If $\mathcal{P}^{(k)}$ is not a perfect pull set or $|\varphi_2^{-1}(\mathcal{P}^{(k)})| < 2$, either $\mathcal{P}^{(k)}$ is not a pull set or $\mathcal{P}^{(k)}$ is an imperfect pull set.

Case 1: $\mathcal{P}^{(k)}$ is not a pull set. In this case, there is a non-3-block B_j not in $\mathcal{P}^{(k)}$ that is between two blocks B_{ℓ_1}, B_{ℓ_2} of $\mathcal{P}^{(k)}$. If $|B_j| \geq 5$, then B_j would be added to $\mathcal{P}^{(k)}$. Therefore, $|B_j| \in \{2, 4\}$.

Case 1.i: $|B_j| = 4$. Every frame containing B_j also contains either B_{ℓ_1} or B_{ℓ_2}. Therefore, these t frames contain a 4-block and at least one pull set with non-negative defect so they have ν'-charge at least two. The frame starting at B_{ℓ_1} also contains B_j and B_{ℓ_2}, so this frame has two disjoint maximal pull sets and a 4-block and has ν'-charge at least three. Therefore, if $\mathcal{H}$ is the family of frames containing B_j, $\sum_{F\in\mathcal{H}}[\nu'(F) - 1] \geq t + 1$.

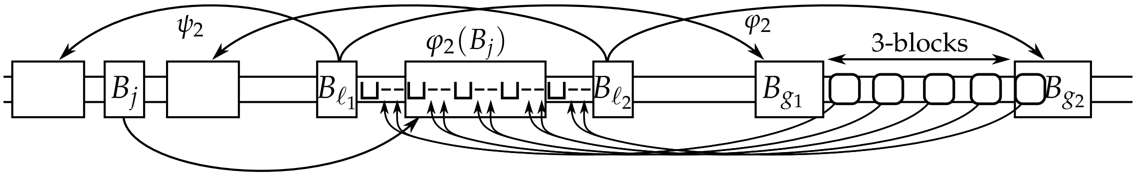


Figure 11.9: Claim 11.14.1, Case 1.ii.

Case 1.ii: $|B_j| = 2$. Let B_{ℓ_1} be the last 2-block preceding $\varphi_2(B_j)$ and B_{ℓ_2} be the first 2-block following $\varphi_2(B_j)$. Note that B_j is between $\psi_2(B_{\ell_1})$ and $\psi_2(B_{\ell_2})$, which must be in $\mathcal{P}^{(k)}$.

(a) Suppose $\{\varphi_2(B_j)\}$ is an imperfect pull set. Then $\varphi_2(B_j)$ contributes one to the defect of any pull set containing $\varphi_2(B_j)$. Place all frames containing $\varphi_2(B_j)$ into $\mathcal{H}$, as they have ν'-charge at least two. Also place the frame F starting at $\psi_2(B_{\ell_1})$ into $\mathcal{H}$. If F also contains $\psi_2(B_{\ell_2})$, it contains two disjoint maximal pull sets and thus has ν'-charge at least two. Otherwise, F must contain at least two 2-blocks which either pull a charge in Stage 2 or are separated by a block of size at least four and $\nu'(F) \geq 2$ in any case. This frame family $\mathcal{H}$ satisfies the claim.

(b) Suppose $\{\varphi_2(B_j)\}$ is a perfect pull set. Therefore, $|\varphi_2(B_j)| = 3 + 2h$ for some integer $h \geq 1$ and hence is odd. Let $B_{g_1} = \varphi_2(B_{\ell_1})$ and $B_{g_2} = \varphi_2(B_{\ell_2})$. Since B_{g_1} and B_{g_2} are in $\mathcal{P}^{(k-1)}$ and $\mathcal{P}^{(k-1)}$ is a pull set, there are only 3-blocks between B_{g_1} and B_{g_2}. Therefore, the elements $x_{g_1+1}, x_{g_1+2}, \dots, x_{g_2}$ have $x_{g_1+i+1} = x_{g_1+i} + 3$ for all $i \in \{1, \dots, g_2 - g_1 - 1\}$. The generators $3t - 1$ and $3t$ guarantee that the elements of X strictly between x_{ℓ_1} and x_{ℓ_2} are a subset of $\{x_{\ell_1} + 2 + 3i : i \in \{0, 1, \dots, g_2 - g_1\}\}$. Therefore, all blocks between B_{ℓ_1} and B_{ℓ_2} (including $\varphi_2(B_j)$) have size divisible by three. So, $|\varphi_2(B_j)|$ is an odd multiple of three, but strictly larger than three; $|\varphi_2(B_j)| \geq 9$ and $|\varphi_2^{-1}(\varphi_2(B_j))| \geq 3$.

There are $t - 2$ frames containing the first three 2-blocks in $\varphi_2^{-1}(\varphi_2(B_j))$. Since these 2-blocks are consecutive, each frame pulls two charge in Stage 2. Also, let F' be the frame whose last two blocks are the first two 2-blocks in $\varphi_2^{-1}(\varphi_2(B_j))$ and let F'' be the frame whose first two blocks are the last two 2-blocks in $\varphi_2^{-1}(\varphi_2(B_j))$. Either F' contains $\psi_2(B_{\ell_1})$ or contains another 2-block preceding $\varphi_2^{-1}(\varphi_2(B_j))$ and thus $\nu'(F') \geq 2$; by symmetric argument, $\nu'(F'') \geq 2$. Let $\mathcal{H}$ contain these frames and note that $\sum_{F\in\mathcal{H}}[\nu'(F) - 1] \geq t$. Also, add the frame F_i whose last block is $\varphi_2(B_j)$ to $\mathcal{H}$. If this frame is already included in $\mathcal{H}$, then the charge contributed by $\varphi_2(B_j)$ was not counted in the previous bound and $\sum_{F \in \mathcal{H}}[\nu'(F) - 1] \geq t + 1$. Otherwise, F_i does not contain two 2-blocks from $\varphi_2^{-1}(\varphi_2(B_j))$ and so F_i spans fewer than $3t - 8$ elements preceding $\varphi_2(B_j)$. Thus, F_i contains at least two 2-blocks which are separated either by only 3-blocks (where F_i pulls a charge in Stage 2) or by a block of size at least four (which contributes at least an additional charge to F_i) and so $\nu'(F_i) \geq 2$ and $\sum_{F \in \mathcal{H}}[\nu'(F) - 1] \geq t + 1$.

Case 2: $\mathcal{P}^{(k)}$ is an imperfect pull set. There is a block $B_\ell \in \mathcal{P}^{(k)}$ so that $|B_\ell| \geq 2|\varphi_2^{-1}(B_\ell)| + 4$. Since B_ℓ contributes at least one to the defect of every pull set that contains B_ℓ, every frame containing B_ℓ has ν'-charge at least two. Let F_{j_k} be the frame that starts at the last block of $\mathcal{P}^{(k)}$ and note that F_{j_k} contains at least two 2-blocks. Therefore, F_{j_k} either contains a pull set and two 2-blocks separated by only 3-blocks, two disjoint maximal pull sets, or a pull set and a 4-block and in any case has ν'-charge at least two. If F_{j_k} contains B_ℓ, then one of the pull sets in F_{j_k} is imperfect and $\nu'(F_{j_k}) \geq 3$. Therefore, let $\mathcal{H}$ contain F_{j_k} and the frames containing B_ℓ, and $\mathcal{H}$ satisfies the claim.

□

Claim 11.14.2. *Let B_i be a 5-block with $x_{k_2} - 9t \leq x_i \leq x_{k_2}$. If every pull set $\mathcal{P}$ containing B_i has $|\varphi_2^{-1}(\mathcal{P})| = |\varphi_2^{-1}(B_i)| = 1$, then there is a set $\mathcal{H}$ of frames with $\sum_{F_j \in \mathcal{H}}(\nu'(F_j) - 1) \geq t + 1$.*

Proof of Claim 11.14.2. Let $B_j = \psi_2(\varphi_2^{-1}(B_i))$. If there is a pull set $\mathcal{P}$ containing B_j where $|\varphi_2^{-1}(\mathcal{P})| \geq 2$, then Claim 11.14.1 applies to $\mathcal{P}$ and we can set $\mathcal{H}$ to be the $t + 1$ frames with ν'-charge at least two. Therefore, we assume no such pull set exists. This implies $|\varphi_2^{-1}(B_j)| \in \{0, 1\}$.

We shall construct two disjoint sets $\mathcal{H}_1$ and $\mathcal{H}_2$ so that $\sum_{F\in\mathcal{H}_1}[\nu'(F_j)-1] \geq t$ and $\sum_{F\in\mathcal{H}_2}[\nu'(F)-1] \geq 1$ so $\mathcal{H} = \mathcal{H}_1 \cup \mathcal{H}_2$ satisfies $\sum_{F_j\in\mathcal{H}}(\nu'(F_j)-1) \geq t+1$. To guarantee disjointness, there are blocks that must be contained in frames of $\mathcal{H}_2$ that cannot be contained in frames of $\mathcal{H}_1$. For instance, a frame in $\mathcal{H}_2$ may contain B_j, but no frames in $\mathcal{H}_1$ may contain B_j.

If $\varphi_2^{-1}(B_j) = \emptyset$ or if $|B_j| \geq 6$, then B_j contributes one to the defect of every pull set containing B_j and hence every frame containing B_j has charge at least two. Place all of these frames in $\mathcal{H}_2$ and $\sum_{F\in\mathcal{H}_2}[\nu'(F)-1] \geq t$.

Therefore, we may assume that $|\varphi_2^{-1}(B_j)| = 1$ and $|B_j| = 5$. Hence, there are exactly $3t-4$ elements between $\varphi_2^{-1}(B_j)$ and B_j. Similarly, there are exactly $3t-4$ elements between B_j and $\psi_2^{-1}(B_j)$. In either of these regions, not all blocks may be 3-blocks. Let B_{g_1} be the last non-3-block preceding B_j and B_{g_2} be the first non-3-block following B_j. We shall guarantee that all frames in $\mathcal{H}_2$ contain at least one of B_j, B_{g_1}, or B_{g_2}.

There are exactly $3t-4$ elements between $\varphi_2^{-1}(B_i)$ and B_i. Since $3t-4 \equiv 2 \pmod 3$, this range contains at least one 2-block, two 4-blocks, or one block of order at least five. Let B_{ℓ_1} be the first non-3-block following $\varphi_2^{-1}(B_i)$ and B_{ℓ_2} be the first non-3-block preceding B_i.

Figure 11.10 demonstrates the arrangement of the blocks $B_i, B_j, B_{g_1}, B_{g_2}, B_{\ell_1}$, and B_{ℓ_2}, as well as two blocks B_{h_1} and B_{h_2} which will be selected later in a certain case based on the sizes of B_{g_1} and B_{g_2}.

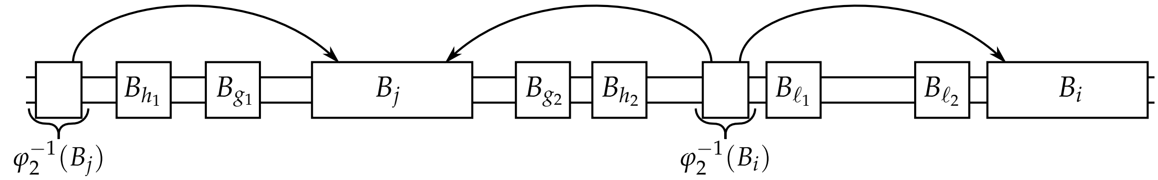


Figure 11.10: The blocks involved in the proof of Claim 11.14.2.

We consider cases depending on $|B_{\ell_1}|$ and $|B_{\ell_2}|$ and either find a contradiction or find at least one frame F to place in $\mathcal{H}_1$ so that F does not contain B_j or B_{g_2} and $[\nu'(F) - 1] \geq 1$.

Case 1: $|B_{\ell_1}| = 2$. The block $\varphi_2(B_{\ell_1})$ follows B_i. If all blocks between B_i and $\varphi_2(B_{\ell_1})$ are 3-blocks, then B_i and $\varphi_2(B_{\ell_1})$ are contained in a common pull set $\mathcal{P}$ with $|\varphi_2^{-1}(\mathcal{P})| \geq 2$, which we assumed does not happen. Therefore, there is a block B_k between B_i and $\varphi_2(B_{\ell_1})$ that is not a 3-block. If B_k is a 2-block, then $\psi_2(B_k)$ would be a large block between $\varphi_2^{-1}(B_i)$ and B_{ℓ_1}), a contradiction. If B_k is a 4-block, then $\psi_4(B_k)$ would be a large block between $\varphi_2^{-1}(B_i)$ and B_{ℓ_1}), another contradiction. Therefore, $|B_k| \geq 5$, but $\varphi_2^{-1}(B_k) = \emptyset$, since otherwise a 2-block from $\varphi_2^{-1}(B_k)$ would be strictly between $\varphi_2^{-1}(B_i)$ and B_{ℓ_1}. Then, every frame containing B_k has ν'-charge at least two. The frame F_k does not contain B_j, B_{g_1}, or B_{g_2}, so place F_k in $\mathcal{H}_1$.

Case 2: $|B_{\ell_2}| \geq 5$. If $\varphi_2^{-1}(B_{\ell_2}) \neq \emptyset$, B_{ℓ_2} and B_i are in a common pull set $\mathcal{P}$ with $|\varphi_2^{-1}(\mathcal{P})| \geq 2$, but we assumed this did not happen. Therefore, $\varphi_2^{-1}(B_{\ell_2}) = \emptyset$ and every frame containing B_{ℓ_2} has ν'-charge at least two. The frame F_{ℓ_2} does not contain B_j, B_{g_1}, or B_{g_2}, so place F_{ℓ_2} in $\mathcal{H}_1$.

Case 3: $|B_{\ell_1}| \geq 5$. Since B_{ℓ_1} and B_i cannot be in a pull set, there is a non-3-block between B_{ℓ_1} and B_i, so $B_{\ell_1} \neq B_{\ell_2}$.

Case 3.i: $|B_{\ell_2}| = 2$. The frame F starting at $\psi_2(B_{\ell_2})$ also contains B_{ℓ_1} but does not contain B_j or B_{g_2}. Since $\varphi_2^{-1}(B_i)$ is between $\psi_2(B_{\ell_2})$ and B_{ℓ_1}, these blocks are in different pull sets and so $\nu'(F) \geq 2$. Place F in $\mathcal{H}_1$.

Case 3.ii: $|B_{\ell_2}| = 4$. The frame F starting at B_{ℓ_1} also contains B_{ℓ_2} but not B_j or B_{g_2}. Since F contains two 4-blocks, $\nu'(F) \geq 2$. Place F in $\mathcal{H}_1$.

Case 4: $|B_{\ell_1}| = 4$. Since $3t - 4 \not\equiv 4 \pmod 3$, B_{ℓ_1} cannot be the only non-3-block between $\varphi_2^{-1}(B_i)$ and B_i, so $B_{\ell_1} \neq B_{\ell_2}$. Consider F_{ℓ_1}, the frame starting at B_{ℓ_1}.

If F_{ℓ_1} does not contain two 2-blocks, $\sigma(F_{\ell_1}) \geq 3t - 4$ and F_{ℓ_1} contains B_i (and B_{ℓ_2}). If $|B_{\ell_2}| = 2$, then since $4 + 2 \not\equiv 3t - 4 \pmod 3$ there is another block B_k between B_{ℓ_1} and B_i that is not a 3-block. Since F_{ℓ_1} does not contain two 2-blocks, $|B_k| \geq 4$ and therefore $\nu'(F_{\ell_1}) \geq 2$. Place F_{ℓ_1} in $\mathcal{H}_1$ and note that F_{ℓ_1} does not contain B_j, B_{g_1}, or B_{g_2}.

If F_{ℓ_1} does contain two 2-blocks, then either those two 2-blocks pull an extra charge in Stage 2, or they are separated by a block of size at least four. In either case, $\nu'(F_{\ell_1}) \geq 2$ so place F_{ℓ_1} in $\mathcal{H}_1$.

We now turn our attention to placing frames in $\mathcal{H}_2$ based on the sizes of B_{g_1} and B_{g_2}. Note that $\varphi_2^{-1}(B_{g_1}) = \varphi_2^{-1}(B_{g_2}) = \emptyset$, or else Claim 11.14.1 applies. If $|B_{g_1}| \geq 5$, then every frame containing B_{g_1} has ν'-charge at least two, so add these t frames to $\mathcal{H}_2$ to result in $\sum_{F \in \mathcal{H}}[\nu'(F) - 1] \geq t + 1$. Similarly, if $|B_{g_2}| \geq 5$, then every frame containing B_{g_2} has ν'-charge at least two, add these frames to $\mathcal{H}_2$. Therefore, we may assume that $|B_{g_1}|, |B_{g_2}| \in \{2, 4\}$ which provides four cases.

Case 1: $|B_{g_1}| = |B_{g_2}| = 2$. There are at most $3t - 4$ elements between B_{g_1} and $\varphi_2(B_{g_1})$ or between $\psi_2(B_{g_2})$ and B_{g_2}. Let B_{h_1} be the last non-3-block preceding B_{g_1} and B_{h_2} be the first non-3-block following B_{g_2}. If B_{h_1} is a 2-block, let $\mathcal{P}_1 = \{\varphi_2(B_{h_1}), \varphi_2(B_{g_1})\}$. There cannot be a 4-block B_k or 2-block $B_{k'}$ between $\varphi_2(B_{h_1})$ and $\varphi_2(B_{g_1})$ or else $\psi_4(B_k)$ or $\psi_2(B_{k'}$ would be between B_{h_1} and B_{g_1}. Therefore, adding any non-3-block between $\varphi_2(B_{h_1})$ and $\varphi_2(B_{g_1})$ to $\mathcal{P}_1$ makes $\mathcal{P}_1$ be a pull set where $|\varphi_2^{-1}(\mathcal{P}_1)| \geq 2$ and by Claim 11.14.1 we are done. Similarly if B_{ℓ_2}, the first non-3-block following B_{g_2}, is a 2-block, then let $\mathcal{P}_2 = \{\varphi_2(B_{\ell_2}), \varphi_2(B_{g_2})\}$

and we can expand $\mathcal{P}_2$ to a pull set where $|\varphi_2^{-1}(\mathcal{P}_2)| \geq 2$ and by Claim 11.14.1 we are done. Since we assumed this is not the case, B_{h_1} and B_{h_2} have size at least four. Either $\psi_2(B_{g_2}) = B_{h_1}$ or B_{h_1} follows $\psi_2(B_{g_2})$. Either $\varphi_2(B_{g_1}) = B_{h_2}$ or B_{h_2} precedes $\psi_2(B_{g_2})$. Thus, every frame containing B_j also contains B_{h_1} or B_{h_2} and thus contains at least a pull set and a 4-block or two maximal pull sets which implies the frame has ν'-charge at least two. Place these frames in $\mathcal{H}_2$.

Case 2: $|B_{g_1}| = |B_{g_2}| = 4$. There are at most $3t-3$ elements between B_{g_1} and $\varphi_4(B_{g_1})$ or between $\psi_4(B_{g_2})$ and B_{g_2}. Since B_{g_1} is the last non-3-block preceding B_j, either $\psi_4(B_{g_2}) = B_{g_1}$ or $\psi_4(B_{g_2})$ precedes B_{g_1}. Similarly, either $\varphi_4(B_{g_1}) = B_{g_2}$ or $\varphi_4(B_{g_1})$ follows B_{g_1}. Therefore, every frame containing B_j also contains B_{g_1} or B_{g_2} and thus contains a pull set and a 4-block which implies the frame has ν'-charge at least two. Place these frames in $\mathcal{H}_2$.

Case 3: $|B_{g_1}| = 2$ *and* $|B_{g_2}| = 4$. There are at most $3t-4$ elements between B_{g_1} and $\varphi_2(B_{g_1})$ and at most $3t-3$ elements between $\psi_4(B_{g_2})$ and B_{g_2}. Let B_{h_1} be the last non-3-block preceding B_{g_1}. If B_{h_1} a 2-block, then there is a pull set $\mathcal{P}_1 = \{\varphi_2(B_{h_1}), \varphi_2(B_{g_1})\}$ where $|\varphi_2^{-1}(\mathcal{P}_1)| \geq 2$. We assumed this is not the case, so $|B_{h_1}| \geq 4$. Either $B_{h_1} = \psi_4(B_{g_2})$ or B_{h_1} follows $\psi_4(B_{g_2})$. Therefore, every frame containing B_j also contains B_{h_1} or B_{g_2} and thus contains a pull set and a 4-block or two maximal pull sets which implies the frame has ν'-charge at least two. Place these frames in $\mathcal{H}_2$.

Case 4: $|B_{g_1}| = 4$ *and* $|B_{g_2}| = 2$. This case is symmetric to Case 3.

Thus, $\mathcal{H} = \mathcal{H}_1 \cup \mathcal{H}_2$ has been selected from $\mathcal{H}_1$ and $\mathcal{H}_2$ so that $\sum_{F \in \mathcal{H}}[\nu'(F) - 1] \geq t+1$. $\square$

Claim 11.14.3. *If there is a block B_ℓ with $|B_\ell| = 4$, $x_{k_2} - 12t \leq x_\ell \leq x_{k_2}$, and there is a block B_i between $\psi_4(B_\ell)$ and B_ℓ with $|B_i| \neq 3$, then there is a set $\mathcal{H}$ of frames so that* $\sum_{F\in\mathcal{H}} \left[\nu'(F) - 1\right] \geq t + 1$.

Proof of Claim 11.14.3. Note that it may be the case that $B_\ell = B_{k_2}$. For the remainder of the proof, B_ℓ will not be used to bound the ν'-charge of frames in $\mathcal{H}$ and all other blocks will contain elements between $x_\ell - 12t$ and x_ℓ, so these blocks will not be one of B_0, B_{k_1}, or B_{k_2}.

Let $\psi_4^{(d)}$ denote the dth composition of the map ψ_4. Let $D \geq 1$ be the first integer so that $|\psi_4^{(D)}(B_\ell)| \neq 4$, if it exists. We will select blocks $B_{\ell_1}, B_{\ell_2}, B_{\ell_3}$, and B_{ℓ_4} based on the value of D. For all $d \leq D$, let $B_{\ell_d} = \psi_4^{(d)}(B_\ell)$.

If $D < 4$, then we must use different methods to find the remaining blocks B_{ℓ_d}. Note that $|B_{\ell_D}| \geq 5$. If $|\varphi_2^{-1}(B_{\ell_D})| \geq 2$, then by Claim 11.14.1 we are done. If $|\varphi_2^{-1}(B_{\ell_D})| = 1$ and $|B_{\ell_D}| = 5$, then either there is a pull set $\mathcal{P}$ containing B_{ℓ_D} with $|\varphi_2^{-1}(\mathcal{P})| \geq 2$ and by Claim 11.14.1 we are done or every pull set $\mathcal{P}$ containing B_{ℓ_D} has $|\varphi_2^{-1}(\mathcal{P})| = 1$ and by Claim 11.14.2 we are done. Therefore, there are two remaining cases for B_{ℓ_D}: either (a) $\varphi_2^{-1}(B_{\ell_D}) = \emptyset$, or (b) $|\varphi_2^{-1}(B_{\ell_D})| = 1$ and $|B_{\ell_D}| \geq 6$.

We consider cases based on $|B_i|$.

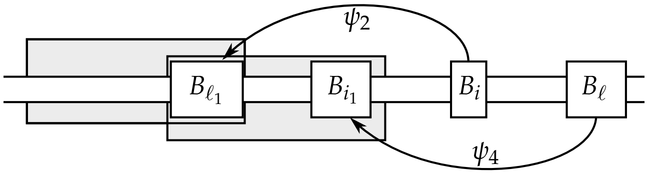


Figure 11.11: Claim 11.14.3, Case 1: $|B_\ell| = 4$ and $|B_i| = 2$, shown with $D \geq 4$.

Case 1: $|B_i| = 2$. Let $B_{i_1} = \psi_2(B_i)$. B_{i_1} is a block of size at least five preceding B_{ℓ_1}.

If there exists a pull set $\mathcal{P}$ containing B_{i_1} so that $|\varphi_2^{-1}(\mathcal{P})| \geq 2$, then by Claim

11.14.1 we are done. Therefore, $|\varphi_2^{-1}(B_{i_1})| \in \{0,1\}$.

Case 1.i: Suppose $|\varphi_2^{-1}(B_{i_1})| = 1$. If $|B_{i_1}| = 5$, then by Claim 11.14.2 we are done. Therefore, $|B_{i_1}| \geq 6$ and B_{i_1} contributes at least one to the defect of every pull set containing B_{i_1}, so every frame containing B_{i_1} has ν'-charge at least two. Place these frames in $\mathcal{H}$.

There are at most $3t-4$ elements between B_{i_1} and B_i, so if does not contain B_{ℓ_1}, then F_{i_1} contains at least two 2-blocks. If these 2-blocks are separated only by 3-blocks, then $\nu'(F_{i_1}) \geq 3$ because the imperfect pull set containing B_{i_1} contributes two charge and these 2-blocks pull one charge in Stage 2. Otherwise, these 2-blocks are separated by some block of order at least four. Therefore, $\nu'(F_{i_1}) \geq 3$ since the imperfect pull set containing B_{i_1} contributes two charge and either the 4-blocks between the 2-blocks contributes one charge or the block of size at least five between the 2-blocks is contained in a pull set that contributes at least one charge. Thus, if F_{i_1} does not contain B_{ℓ_1}, we are done. We now assume that $B_{\ell_1} \in F_{i_1}$.

If $D \geq 2$, then $|B_{\ell_1}| = 4$. Then $\nu'(F_{i_1}) \geq 3$ because the imperfect pull set containing B_{i_1} contributes two charge and B_{ℓ_1} contributes one charge.

If $D = 1$, then $|B_{\ell_1}| \geq 5$. If $\varphi_2^{-1}(B_{\ell_1}) = \emptyset$, then B_{ℓ_1} contributes two charge to F_{i_1} and $\nu'(F_{i_1}) \geq 4$. Otherwise $|\varphi_2^{-1}(B_{\ell_1})| = 1$ and $|B_{\ell_1}| \geq 6$, so B_{ℓ_1} contributes at least one to the defect of any pull set containing B_{ℓ_1} and thus $\nu'(F_{i_1}) \geq 3$.

Since $\mathcal{H}$ contains t frames of ν'-charge at least two and at least one frame (F_{i_1}) with ν'-charge at least three, $\sum\limits_{F\in\mathcal{H}} \left[\nu'(F)-1\right] \geq t+1$.

Case 1.ii: Suppose $|\varphi_2^{-1}(B_{i_1})| = 0$. B_{i_1} contributes at least two to the ν'-charge for every frame containing B_{i_1}. Place these t frames in $\mathcal{H}$. As in Case 1.i, the frame F_{i_1} must have charge $\nu'(F_{i_1}) \geq 3$ and $\sum_{F \in \mathcal{H}} \left[\nu'(F) - 1\right] \geq t + 1$.

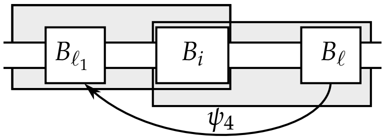


Figure 11.12: Claim 11.14.3, Case 2: $|B_\ell| = 4$ and $|B_i| = 2$, shown with $D \geq 4$.

Case 2: $|B_i| \geq 5$. Let $\mathcal{H}$ be the frames containing B_i. If there exists a pull set $\mathcal{P}$ containing B_i with $|\varphi_2^{-1}(\mathcal{P})| \geq 2$, then by Claim 11.14.1, we are done. If $|B_i| = 5$ and $|\varphi_2^{-1}(B_i)| = 1$, then by Claim 11.14.2, we are done. Therefore, either $\varphi_2^{-1}(B_i) = \emptyset$ and $|B_i| \geq 5$, or $|\varphi_2^{-1}(B_i)| = 1$ and $|B_i| \geq 6$. In either case, B_i contributes at least two charge to every frame in $\mathcal{H}$.

Consider the frame $F_{i-t+1} \in \mathcal{H}$ where B_i is the last block of F_{i-t+1}.

If F_{i-t+1} has fewer than two 2-blocks, then $\sigma(F_{i-t+1}) \geq 2 + 3(t-2) + |B_i| \geq 3t + 1$. Since there are at most $3t - 3$ elements between B_{ℓ_1} and B_ℓ, then $B_{\ell_1} \in F_{i-t+1}$ when F_{i-t+1} has fewer than two 2-blocks. If $|B_{\ell_1}| = 4$, then B_{ℓ_1} contributes another charge to F_{i-t+1} and $\nu'(F_{i-t+1}) \geq 3$. If $|B_{\ell_1}| \geq 5$ and $\varphi_2^{-1}(B_{\ell_1}) = \emptyset$ and B_{ℓ_1} contributes at least two charge to F_{i-t+1} and $\nu'(F_{i-t+1}) \geq 4$. Otherwise, $|B_{\ell_1}| \geq 5$ and $\varphi_2^{-1}(B_{\ell_1}) \neq \emptyset$. Since B_i is not contained within any pull set $\mathcal{P}$ with $|\varphi_2^{-1}(\mathcal{P})| \geq 2$, then either $\varphi_2^{-1}(B_i) = \emptyset$ or B_i and B_{ℓ_1} are not contained in a common pull set. In either case, B_{ℓ_1} contributes at least one more charge to F_{i-t+1} and $\nu'(F_{i-t+1}) \geq 3$.

If F_{i-t+1} has two or more 2-blocks, then either two 2-blocks are separated only by 3-blocks and contribute an extra charge to F_{i-t+1} or they are separated by a block of size at least four which is not in a pull set with B_i and contributes an extra charge to F_{i-t+1}.

Therefore, $\nu'(F_{i-t+1}) \geq 3$ and $\sum_{F \in \mathcal{H}} [\nu'(F) - 1] \geq t + 1$.

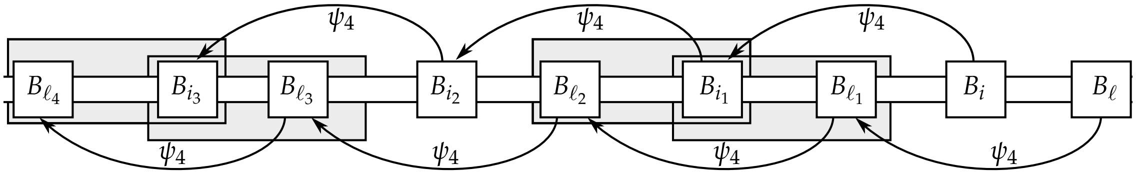


Figure 11.13: Claim 11.14.3, Case 3: $|B_\ell| = 4$ and $|B_i| = 4$, shown with $D \geq 4, D' \geq 3$.

Case 3: $|B_i| = 4$. Let $D' \geq 1$ be the first integer so that $|\psi_4^{(D')}(B_i)| \neq 4$. For $d \in \{1, \ldots, D'\}$, define $B_{i_d} = \psi_4^{(d)}(B_i)$.

Case 3.i: $D \geq 4$ and $D' \geq 3$. Note that for $j \in \{1, 2, 3\}$, B_{i_j} is between $B_{\ell_{j+1}}$ and B_{ℓ_j}. There are at most $3t - 3$ elements between $B_{\ell_{j+1}}$ and B_{ℓ_j}, so every frame F containing B_{i_j} either contains one of $B_{\ell_{j+1}}$ or B_{ℓ_j} or has $\sigma(F) \leq 3t - 4$. If F contains B_{i_j} and one of $B_{\ell_{j+1}}$ or B_{ℓ_j}, then either $\nu'(F) \geq 2$ or B_{i_j} is contained in a perfect pull set $\mathcal{P}$ with the other block and $|\varphi_2^{-1}(\mathcal{P})| \geq 2$ so by Claim 11.14.1 we are done. If $\sigma(F) \leq 3t - 3$, then there are at least three 2-blocks in F. At least two of these 2-blocks are on a common side of B_{i_j}, and either they are separated only by 3-blocks (and pull an extra charge to F) or they are separated by a block of size at least four (which contributes an extra charge to F). Therefore, every frame containing B_{i_j} has ν'-charge at least two. Build $\mathcal{H}$ from the frames containing B_{i_1} and the frames containing B_{i_3}. Then $\sum_{F \in \mathcal{H}} [\nu'(F) - 1] \geq 2t$.

Case 3.ii: $D' < D < 4$. By definition, $|B_{i_{D'}}| \geq 5$. Let $\mathcal{H}$ be the set of frames containing $B_{i_{D'}}$.

If there exists a pull set $\mathcal{P}$ containing $B_{i_{D'}}$ so that $|\varphi_2^{-1}(\mathcal{P})| \geq 2$ then by Claim 11.14.1 we are done. If $|\varphi_2^{-1}(B_{i_{D'}})| = 1$ and $|B_{i_{D'}}| = 5$, then by Claim 11.14.2

we are done. Therefore, $B_{i_{D'}}$ contributes at least one to the defect of every pull set containing $B_{i_{D'}}$ and hence every frame containing $B_{i_{D'}}$ has ν'-charge at least two.

The block $B_{i_{D'}}$ is between $B_{\ell_{D'+1}}$ and $B_{\ell_{D'}}$ and there are at most $3t-3$ elements between $B_{\ell_{D'+1}}$ and $B_{\ell_{D'}}$. Consider the frame $F_{i_{D'}}$, which has $B_{i_{D'}}$ as the first block. If $F_{i_{D'}}$ contains $B_{\ell_{D'}}$, then $\nu'(F_{i_{D'}}) \geq 3$ since $B_{\ell_{D'}}$ is a 4-block and $B_{i_{D'}}$ contributed two charge to $F_{i_{D'}}$. Otherwise, $\sigma(F_{i_{D'}}) \leq 3t-3$ and $F_{i_{D'}}$ contains at least two 2-blocks. Either these 2-blocks are separated by 3-blocks and pull a charge in Stage 2, or there is a block of size at least four between these blocks and contributes at least one more charge to $F_{i_{D'}}$. Therefore, $\nu'(F_{i_{D'}}) \geq 3$ and $\sum_{F\in\mathcal{H}} [\nu'(F)-1] \geq t+1$.

Case 3.iii: $D \leq D' < 4$. By definition, $|B_{\ell_D}| \geq 5$. Let $\mathcal{H}$ be the set of frames containing B_{ℓ_D}.

If there exists a pull set $\mathcal{P}$ containing B_{ℓ_D} so that $|\varphi_2^{-1}(\mathcal{P})| \geq 2$ then by Claim 11.14.1 we are done. If $|\varphi_2^{-1}(B_{\ell_D})| = 1$ and $|B_{\ell_D}| = 5$, then by Claim 11.14.2 we are done. Therefore, B_{ℓ_D} contributes at least one to the defect of every pull set containing B_{ℓ_D} and hence every frame containing B_{ℓ_D} has ν'-charge at least two.

The block B_{ℓ_D} is between B_{i_D} and $B_{i_{D-1}}$ and there are at most $3t-3$ elements between B_{i_D} and $B_{i_{D-1}}$. Consider the frame F_{ℓ_D}, which has B_{ℓ_D} as the first block. If F_{ℓ_D} contains $B_{i_{D-1}}$, then $\nu'(F_{\ell_D}) \geq 3$ since $B_{i_{D-1}}$ is a 4-block and B_{ℓ_D} contributed two charge. Otherwise, $\sigma(F_{\ell_D}) \leq 3t-3$ and F_{ℓ_D} contains at least two 2-blocks. Either these 2-blocks are separated by 3-blocks and pull a charge in Stage 2, or there is a block of size at least four between these blocks and contributes at least one more charge to F_{ℓ_D}. Therefore, $\nu'(F_{\ell_D}) \geq 3$ and

$\sum_{F \in \mathcal{H}} [\nu'(F) - 1] \geq t + 1$. □

Since $\sum_{j=1}^{r} \nu'(F_j) = r - 1$, there is some frame F_z with $\nu'(F_z) = 0$. Also, the only frames where $\nu'(F_j)$ may be zero are those containing B_0, B_{k_1}, or B_{k_2}.

Claim 11.14.4. *There exists a block B_* and a frame F_z so that $B_* \in F_z$, $\nu'(F_z) = 0$, and for all 2-blocks B_j, B_* does not appear between $\psi_2(B_j)$ and $\varphi_2(B_j)$, inclusive.*

Proof of Claim 11.14.4. Using any frame F_z with $\nu'(F_z) = 0$, we will show that there is a block $B_* \in \{B_0, B_{k_1}, B_{k_2}\} \cap F_z$ so that for all 2-blocks B_j, B_* does not appear between $\psi_2(B_j)$ and $\varphi_2(B_j)$.

Consider five cases based on which blocks (B_0, B_{k_1}, or B_{k_2}) are within F_z and if there are other frames with zero charge.

Case 1: For some $i \in \{1,2\}$, $B_{k_i} \in F_z$ and $|B_{k_i}| = 4$. Since $\nu'(F_z) = 0$, we must have that either $\nu^*(F_z) = 0$ or $\nu^*(F_z) > 0$ and charge was pulled from F_z in Stage 2.

If $\nu^*(F_z) = 0$, then F_z contains no block of size at least four other than B_{k_i}. If there are no 2-blocks, then every block of $F_z \setminus \{B_{k_i}\}$ is a 3-block and $\sigma(F_z) = 3t + 1$. All 2-blocks B_j have at most $3t - 4$ elements between B_j and $\varphi_2(B_j)$ or between $\psi_2(B_j)$ and B_j, so there are not enough elements to fit F_z in these ranges and hence $B_* = B_{k_i}$ suffices.

If there is exactly one 2-block in F_z, then $\sigma(F_z) = 3t$, a contradiction. Similarly, if there are exactly two 2-blocks in F_z, then $\sigma(F_z) = 3t - 1$, a contradiction. Hence, there are at least three 2-blocks in F_z and some pair of 2-blocks is separated by only 3-blocks, so Stage 2 pulled at least one charge from another frame, contradicting $\nu'(F_z) = 0$.

If $\nu^*(F_z) > 0$, then there must be at least one block of order four or more other than B_{k_i}. If any of these blocks are 4-blocks, then the positive charge contributed cannot be removed by Stage 2. If any of these blocks have size at least five, the associated maximal pull set in F_z does not contain B_{k_1} or B_{k_2} so the defect is non-negative and Stage 2 leaves at least one charge, so $\nu'(F_z) > 0$.

Case 2: $B_{k_1} \in F_z$ and $|B_{k_1}| \geq 5$. Since $x_0 + 3t \in B_{k_1}$ and B_0 is not included in $\varphi_2^{-1}(B_{k_1})$, we have $|B_{k_1}| \geq 2|\varphi_2^{-1}(B_{k_1})| + 4$. Thus the maximal pull set in F_z containing B_{k_1} is imperfect and $\nu'(F_z) > 0$, a contradiction.

Case 3: $B_0 \in F_z$, there are no 2-blocks in F_z, and F_z does not contain B_{k_1} or B_{k_2}. Since $\nu'(F_z) = 0$, there is no block in F_z with size at least four, hence F_z contains $t-1$ 3-blocks and B_0, so $\sigma(F_z) = 3t-2$. For a 2-block B_j, there are at most $3t-4$ elements contained in the blocks strictly between B_j and $\varphi_2(B_j)$ or the blocks strictly between B_j and $\psi_2(B_j)$. Then, if B_0 appears between $\psi_2(B_j)$ and $\varphi_2(B_j)$, then one of $\psi_2(B_j)$, B_j, or $\varphi_2(B_j)$ must be within F_z, a contradiction. Thus, $B_* = B_j$ suffices.

Case 4: $B_0 \in F_z$, F_z contains at least one 2-block, F_z does not contain B_{k_1} or B_{k_2}. Since F_z does not contain B_{k_1} or B_{k_2}, any block of size at least four implies $\nu'(F_z) \geq 1$, a contradiction. Further, if there are at least three 2-blocks in F_z, then two 2-blocks are separated by only 3-blocks and F_z pulls a charge in Stage 2, a contradiction. Therefore, F_z contains either one or two 2-blocks. If there are two 2-blocks, there must be one 2-block (call it B_{i_1}) preceding B_0 and another (call it B_{i_2}) following B_0. In either case, $\sigma(F_z) \in \{3t-4, 3t-3\}$.

Let B_{ℓ_1} be the block immediately following F_z and B_{ℓ_2} be the block immediately preceding F_z. If $\sigma(F_z) = 3t-3$ and B_{ℓ_j} has size two or three (for some $j \in \{1,2\}$), then $\sigma(F_z \cup \{B_{\ell_j}\}) \in \{3t-1, 3t\}$, a contradiction. If $\sigma(F_z) = 3t-4$ and $|B_{\ell_j}| \in \{3,4\}$ (for some $j \in \{1,2\}$), then $\sigma(F_z \cup \{B_{\ell_i}\}) \in \{3t-1, 3t\}$, a contradiction.

Hence, $|B_{\ell_1}|, |B_{\ell_2}| \geq 4$ when exactly one 2-block exists, or $|B_{\ell_j}| = 2$ and the 2-block B_{i_j} is between B_0 and B_{ℓ_j} (and every frame containing both B_{i_j} and B_{ℓ_j} pulls a charge in Stage 2). Since all other frames containing B_0 contain either B_{ℓ_1} or B_{ℓ_2}, they have positive ν'-charge. Therefore, F_z is the *only* frame with zero charge and $\sum\limits_{j:\nu'(F_j)>0} [\nu'(F_j) - 1] = 0$. Hence, if there exists any frame with ν'-charge at least two, we have a contradiction.

We consider if B_{i_1} and B_{i_2} both exist and whether or not $\psi_2(B_{i_j})$ is equal to B_{k_2} for some j.

Case 4.i: $\psi_2(B_{i_j}) = B_{k_2}$ for some $j \in \{1,2\}$. Since $|B_{k_2}| \geq 2|\psi_2^{-1}(B_{k_2})| + 4$, $|B_{k_2}| \geq 6$. If $\varphi_2^{-1}(B_{k_2}) = \emptyset$, then $\mu^*(B_{k_2}) \geq 2$ and every frame containing B_{k_2} has ν'-charge at least two, a contradiction. If $|\varphi_2^{-1}(B_{k_2})| \geq 2$, Claim 11.14.1 implies $\sum\limits_{j:\nu'(F_j)>0} [\nu'(F_j) - 1] \geq t + 1$, a contradiction. Thus, $|\varphi_2^{-1}(B_{k_2})| = 1$. Let B_g be the unique 2-block in $\varphi_2^{-1}(B_{k_2})$. Note that $|\psi_2(B_g)| \geq 5$. If $|\psi_2(B_g)| \geq 2|\varphi_2^{-1}(\psi_2(B_g))| + 4$, then $\psi_2(B_g)$ contributes one to the defect of every pull set containing $\psi_2(B_g)$ and every frame containing $\psi_2(B_g)$ has ν'-charge at least two, a contradiction. Thus, $|\psi_2(B_g)| = 2|\varphi_2^{-1}(\psi_2(B_g))| + 3 \geq 5$ and every pull set $\mathcal{P}$ which contains $\psi_2(B_g)$ has $|\varphi_2^{-1}(\mathcal{P})| \geq 1$. If any such pull set has $|\varphi_2^{-1}(\mathcal{P})| \geq 2$, then Claim 11.14.1 implies $\sum_{j:\nu'(F_j)>0}[\nu'(F_j) - 1] \geq t + 1$. Otherwise, every pull set containing $\psi_2(B_g)$ has $|\varphi_2^{-1}(\mathcal{P})| = 1$ and Claim 11.14.2 implies $\sum_{j:\nu'(F_j)>0}[\nu'(F_j) - 1] \geq t + 1$.

Case 4.ii: $\psi_2(B_{i_j}) \neq B_{k_2}$ for both $j \in \{1,2\}$. Consider some $j \in \{1,2\}$ so that B_{i_j} exists. If $|\varphi_2^{-1}(\psi_2(B_{i_j}))| \geq 2$, then Claim 11.14.1 provides a contradiction. If $|\psi_2(B_{i_j})| \geq 2|\varphi_2^{-1}(\psi_2(B_{i_j}))| + 4$, then $\psi_2(B_{i_j})$ contributes at least one to the defect of any pull set containing $\psi_2(B_{i_j})$, and every frame containing $\psi_2(B_{i_j})$

has ν'-charge at least two, a contradiction. Therefore, the size of $\varphi_2^{-1}(\psi_2(B_{i_j}))$ is 1 and $|\psi_2(B_{i_j})| = 5$.

Every pull set $\mathcal{P}$ which contains $\psi_2(B_{i_j})$ has $|\varphi_2^{-1}(\mathcal{P})| \geq 1$. If any such pull set has $|\varphi_2^{-1}(\mathcal{P})| \geq 2$, then Claim 11.14.1 provides a contradiction. Otherwise, every pull set containing $\psi_2(B_{i_j})$ has $|\varphi_2^{-1}(\mathcal{P})| = 1$ and Claim 11.14.2 provides a contradiction.

Case 5: $B_{k_2} \in F_z$ and $|B_{k_2}| \geq 5$. If $|B_{k_2}| \geq 2|\varphi_2^{-1}(B_{k_2})| + 4$, then every pull set containing B_{k_2} is imperfect and contributes at least one charge to every frame containing B_{k_2}, including F_z, a contradiction. Hence, $|B_{k_2}| = 2|\varphi_2^{-1}(B_{k_2})| + 3$. Since we are not in Case 1 or Case 2, every frame with ν'-charge zero must contain B_{k_2} or B_0.

Suppose there is a frame $F_{z'}$ containing B_0 and not containing B_{k_2} with $\nu'(F_{z'}) = 0$. Since we are not in Case 3, $F_{z'}$ contains at least one 2-block and the proof of Case 4 shows that $F_{z'}$ is the *only* frame with ν'-charge zero containing B_0 and not containing B_{k_2}.

Therefore, there are at most $t+1$ frames with ν'-charge zero, whether or not there is a frame $F_{z'}$ with $\nu'(F_{z'}) = 0$ containing B_0 and not B_{k_2} and hence we have the inequality $\sum_{j:\nu'(F_j)>0}[\nu'(F_j) - 1] \leq t$.

If $|\varphi_2^{-1}(B_{k_2})| \geq 2$, then Claim 11.14.1 implies $\sum_{j:\nu'(F_j)>0}[\nu'(F_j) - 1] \geq t+1$. If $|\varphi_2^{-1}(B_{k_2})| = 1$, then Claim 11.14.2 implies $\sum_{j:\nu'(F_j)>0}[\nu'(F_j) - 1] \geq t+1$. In either case we have a contradiction.

This completes the proof of Claim 11.14.4 □

Thus, we have a block B_* and a frame F_z so that $B_* \in F_z$, $\nu'(F_z) = 0$, and every 2-block B_j has $B_*, \psi_2(B_j), B_j$, and $\varphi_2(B_j)$ appearing in the cyclic order of blocks of

X. Fix B_j to be the first 2-block that appears after B_* in the cyclic order. We will now prove that $a+b+c \geq 3$.

Consider $\psi_2(B_j)$. Observe that $\varphi_2^{-1}(\psi_2(B_j)) = \emptyset$, by the choice of B_* and B_j. Hence, $a \geq |\psi_2(B_j)| - 4$. If $|\psi_2(B_j)| \geq 7$, then $a \geq 3$. Thus, $|\psi_2(B_j)| \in \{5, 6\}$ and $\psi_2^{-1}(\psi_2(B_j)) = \{B_j\}$.

Consider the frame F_{j-t+1}, whose last block is B_j. By the choice of B_j, all blocks in $F_{j-t+1} \setminus \{B_j\}$ have size at least three, so $\sigma(F_{j-t+1}) \geq 3t-1$. This implies $\psi_2(B_j) \in F_{j-t+1}$. Since $\psi_2(B_j) \ni x_j - 3t$ and $|\psi_2(B_j)| \leq 6$, there are at least $3t-4$ elements strictly between $\psi_2(B_j)$ and B_j which must be covered by at most $t-2$ blocks. Therefore, there exists some block B_k strictly between $\psi_2(B_j)$ and B_j with $|B_k| \geq 4$. Select B_k to be the first such block appearing after $\psi_2(B_j)$.

Case 1: $|\psi_2(B_j)| = 6$. This implies $a \geq 2$. If $|B_k| \geq 5$, by choice of B_j we have $\varphi_2^{-1}(B_k) = \emptyset$ and $a \geq 3$. Therefore, $|B_k| = 4$ and $\psi_4(B_k)$ is a block of order at least four. If $|\psi_4(B_k)| \geq 5$, then $\varphi_2^{-1}(\psi_4(B_k)) = \emptyset$ and $a \geq 3$. Otherwise, $|\psi_4(B_k)| = 4$, and the frame F_i starting at $B_i = \psi_4(B_k)$ also contains $\psi_2(B_j)$ and B_k. Thus, $c = 1$ and $a + c \geq 3$.

Case 2: $|\psi_2(B_j)| = 5$ *and* $|B_k| \geq 5$. Note that $\varphi_2^{-1}(B_k) = \emptyset$ by choice of B_j, which implies that $a \geq 2$. If $|B_k| \geq 6$, then $a \geq 3$; hence $|B_k| = 5$. Let $B_i = \psi_2(B_j)$ and consider the set $N_k = \{x_k - 3t, x_k - 3t + 1, x_k - 3t + 5, x_k - 3t + 6\}$. The elements in N_k are non-neighbors with x_k or x_{k+1}. Since X is a clique, X is disjoint from N_k. We must consider which elements in $A_k = \{x_k - 3t + 2, x_k - 3t + 3, x_k - 3t + 4\}$ are contained in X. If B_* appears before A_k, then since B_j is the first 2-block after B_*, there is at most one element of X in A_k. If B_* appears after A_k and two elements of A_k are in X, then they form a 2-block $B_{j'}$ with $\varphi_2(B_{j'}) = B_k$, contradicting the choice of B_*. Hence, $|X \cap A_k| \leq 1$ and the elements from X in

A_k form either blocks of size at least five or two consecutive blocks of order at least four.

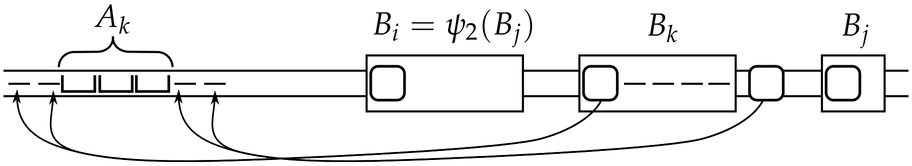


Figure 11.14: Claim 11.14, Case 2.

Case 2.i: $A_k \cap X = \emptyset$*.* Let B_ℓ be the block containing $x_k - 3t$. Note that $|B_\ell| \geq 8$. If $\varphi_2^{-1}(B_\ell) = \emptyset$, then $a \geq 4$. Otherwise $\varphi_2^{-1}(B_\ell) \neq \emptyset$, and B_* appears between B_ℓ and B_i. Then, there are at most $3t - 7$ elements between B_ℓ and B_k. Since $|B_*| \geq 1$, $|B_i| \geq 5$, and all other blocks have size at least three, the $t - 2$ blocks after B_ℓ cover at least $3t - 6$ elements. Thus, every frame containing B_* (including F_z) must also contain B_ℓ or B_k. This implies that $\nu'(F_z) \neq 0$, a contradiction.

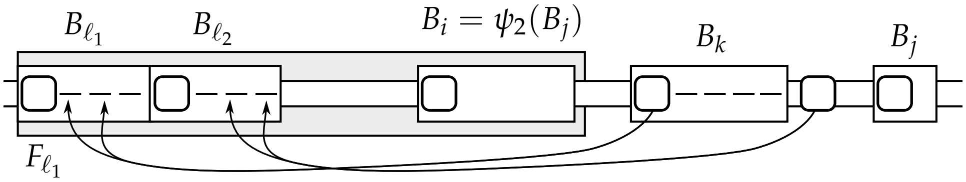


Figure 11.15: Claim 11.14, Case 2.ii.

Case 2.ii: $A_k \cap X = \{x_k - 3t + 3\}$*.* Then, the block starting at $x_k - 3t + 3$ and the block preceding it have size at least four. These two blocks (call them B_{ℓ_1} and B_{ℓ_2}) and $\psi_2(B_j)$ are contained in a single frame, F_{ℓ_1}, so $c = 1$ and $a + c \geq 3$.

Case 2.iii: $A_k \cap X \neq \{x_k - 3t + 3\}$ *and* B_* *appears before* A_k*.* Thus, the element in $A_k \cap X$ is either the first element in a block of size at least five or is the first element following a block of size at least five. In either case, this block, B_ℓ, has $\varphi_2^{-1}(B_\ell) = \emptyset$, by the choice of B_* and B_j. This implies $a \geq 3$.

Case 2.iv: $A_k \cap X \neq \{x_k - 3t + 3\}$ and B_* appears between A_k and B_i. Let B_ℓ be the block of size at least five that is guaranteed by the element in $A_k \cap X$. There are at most $3t - 3$ elements between B_ℓ and B_k. Since $|B_*| \geq 1$, $|B_i| = 5$, and all other blocks between B_ℓ and B_k have size at least three, the $t - 1$ blocks following B_ℓ cover at least $3t - 3$ elements. Thus, any frame containing B_* also contains either B_ℓ or B_k, and thus has positive charge. This includes F_z, but $\nu'(F_z) = 0$, a contradiction.

Case 3: $|\psi_2(B_j)| = 5$ and all blocks between $\psi_2(B_j)$ and B_j have size at most four. Since there are $3t - 4$ elements strictly between $\psi_2(B_j)$ and B_j that must be covered by at most $t - 2$ blocks of size at least three, there are at least two 4-blocks $B_k, B_{k'}$ between $\psi_2(B_j)$ and B_j. Thus, the blocks $B_{\ell_0} = \psi_2(B_j), B_{\ell_1} = B_k$, and $B_{\ell_2} = B_{k'}$ are contained in a single frame and $c = 1$ giving $a + c \geq 3$.

This completes the proof of Claim 11.14. □

Claims 11.13 and 11.14 imply that an r-clique X in $G + \{0, 1\}$ has no 2-blocks. By Claim 11.12, $G + \{0, 1\}$ has a unique r-clique and hence G is r-primitive. □

11.4 Sporadic Constructions

In this section, we give explicit constructions for all known r-primitive graphs, including those found in previous work. It is a simple computation to verify that every graph presented is uniquely K_r-saturated, so proofs are omitted. In addition to the descriptions given here, all graphs are available online[3].

11.4.1 Uniquely K_4-Saturated Graphs

Construction 11.17 (Cooper [31], Figure 11.16(a))**.** G_{10} is the graph built from two 5-cycles a_0, a_1, a_2, a_3, a_4 and b_0, b_1, b_2, b_3, b_4 where a_i is adjacent to b_{2i-1}, b_{2i}, and b_{2i+1}.

Construction 11.18 (Collins [31], Figure 11.16(b))**.** The graph G_{12} is the vertex graph of the icosahedron with a perfect matching added between antipodal vertices. Another description takes vertices v_0, v_1 and two 5-cycles $u_{j,0}, \dots, u_{j,4}$ ($j \in \{0,1\}$) with v_j adjacent to v_{j+1} and $u_{j,i}$ for all $i \in [5]$ and $u_{0,i}$ adjacent to $u_{1,i}$, $u_{1,i+1}$, and $u_{1,i+3}$ for all $i \in \mathbb{Z}_5$.

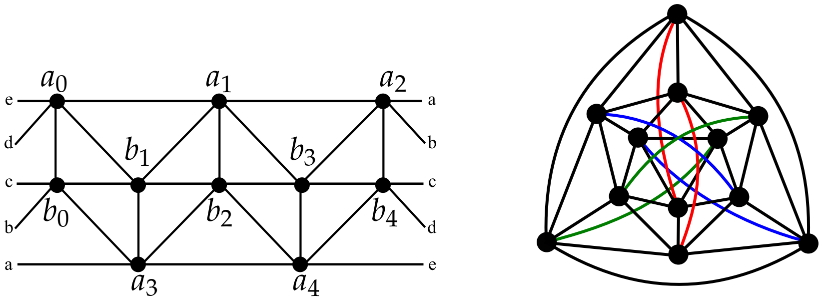
Construction 11.19 (Figure 11.16(c))**.** G_{13} is given by vertices $x, y_1, \dots, y_6, z_1, \dots, z_6$, where x is adjacent to every y_i, y_i and y_{i+1} are adjacent for all $i \in \{1, \dots, 6\}$, and z_i and z_{i+1} are adjacent for all $i \in \{1, \dots, 6\}$. Further, z_i is adjacent to z_{i+3}, y_i, y_{i-1}, and y_{i+2}.

Construction 11.20 (Figure 11.16(d))**.** The Paley graph [105] of order 13, Paley(13), is isomorphic to the Cayley complement $\overline{C}(\mathbb{Z}_{13}, \{1,3,4\})$.

Construction 11.21 (Figure 11.17)**.** Let H be the graph on vertices $x, v_1, \dots, v_5$ with x adjacent to every v_i and the vertices $v_1, \dots, v_5$ form a 5-cycle. Note that H is uniquely K_4-saturated, as $v_1, \dots, v_5$ induce C_5, which is 3-primitive. $G_{18}^{(A)}$ has vertex set $V = \{1,2,3\} \times \{x, v_1, v_2, v_3, v_4, v_5\}$. A vertex (a, x) or (a, v_i) in V considers the number a modulo three and i modulo 5. The vertices (a, x) with $a \in$

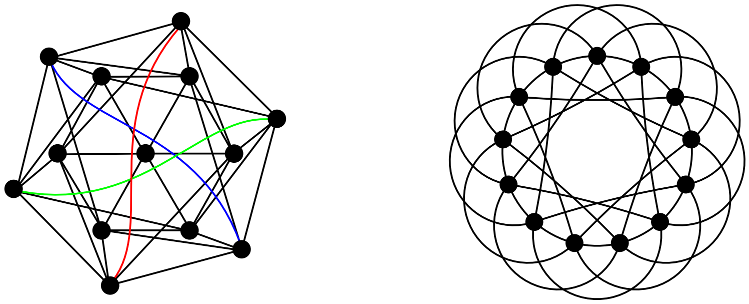
---

[3] Graphs available in graph6 format or as adjacency matrices at `http://www.math.unl.edu/~shartke2/math/data/data.php`.



(a) Construction 11.17, G_{10}.

(b) Construction 11.18, G_{12}.



(c) Construction 11.19, G_{13}.

(d) Construction 11.20, Paley(13).

Figure 11.16: Uniquely K_4-saturated graphs on 10–13 vertices.

$\{1,2,3\}$ form a triangle. For each a, (a,x) is adjacent to (a,v_i) for each i but is not adjacent to $(a+1,v_i)$ or $(a+2,v_i)$ for any i. For each a and i, the vertex (a,v_i) is adjacent to (a,v_{i-1}) and (a,v_{i+1}) (within the copy of H) and also $(a+1,v_{i+2}),(a+1,v_{i-2}),(a-1,v_{i+2}),(a-1,v_{i-2})$ (outside the copy of H).

Construction 11.22 (Figure 11.18)**.** Let $G_{18}^{(B)}$ have vertex set $\mathbb{Z}_2 \times \mathbb{Z}_9$ where each coordinate is taken modulo two and nine, respectively. For fixed a, the vertices (a,i) and (a,j) are adjacent if and only if $|i-j| \leq 2$. For fixed i, the vertex $(0,i)$ is adjacent to $(1,2i),(1,2i+4)$ and $(1,2i+5)$. Conversely, for fixed j the vertex $(1,j)$ is adjacent to $(0,5j),(0,5j+7)$ and $(0,5j+2)$.

11.4.2 Uniquely K_5-Saturated Graphs

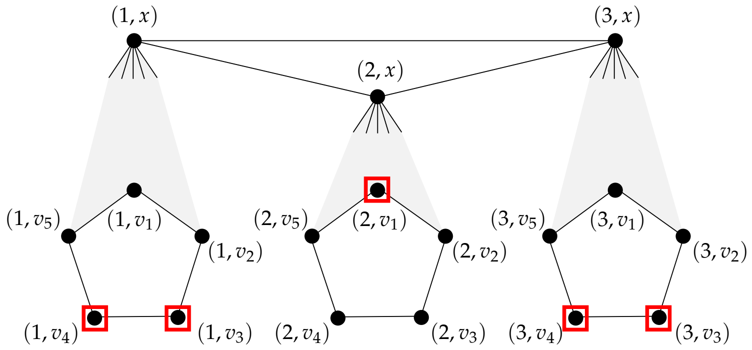
Construction 11.23 (Figure 11.19)**.** Let $G_{16}^{(A)}$ have vertex set $\{v_1, v_2\} \bigcup (\{1,2\} \times \mathbb{Z}_7)$. The vertices v_1 and v_2 are adjacent. For each $j \in \{1,2\}$ and $i \in \mathbb{Z}_7$, v_j is adjacent to (j,i) and (j,i) is adjacent to $(j,i+1),(j,i+2),(j,i-1)$ and $(j,i-2)$. (Hence, the subgraph induced by (j,i) for fixed j and $i \in \mathbb{Z}_7$ is isomorphic to C_7^2.) For $i \in \mathbb{Z}_7$, the vertex $(1,i)$ is adjacent to $(2,2i),(2,2i+1),(2,2i-1)$, and $(2,2i-3)$. Conversely, for $i \in \mathbb{Z}_7$, the vertex $(2,i)$ is adjacent to $(1,4i),(1,4i-2),(1,4i+3)$, and $(1,4i-3)$.

An interesting feature of $G_{16}^{(A)}$ is that it is not regular: v_1, and v_2 have degree 8 while the other vertices have degree 9. This is a counterexample to previous thoughts that all uniquely K_r-saturated graphs with no dominating vertex were regular.

Construction 11.24 (Figure 11.20)**.** The graph $G_{16}^{(B)}$ has vertex set $\{x\} \cup \{u_i : i \in \mathbb{Z}_3\} \cup \{v_j : j \in \mathbb{Z}_6\} \cup \{z_{k,i} : k \in \{0,1\}, i \in \mathbb{Z}_3\}$. The vertex x is adjacent to u_i for all $i \in \mathbb{Z}_3$ and v_j for all $j \in \mathbb{Z}_6$. There are no edges among the vertices u_i. The vertices v_j form a cycle, with an edge v_jv_{j+1} for all $j \in \mathbb{Z}_6$. The vertices $z_{k,i}$ form a complete bipartite graph, with an edge $z_{0,i}z_{1,j}$ for all $i,j \in \mathbb{Z}_3$. For $i \in \{0,1,2\}$, the vertex u_i is adjacent to v_{2i-1}, v_{2i}, v_{2i+1}, and v_{2i+2}, and adjacent to $z_{k,i+1}$ and $z_{k,i-1}$ for $k \in \{0,1\}$. For $i \in \{0,1,2\}$, the vertex $z_{0,j}$ is adjacent to v_{2i}, v_{2i+1}, v_{2i+2}, and v_{2i+4}, while the vertex $z_{1,i}$ is adjacent to v_{2i-1}, v_{2i}, v_{2i+1}, and v_{2i+3}.

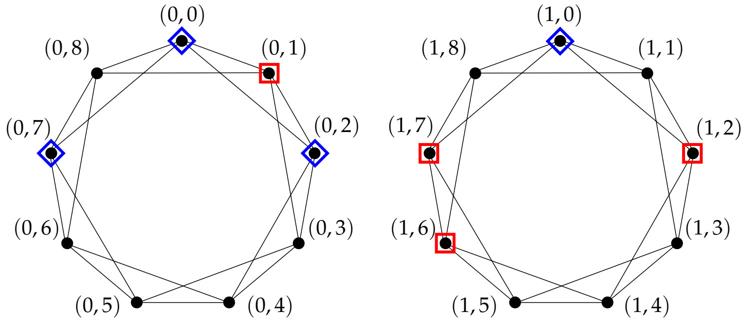
11.4.3 Uniquely K_6-Saturated Graphs

Construction 11.25 (Figure 11.21)**.** The graph $G_{15}^{(A)}$ has vertices $x, v_0, v_1, u_1, \dots, u_4$, $c_1, \dots, c_4$, $q_1, \dots, q_4$. The vertex x dominates all but the q_i's. The vertices v_0, v_1



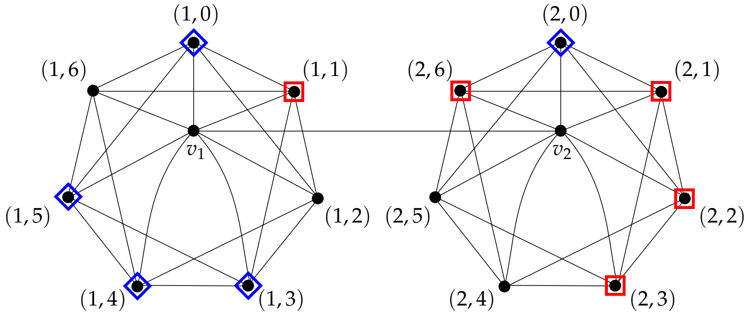
□ — $\{(2, v_1)\} \cup (N((2, v_1)) \cap \{(j, v_i) : j \in \{1, 3\}, i \in \{1, \dots, 5\}\})$.

Figure 11.17: Construction 11.21, $G_{18}^{(A)}$, is 4-primitive, 7-regular, on 18 vertices.



□ — $\{(0, 1)\} \cup (N((0, 1)) \cap \{(1, i) : i \in \mathbb{Z}_9\})$.
◇ — $\{(1, 0)\} \cup (N((1, 0)) \cap \{(0, i) : i \in \mathbb{Z}_9\})$.

Figure 11.18: Construction 11.22, $G_{18}^{(B)}$, is 4-primitive, 7-regular, on 18 vertices.



□ — $\{(1, 1)\} \cup (N((1, 1)) \cap \{(2, i) : i \in \{0, 1, \dots, 6\}\})$.
◇ — $\{(2, 0)\} \cup (N((2, 0)) \cap \{(1, i) : i \in \{0, 1, \dots, 6\}\})$.

Figure 11.19: Construction 11.23, $G_{16}^{(A)}$, is 5-primitive *and irregular*, on 16 vertices.

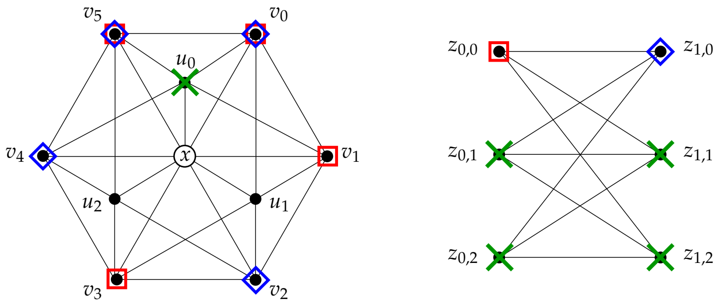
are adjacent and dominate the u_i's. Also, v_i dominates $c_{2i}, c_{2i+1}, q_{2i}, q_{2i+1}$ for each $i \in \mathbb{Z}_2$. The vertices u_0 and u_2 are adjacent as well as u_1 and u_3. The vertices u_i dominate the vertices c_j. Also, the vertex u_i is adjacent to q_j if and only if $i \neq j$. The vertices $c_1, \ldots, c_4$ form a cycle with edges $c_i c_{i+1}$. The vertices $q_1, \ldots, q_4$ form a clique. The vertices c_i and q_j are adjacent if and only if $i \neq j$.

Construction 11.26 (Figure 11.22)**.** The graph $G_{15}^{(B)}$ has vertices $q_i, c_{1,i}$, and $c_{2,i}$ for each $i \in \mathbb{Z}_5$. The subgraph induced by vertices q_i is a 5-clique. For each $j \in \{1,2\}$, the subgraph induced by vertices $c_{j,i}$ for $i \in \mathbb{Z}_5$ is isomorphic to C_5 with edges $c_{j,i}c_{j,i+1}$ between consecutive elements. For each $i, i' \in \mathbb{Z}_5$, there is an edge between $c_{1,i}$ and $c_{2,i'}$. For each $i \in \mathbb{Z}_5$, the vertex q_i is adjacent to $c_{1,i}, c_{1,i-1}$, and $c_{1,i+1}$ as well as $c_{2,2i}$, $c_{2,2i-1}$, and $c_{2,2i+2}$.

Construction 11.27 (Figure 11.23)**.** The graph $G_{16}^{(C)}$ is composed of three disjoint induced subgraphs isomorphic to K_4, K_4, and $\overline{C_8}$. Let the vertices $q_{0,0}, \ldots, q_{0,3}$, and $q_{1,0}, \ldots, q_{1,3}$ be the two copies of K_4 and vertices $c_0, \ldots, c_7$ be the $\overline{C_8}$, where the non-edges are for consecutive elements $(0,i)$ and $(0,i+1)$. For $i \in \{0,1,2,3\}$, the vertex $q_{1,i}$ is adjacent to c_{2i+d} for all $d \in \{0,1,2,3,4,5\}$. For $i \in \{0,1,2,3\}$, the vertex $q_{2,i}$ is adjacent to c_{2i+d} for all $d \in \{0,1,3,4,5,6\}$. For $i \in \mathbb{Z}_4$, the vertex $q_{1,i}$ is adjacent to $q_{2,i+1}$ and $q_{2,i-1}$.

Acknowledgements

We thank David Collins, Joshua Cooper, Bill Kay, and Paul Wenger for sharing their early observations on this problem. We also thank Jamie Radcliffe for contributing to the averaging argument found in Claim 11.6.

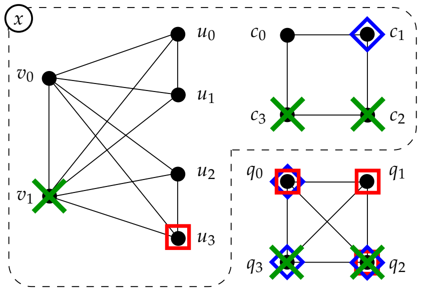


✕ — $\{u_0\} \cup \left(N(u_0) \cap \{z_{j,i} : j \in \{0,1\}, i \in \mathbb{Z}_3\}\right)$.

□ — $\{z_{0,0}\} \cup (N(z_{0,0}) \cap \{v_i : i \in \mathbb{Z}_6\})$.

◇ — $\{z_{1,0}\} \cup (N(z_{1,0}) \cap \{v_i : i \in \mathbb{Z}_6\})$.

Figure 11.20: Construction 11.24, $G_{16}^{(B)}$, is 5-primitive, 9 regular, on 16 vertices.

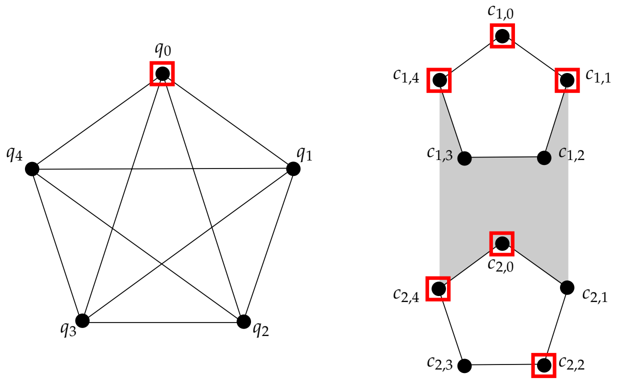


□ — $\{u_3\} \cup (N(u_3) \cap \{q_i : i \in [4]\})$.

◇ — $\{c_1\} \cup (N(c_1) \cap \{q_i : i \in [4]\})$.

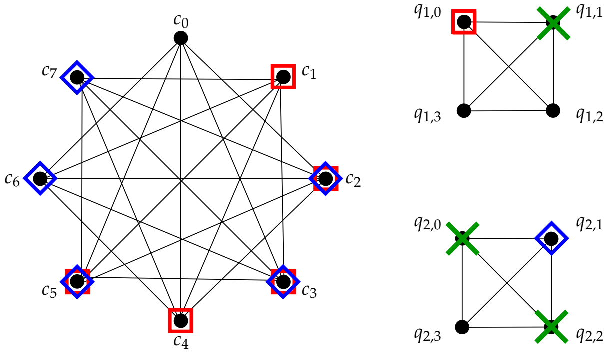
✕ — $\{v_1\} \cup (N(v_1) \cap \{c_i, q_i : i \in [4]\})$.

Figure 11.21: Construction 11.25, $G_{15}^{(A)}$, is 6-primitive, 10 regular, on 15 vertices.



□ — $\{q_0\} \cup \left(N(q_0) \cap \{c_{j,i} : j \in \{1,2\}, i \in \mathbb{Z}_5\}\right)$.

Figure 11.22: Construction 11.26, $G^{(B)}_{15}$, is 6-primitive, 10 regular, on 15 vertices.



□ — $\{q_{1,0}\} \cup (N(q_{1,0}) \cap \{c_i : i \in \mathbb{Z}_8\})$.
✕ — $\{q_{1,1}\} \cup (N(q_{1,1}) \cap \{q_{2,i} : i \in \mathbb{Z}_4\})$.
◇ — $\{q_{2,1}\} \cup (N(q_{2,1}) \cap \{c_i : i \in \mathbb{Z}_8\})$.

Figure 11.23: Construction 11.27, $G^{(C)}_{16}$, is 6-primitive, 10 regular, on 16 vertices.

Part IV

Reachability Problems in Space-Bounded Complexity

Chapter 12

Space-Bounded Computational Complexity

This chapter will define some basics of complexity theory with a focus on space-bounded computation. For a more detailed description, see [7, Chapters 1, 2, and 4]

We shall always consider *decision problems*, where we attempt to decide if a word $\mathbf{x} \in \{0,1\}^*$ is contained in a specified *language* $L \subseteq \{0,1\}^*$.

12.1 Turing Machines

The fundamental object of complexity theory is the Turing machine. These machines specify how computation works at a very low level, using a simple model. A machine has a finite list of instructions (given as a set of states and a transition function) and has access to three infinite tapes (containing cells to store bits) for input, work, and output. The machine has three tape heads which allow it to view exactly one cell of each tape, so it can read or write exactly one cell per step of computation. In space-bounded complexity, we measure the efficiency of such a machine by the number of cells that are used on the work tape; we do not count the input cells or output cells against the machine. To avoid exploitation of this fact, we specify the input as read-only (no cells can be written) and the output as write-once-only (every cell can be written exactly once and then never visited again). Further, the tape head(s) can only move locally: at every step the head can move left, right, or stay in place.

The formal definition follows.

Definition 12.1. A *Turing machine* is a tuple $M = (\Gamma, Q, \delta)$ where

1. The *tape alphabet* Γ is a finite alphabet of characters that can be stored on a tape.
2. The *state set* Q is a finite set of states. Four special states q_{start}, q_{accept}, q_{reject}, and q_{halt} have particular significance.
3. The *transition function* δ is a function

   $$\delta : Q \times \Gamma^2 \to Q \times \Gamma^2 \times \{\text{Left}, \text{Right}, \text{Stay}\}^2$$

   which defines the action of the machine at a given step.

Given an input $\mathbf{x} \in \{0,1\}^*$, the bits of $\mathbf{x}$ are placed on the input tape and all other cells are blank. The current state is set to q_{start}.

At every step of computation, the current state $q \in Q$, current input cell $x_i \in \{0,1,\sqcup\}$ at the input tape head, and current work cell $w_j \in \{0,1,\sqcup\}$ at the work tape head are given as input to the transition function δ. The transition functions outputs a new state $q' \in Q$, a value $w'_j \in \{0, 1\}$ to write to the work tape, a value $y_k \in \{0, 1, \sqcup\}$ to write to the output tape, and two directions in $\{\text{Left}, \text{Right}, \text{Stay}\}$.

In this time step:

1. The state is set to q'.
2. The input tape head moves according to the first direction.
3. The value w'_j is written to the current work cell and the work tape head moves based on the second direction.
4. If $y_k \neq \sqcup$, then y_j is written to the output tape and the tape head is advanced one position.

If the state q ever becomes q_{accept}, q_{reject}, or q_{halt}, then the machine stops. These states are called *halting states*. If the state is q_{accept}, then M *accepts* the input; if it is q_{reject}, then M *rejects* the input. The state q_{halt} can be interpreted as "I don't know."

A Turing machine M *decides* a language $L \subseteq \{0, 1\}^*$ if M halts on every input and M accepts $\mathbf{x}$ if and only if $\mathbf{x} \in L$. Conversely, for a machine M which halts on every input, the language L_M consists exactly of the words $\mathbf{x}$ where $M(\mathbf{x}) = 1$.

A *nondeterministic* Turing machine also has access to a fourth tape which is filled with a *certificate* $\mathbf{u} \in \{0, 1\}^*$. This certificate tape is read-only and can only advance in the right direction. However, the certificate can be arbitrarily long and the bits of $\mathbf{u}$ can be used in the transition function to define the behavior of the machine. Typically, a nondeterministic machine will accept an input $\mathbf{x}$ if there exists at least one assignment of bits to the certificate so that the machine halts at the q_{accept} state.

A Turing machine *halts* on an input if it reaches a halting state in a finite number of steps.

12.2 Complexity Classes

The main thrust of computational complexity theory is to determine exactly how difficult it is to compute the answers to given problems. While Turing machines answered the answer to *what* computation is, we currently lack any strong capability to prove lower bounds on some of our most important problems. However, this did not stop the community from defining *classes* of problems which share similar measures of efficiency.

A *complexity class* is a family of languages (or functions) defined in terms of the efficiency of Turing machines that decide (or compute) them. In this sense, the complexity classes are well-defined, but it is sometimes difficult to determine relations between complexity classes, especially when different types of computation (such as deterministic, nondeterministic, or randomized) are used to define the classes.

12.2.1 Time-Bounded Complexity Classes

An important resource is time. Some naïve algorithms take exponential time and even on relatively small instances would take longer than the history of the universe to finish.

In order to avoid some complications, we will restrict our time bounds to a special class of growth functions.

Definition 12.2. A function $T : \mathbb{N} \to \mathbb{N}$ is *time-computable* if $T(n) \geq n$ always and there is a Turing machine M so that on an input $\mathbf{x}$, $M(\mathbf{x})$ computes $T(|\mathbf{x}|)$ in $O(T(|\mathbf{x}|))$ steps.

Time-computable functions include n, $n \log n$, n^c, and 2^n. Non-time-computable functions include $\log n$ and $\sqrt{n}$, for trivial reasons: there is not enough time to read the entire input. The *busy beaver function* $\mathsf{BB}(n)$ is the largest number output by a Turing machine that can be described in n bits. This function is not computable, so it is not time-computable.

Definition 12.3 (Deterministic Time)**.** For a time-computable function $T(n)$, the class $\mathsf{DTIME}[T(n)]$ consists of languages which are decidable by a deterministic Turing machine M where $M(\mathbf{x})$ terminates within $c \cdot T(|\mathbf{x}|)$ steps for some constant $c > 0$.

Definition 12.4 (Nondeterministic Time)**.** For a time-computable function $T(n)$, the class $\mathsf{NTIME}[T(n)]$ consists of languages which are decidable by non-deterministic Turing machines M where $M(\mathbf{x})$ terminates within $cT(|\mathbf{x}|)$ steps for some constant $c > 0$.

Definition 12.5. The following classes are some important classes for time-bounded complexity:

Polynomial Time : $\mathsf{P} = \cup_{p \geq 1} \mathsf{DTIME}[n^p]$.

Non-deterministic Polynomial Time : $\mathsf{NP} = \cup_{p \geq 1} \mathsf{NTIME}[n^p]$.

Exponential Time : $\mathsf{EXP} = \cup_{p \geq 1} \mathsf{DTIME}[2^{n^p}]$.

Non-deterministic Exponential Time : $\mathsf{NEXP} = \cup_{p \geq 1} \mathsf{NTIME}[2^{n^p}]$.

12.2.2 Space-Bounded Complexity Classes

For space-bounded complexity, we focus on the work tape as an analogue of computer memory. In order to deal with sub-linear space bounds, we adjust the definition of a Turing machine to include three tapes:

1. The input tape is *read-only* (the machine cannot write to these cells).

2. The work tape is *read/write* (the machine can read and write to these cells).
3. The output tape is *write-only-once* (the machine writes to each cell exactly one, in order, and cannot read from these cells).

This allows the number of cells used in the work tape to be measured as a resource.

Definition 12.6. A function $s : \mathbb{N} \to \mathbb{N}$ is *space-computable* if there is a Turing machine M so that on an input $\mathbf{x}$, $M(\mathbf{x})$ computes $s(|\mathbf{x}|)$ using at most $O(s(|\mathbf{x}|))$ cells on the work tape.

The smallest space-computable function is $\log n$, since at least $\Omega(\log n)$ bits (of any finite alphabet) are required to store a representation of $n = |\mathbf{x}|$.

Definition 12.7 (Deterministic Space)**.** For a space-computable function $s(n)$, the class $\mathsf{SPACE}[s(n)]$ consists of languages which are decidable by a deterministic Turing machine M where $M(\mathbf{x})$ uses at most $c \cdot s(|\mathbf{x}|)$ positions in the work tape, for some constant $c > 0$.

Definition 12.8 (Nondeterministic Space)**.** For a space-computable function $s(n)$, the class $\mathsf{NSPACE}[s(n)]$ consists of languages which are decidable by a non-deterministic Turing machine M where $M(\mathbf{x})$ uses at most $c \cdot s(|\mathbf{x}|)$ positions in the work tape, for some constant $c > 0$.

Definition 12.9. The following classes are important for space-bounded complexity:

Log-space : $\mathsf{L} = \mathsf{SPACE}[\log n]$.

Non-deterministic Log-space : $\mathsf{NL} = \mathsf{NSPACE}[\log n]$.

Polynomial Space : $\mathsf{PSPACE} = \cup_{p \geq 1} \mathsf{SPACE}[n^p]$.

12.2.3 Space-bounded Reductions

An important tool in complexity theory is the *reduction*. A reduction maps instances of one problem into instances of another.

Definition 12.10 (Space-bounded Reduction)**.** Let $s(n)$ be a space-computable function. A language L_1 is $s(n)$*-space reducible* to a language L_2 if there exists a deterministic $s(n)$-space machine M so that for all inputs $\mathbf{x}$, $\mathbf{x}$ is in L_1 if and only if $M(\mathbf{x})$ is in L_2.

Reductions play an important role in classifying the difficulty of solving a problem. Since we lack strong lower bounds on the efficiency of Turing machines, the best we can do right now is compare the difficulty (or efficiency) of problems using reductions.

Proposition 12.11. *Let $s(n)$ and $r(n)$ be space-computable functions. If a language L_1 is $s(n)$-space reducible to a language L_2, and L_2 is solvable by an $r(n)$-space deterministic (nondeterministic) Turing machine, then L_1 can be decided by an $r(2^{O(s(n))})$-space deterministic (nondeterministic) Turing machine.*

Sketch. Let M be an $s(n)$-space deterministic Turing machine so that $\mathbf{x} \in L_1$ if and only if $M(\mathbf{x}) \in L_2$. Let N be an $r(n)$-space deterministic (nondeterministic, random) Turing machine that decides L_2. There exists a deterministic (nondeterministic, random) Turing machine P which decides L_1 by simulating $N(M(\mathbf{x}))$. This simulation uses the operation of N without storing $M(\mathbf{x})$ by querying the ith bit of $M(\mathbf{x})$ whenever the input tape is read at the ith position. (Since $r(n) \geq \log(n)$, the index i can be stored.) Segment $s(n)$ space for the queries to $M(\mathbf{x})$, noting that $M(\mathbf{x})$ can output at most $2^{O(s(n))}$ bits, and segment $r(2^{O(s(n))})$ space for the simulation of N on $M(\mathbf{x})$. □

Corollary 12.12. *Let $s(n)$ be a space-computable function. If L_1 is log-space reducible to L_2 and L_2 is decidable in deterministic (nondeterministic, random) $s(n)$-space, then L_1 is decidable in deterministic (nondeterministic, random) $s(\mathsf{poly}(n))$-space.*

12.2.4 Configurations

An important tool that is used frequently in space-bounded complexity is the *configuration graph.*

Definition 12.13 (Configurations)**.** For a Turing machine $M = (\Gamma, Q, \delta)$ which requires at most $s(n)$ work cells, a *configuration* is a tuple $(q, k_i, k_w, k_o, \mathbf{D})$ where:

1. $q \in Q$ is a state of the machine,
2. k_i, k_w, and k_o (all in $\mathbb{N}$) are the positions of the tape heads for the input, work, and output tapes, respectively.
3. $\mathbf{D} \in \Gamma^*$ is the word describing the work cells which are in use.

Note that $|\mathbf{D}| \leq s(n)$ and so a configuration can be encoded in $O(s(n))$ bits.

These configurations represent the full state of the machine at a given time step. From this information, we can completely reconstruct the rest of the execution of the machine. In fact, we can create a graph from all of the configurations by placing an edge between configurations that appear in consecutive time steps.

Definition 12.14 (Configuration Graph)**.** Let $M = (\Gamma, Q, \delta)$ be a Turing machine which requires at most $s(n)$ work cells and let $\mathbf{x} \in \{0,1\}^*$ be an input. The *configuration graph* of M on $\mathbf{x}$ is the directed graph $G_{M,\mathbf{x}}$ where the vertices are all configurations $(q, k_i, k_w, k_o, \mathbf{D})$ where $k_i, k_w, k_o, |\mathbf{D}| \leq s(|\mathbf{x}|)$ and the transition function δ

defines the edges: if the transition function allows a modification of one configuration to another, place an edge in that direction.

A deterministic machine results in a graph where every non-halting configuration has out-degree one. Note that in a nondeterministic machine, there may exist vertices with large out-degree. Any nondeterministic machine can be modified to track the current time step (in binary) which makes the configuration graph be acyclic. Thus, every configuration graph is a directed acyclic graph (DAG).

Also, there are at most $2^{O(s(n))}$ vertices in a configuration graph for a machine using $O(s(n))$ space for an input of size n.

Finally, it is not difficult (but tedious) to show that for any machine M that requires $O(s(n))$ space, there is a deterministic machine $\hat{M}$ that on input $\mathbf{x}$ outputs the configuration graph for M on $\mathbf{x}$ using at most $O(s(n))$ space. This is a crucial fact that leads to many natural complete problems.

Proposition 12.15. *Directed reachability is* NL*-complete.*

This proposition can be derived from the following facts:

1. A nondeterministic machine using $O(\log n)$ space has a configuration graph of order polynomial in n.
2. The machine accepts a given input if and only if there is a directed path from the initial configuration to an accepting configuration.
3. For any nondeterministic log-space machine M, there is a deterministic log-space machine that outputs the configuration graph for M on the input $\mathbf{x}$.

This provides a log-space reduction from any language decided by an NL-machine to directed reachability (note that the graph has order n^c for some constant c depending on the language).

Also, by nondeterministically selecting the next vertex in a $u \to v$ path, directed reachability is in NL.

Thus, $\mathsf{L} = \mathsf{NL}$ if and only if there exists a log-space machine to solve directed reachability.

12.3 Relations

The following relations are not difficult to prove:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$$

1. $\mathsf{L} \subseteq \mathsf{NL}$: $\mathsf{L} = \mathsf{SPACE}[\log n] \subseteq \mathsf{NSPACE}[\log n] = \mathsf{NL}$.
2. $\mathsf{NL} \subseteq \mathsf{P}$: Directed reachability can be solved in $\mathsf{DTIME}[n]$ using breadth-first search.
3. $\mathsf{P} \subseteq \mathsf{NP}$: For all $c \geq 1$, $\mathsf{DTIME}[n^c] \subseteq \mathsf{NTIME}[n^c]$.
4. $\mathsf{NP} \subseteq \mathsf{PSPACE}$: Every certificate of nondeterministic bits contains a polynomial number of bits. These certificates can be checked in order using polynomial space.
5. $\mathsf{PSPACE} \subseteq \mathsf{EXP}$: For any $c \geq 1$, the configuration graph for an n^c-space machine has $2^{O(n^c)}$ vertices and reachability can be determined in $\mathsf{DTIME}[2^{O(n^c)}]$.

12.4 The Big Results

Below are three of the most important results in the realm of space-bounded complexity.

Savitch's Theorem relates nondeterministic space with deterministic space.

Theorem 12.16 (Savitch's Theorem [117]). *For all space-computable functions $s(n)$,* $\mathsf{NSPACE}[s(n)] \subseteq \mathsf{SPACE}[(s(n))^2]$.

Corollary 12.17. $\mathsf{PSPACE} = \mathsf{NPSPACE}$.

The Immerman-Szelepscényi Theorem shows that nondeterministic space is closed under complement. That is, if L can be computed in $\mathsf{NSPACE}[s(n)]$, then $\overline{L}$ also can. Such a collapse is not known to exist for time complexity (nor is such a collapse expected).

Theorem 12.18 (Immerman-Szelepcsényi Theorem [67, 129]). *For all space-computable functions $s(n)$,* $\mathsf{NSPACE}[s(n)] = \mathsf{coNSPACE}[s(n)]$.

Finally, a more recent result shows that undirected reachability can be computed in log-space. This collapses two complexity classes, L and SL, where SL is the set of languages decidable by a nondeterministic log-space machine where two configurations C_1, C_2 can transition $C_1 \to C_2$ if and only if the reverse transition $C_2 \to C_1$ is also allowed.

Theorem 12.19 (Reingold's Theorem [109]). *Undirected reachability can be decided in log-space:* $\mathsf{L} = \mathsf{SL}$.

Reingold's Theorem has opened a new line of research into finding which classes of directed reachability can be computed in deterministic log-space, now that undirected reachability can be used as a subroutine. This is particularly useful when determining the components of subgraphs defined by certain properties on the edges.

Chapter 13

ReachUL = ReachFewL

A nondeterministic machine is *unambiguous* if for every input there is at most one computation path leading to an accepting configuration. UL is the class of problems that are decided by unambiguous log-space nondeterministic machines. Recent progress indicates that this unambiguous version of nondeterminism is powerful enough to capture general nondeterminism in the log-space setting [4, 111, 20, 135].

This chapter considers a more restricted version of log-space unambiguity. A nondeterministic machine is *reach-unambiguous* if for any input and for any configuration C, there is at most one path from the start configuration to C. (The prefix 'reach' in the term indicates that the property should hold for all configurations reachable from the start configuration). ReachUL is the class of languages that are decided by log-space bounded, reach-unambiguous machines.

ReachUL is a natural and interesting class. Even though the definition of ReachUL is a "syntactic" one, unlike most of the syntactic classes ReachUL has a complete problem and is closed under complement [80]. In addition, ReachUL is characterized by a directed reachability problem that has a deterministic algorithm that beats Savitch's $\log^2 n$ space bound [3].

It is natural to consider a relaxation of unambiguity where we allow a limited number of accepting computations as opposed to a unique one. FewL is the class of problems decided by nondeterministic machines with the condition that on any input there are at most a polynomial number of accepting computations. Thus FewL generalizes the class UL in a natural way. The natural extension of the the ReachUL is the class ReachFewL – the class of problems decided by nondeterministic machines that has at most a polynomial number of paths from the start configurations to *any* configuration. Various notions of unambiguity and fewness continue to be of interest to researchers [27, 25, 6, 26, 1, 106].

In this chapter we show that ReachFewL is same as ReachUL. That is, fewness does not add power to reach-unambiguity.

Theorem 13.1 (Main Theorem)**.** $\mathsf{ReachFewL} = \mathsf{ReachUL}$

This theorem improves a recent upper bound that $\mathsf{ReachFewL} \subseteq \mathsf{UL}$ shown in [106].

We combine several existing techniques to prove our results. Section 13.1 describes a multi-stage graph transformation which is used in Section 13.2 to prove the collapse.

13.1 Necessary Lemmas

We begin by defining graph properties which characterize the configuration graphs of these unambiguous computations.

Definition 13.2. Let G be a graph and s, t be vertices of G. The graph G is *path-unique* with respect to s and t if there is at most one path from s to t in G. G is *reach-unique* with respect to s if for all vertices $x \in V(G)$, there is at most one path from u to x.

These graphical definitions correspond to the necessary properties of the configuration graphs for UL and $\mathsf{ReachUL}$ machines, respectively.

13.1.1 Oracle Machines

We begin by showing that it suffices to give a log-space algorithm with $\mathsf{ReachUL}$ queries in order to give containment in $\mathsf{ReachUL}$.

Lemma 13.3. $\mathsf{L}^{\mathsf{ReachUL}} = \mathsf{ReachUL}$

Proof. The containment $\mathsf{ReachUL} \subseteq \mathsf{L}^{\mathsf{ReachUL}}$ is immediate. We proceed by describing a log-space reduction from any language in $\mathsf{L}^{\mathsf{ReachUL}}$ to a reachability problem on a reach-unique graph, by expanding the configuration graphs of the log-space machine and the oracle queries into a single configuration graph.

Let M be a log-space Turing machine with access to a $\mathsf{ReachUL}$ oracle O. Since $\mathsf{ReachUL}$ is closed under complement, we can assume that the oracle O has three types of terminating configurations: accept, reject, and halt, where there are unique accept and reject configurations. Moreover, the configuration graph for O on input $\mathbf{y}$ is reach-unique with respect to the initial configuration.

Let $\mathcal{C}_{\mathbf{x}}$ be the set of configurations for the machine M on input $\mathbf{x}$. Each configuration $C \in \mathcal{C}_{\mathbf{x}}$ requires $O(\log|\mathbf{x}|)$ bits to describe. If this configuration makes a query to the oracle O, then there is an implicit input $\mathbf{y}$ which is log-space computable given C. This gives a set $\mathcal{D}_{\mathbf{y}}$ of configurations of the oracle O on input $\mathbf{y}$.

Let $G_{M,O,\mathbf{x}}$ be the expanded configuration graph on vertices of two types. The first type of vertex is a configuration C in $\mathcal{C}_{\mathbf{x}}$. The second type of vertex is a configuration pair (C, D), where C is a query-type configuration of M in $\mathcal{C}_{\mathbf{x}}$ with implicit input $\mathbf{y}$ and D is a configuration of O in $\mathcal{D}_{\mathbf{y}}$. This set of vertices is log-space enumerable.

Edges correspond to the four types of transitions.

The first type transitions between configurations C_1, C_2 in $\mathcal{C}_{\mathbf{x}}$ during the execution of the machine M without querying O. Since this computation is deterministic, there is a unique outgoing edge at C_1.

The second type transitions from a configuration $C \in \mathcal{C}_{\mathbf{x}}$ of the machine M which queries O to the initial configuration $D_0 \in \mathcal{D}_{\mathbf{y}}$ of O on the implicit input $\mathbf{y}$. This gives a single edge leaving C, given by $C \to (C, D_0)$.

The third type is given by a query configuration $C \in \mathcal{C}_{\mathbf{x}}$ and a transition between configurations $D_1, D_2 \in \mathcal{D}_{\mathbf{y}}$ of the machine O within a query from M at the configuration C. This transition is represented by an edge from (C, D_1) to (C, D_2). Since O is a non-deterministic machine, there could be several edges from (C, D_1) to other configurations (C, D') for $D' \in \mathcal{D}_{\mathbf{y}}$.

The fourth type is given by a query configuration $C \in \mathcal{C}_{\mathbf{x}}$ of M and an accepting or rejecting configuration $D \in \mathcal{D}_{\mathbf{y}}$ of O. Depending on the accepting or rejecting status of D, the machine M responds to the query by transitioning to a configuration $C' \in \mathcal{C}_{\mathbf{x}}$. This is represented by an edge (C, D) to C'.

Claim 13.4. *This configuration graph is reach-unique with respect to the initial configuration* $C_0 \in \mathcal{C}_{\mathbf{x}}$.

Since O is a ReachUL oracle, for each query-type configuration $C \in \mathcal{C}_{\mathbf{x}}$, the subgraph G_C^O induced by the nodes (C, D) for all $D \in \mathcal{D}_{\mathbf{y}}$ is reach-unique with respect to (C, D_0) where D_0 is the initial configuration of O on $\mathbf{y}$. If D_1 is the accepting configuration and D_2 is the rejecting configuration, there is exactly one path from (C, D_0) to either (C, D_1) or (C, D_2) but not both.

Suppose there is a vertex v in $G_{M,O,\mathbf{x}}$ with two paths from the initial configuration C_0 to v. These paths must enter at least one oracle query, since the machine M is otherwise deterministic and would only give one path. Also, these paths must leave at least one oracle query, since a single query cannot give multiple paths due to the reach-uniqueness of the graphs G_C^O. However, only one of the accepting/rejecting configurations of G_C^O is reachable from the initial configuration of G_C^O. Hence, each path from C_0 to v must use the same sub-paths within every G_C^O visited. This implies that these two paths are the same. Therefore, the full configuration graph is guaranteed to be reach-unique.

Reachability in this graph can be solved in $\mathsf{ReachUL}$, showing $\mathsf{L}^{\mathsf{ReachUL}} \subseteq \mathsf{ReachUL}$.

$\square$

13.1.2 Converting from Few Graphs to Distance Isolated Graphs

A crucial lemma allows a conversion from a graph with polynomially-bounded paths to a distance isolated graph.

Definition 13.5. G is *distance isolated* with respect to a vertex u if for every vertex $v \in V(G)$ and number $d \in \{1, \dots, n\}$ there is at most one path of length d from u to v.

We use a hashing result (Theorem 13.6) due to Fredman, Komlós and Szemerédi to make the given graph distance isolated.

Theorem 13.6 (Fredman, Komlós and Szemerédi [45]). *Let c be a constant and S be a set of n-bit integers with $|S| \leq n^c$. Then there is a c' and a $c' \log n$-bit prime number p so that for any $x \neq y \in S$ $x \not\equiv y \pmod{p}$.*

The next lemma follows easily from Theorem 13.6.

Lemma 13.7. *Let* G *be a graph with edges* $E(G) = \{e_1, e_2, \ldots, e_\ell\}$. *If* G *has at most* n^k *paths from* u *to any vertex* $v \in V(G)$, *then there is a prime* $p \leq n^{k'}$, *for some constant* k', *such that the weight function* $w_p : E(G) \to \{1, \ldots, p\}$ *given by* $w_p(e_i) = 2^i \pmod{p}$ *defines a weighted graph* G_{w_p} *which is distance isolated with respect to* u.

The graph G_{w_p} in Lemma 13.7 can be converted to an unweighted, distance isolated graph by replacing an edge having weight ℓ by a path of length ℓ.

13.1.3 Converting Distance Isolated Graphs to Unique Graphs

Given a distance isolated graph, we can form a reach-unique graph by applying a layering transformation.

Definition 13.8. Let G be a directed graph on n vertices. The *layered graph* $L(G)$ induced by G is the graph on vertices $V(G) \times \{0, 1, \ldots, n\}$ and for all edges xy G and $i \in \{0, 1, \ldots, n-1\}$, the edge $(x, i) \to (y, i+1)$ is in $L(G)$.

Lemma 13.9. *If* G *is acyclic and distance isolated with respect to a vertex* u, *then* $L(G)$ *is reach-unique with respect to* $(u, 0)$, *and there is a path of length* d *from* u *to* v *in* G *if and only if there is a path from* $(u, 0)$ *to* (v, d) *in* $L(G)$.

Proof. Since all edges in $L(G)$ pass between consecutive layers, paths of length d from u to v in G are in bijective correspondence with paths from $(u, 0)$ to (v, d) in $L(G)$. Since there exists at most one path of each length from u to any vertex v in G, there exists at most one path from $(u, 0)$ to any other vertex (v, d) in $L(G)$. □

13.2 ReachFewL = ReachUL

We have sufficient tools to prove our main theorem.

Theorem 13.10. ReachFewL $\subseteq$ ReachUL.

Proof. Let L be a language in ReachFewL. There is a non-deterministic log-space machine M that decides L and a constant k so that the configuration graph G of M on an input $\mathbf{x}$ has at most n^k paths from the initial configuration u to any other configuration. We will produce a ReachUL algorithm for solving the corresponding reachability problem on this graph from the initial configuration u to the accepting configuration v.

The algorithm ReachFewSearch(G, u, v) given in Algorithm 13.1 is a log-space algorithm with queries to two ReachUL-algorithms: the algorithm IsReachUnique(H) decides if H is a reach-unique graph, and the algorithm ReachUnique(H, s, t) decides if there is a path from s to t in a reach-unique graph H.

Algorithm 13.1 ReachFewSearch(G, u, v)

```
Input: $G$ has at most $n^k$ paths between any pair of vertices.
Output: Accepts if and only if there is a path from $u$ to $v$ in $G$.
  for all primes $p \in \{1, \dots, n^{k'}\}$ do
    Define $w_p(e_i) = 2^i \pmod{p}$.
    Construct $G_{w_p}$.
    Construct $L(G_{w_p})$.
    if IsReachUnique($L(G_{w_p})$) then
      for each $d \in \{1, \dots, n(G_{w_p})\}$ do
        if ReachUnique($L(G_{w_p}), (u, 0), (v, d)$) then
          return True
        end if
      end for
      return False
    end if
  end for
  return False
```

By Lemma 13.7, there exists a prime $p \in \{1, \ldots, n^{k'}\}$ so that G_{w_p} is distance isolated. If G_{w_p} is distance isolated, then $L(G_{w_p})$ is reach-unique by Lemma 13.9. The ReachUL-algorithm IsReachUnique can detect if $L(G_{w_p})$ is in fact reach-unique.

Once it is determined that $L(G_{w_p})$ is reach-unique, the ReachUL-algorithm ReachUnique can determine if there is a path from $(u, 0)$ to (v, d) for each length d. If there is a path from u to v in G, there exists some distance $d \in \{1, \ldots, n(G_{w_p})\}$ so that there is a path from $(u, 0)$ to (v, d) in $L(G_{w_p})$

This algorithm is a log-space algorithm with ReachUL queries, giving the inclusion $\mathsf{ReachFewL} \subseteq \mathsf{L}^{\mathsf{ReachUL}}$, which equals ReachUL by Lemma 13.3. □

The complexity class ReachLFew defined by log-space machines with access to a ReachFewL oracle has been investigated before [106]. This class also collapses into ReachUL, since the ReachFewL oracle can be replaced by an ReachUL oracle, and $\mathsf{L}^{\mathsf{ReachUL}} = \mathsf{ReachUL}$.

Corollary 13.11. $\mathsf{ReachLFew} = \mathsf{ReachUL}$.

13.3 Discussion

A natural extension of our results would be to show that $\mathsf{FewL} = \mathsf{UL}$. Previous work of Reinhardt and Allender [111] provides a UL algorithm for graphs where there is a unique path of minimum length from the source to *any other vertex*. Given the configuration graph G of a FewL computation, there exists a prime $p \leq n^{k'}$ that makes G_{w_p} distance isolated with respect to the initial configuration and terminating configurations, which makes G_{w_p} have a unique minimum length path with respect to this pair of configurations. The problem is that the UL algorithm for reachability requires the graph to have a unique minimum length path from the source to any other vertex. If this reach-type restriction could be replaced with a path-type restriction, then the collapse $\mathsf{FewL} = \mathsf{UL}$ would follow. This would be a step towards solving the $\mathsf{NL} = \mathsf{UL}$ problem.

Chapter 14

Reachability in Surface-Embedded Acyclic Graphs

Graph reachability problems are central to space-bounded algorithms. Different versions of this problem characterize several important space complexity classes. The problem of deciding whether there is a path from a node u to v in a directed graph is the canonical complete problem for non-deterministic log-space (NL). A recent breakthrough result of Reingold [109] provides a deterministic log-space algorithm for reachability in undirected graphs. It is also known that some restricted promise versions of the directed reachability problem characterize randomized log-space computations (RL) [110]. We aim to improve upper bounds on the space required to solve the reachability problem in surface-embedded digraphs.

Prior Results

Savitch's $O(\log^2 n)$ space bound for the directed reachability problem [117], Saks and Zhou's $O(\log^{3/2} n)$ bound for reachability problems characterizing RL computations [116], and Reingold's log-space algorithm for the undirected reachability problem [109] are the three most significant results in this topic. Clearly, designing an algorithm for general reachability problem that beats Savitch's bound is one of the most important open questions in this area. While this appears to be a difficult problem, investigating classes of directed graphs for which we can design space efficient algorithms is an important research direction. Recently, there has been progress reported along this theme. Jakoby, Liśkiewicz, and Reischuk [68] and Jakoby and Tantau [69] show that various reachability and optimization questions for *series-parallel* graphs admit deterministic log-space algorithms. Series-parallel graphs are a very restricted subclass of planar DAGs. In particular, such graphs have a single source and a single sink. Allender, Barrington, Chakraborty, Datta, and Roy [2] extended the result of Jakoby *et al.* to show that the reachability problem for Single-source Multiple-sink Planar DAGs (SMPDs) can be decided in log-space. Building on this work, in [126], the authors show reachability can be decided in log-space for planar DAGs with $O(\log n)$ sources. Theorem 14.1 below is implicit in [126].

Theorem 14.1 (Stolee, Bourke, Vinodchandran [126])**.** *Let $\mathcal{G}(m)$ be the set of planar DAGs with at most $m = m(n)$ sources. The reachability problem over $\mathcal{G}(m)$ can be solved by a log-space machine using a non-deterministic certificate with $O(m)$ bits. This yields deterministic space bound of either (1) $O(\log n + m)$ or (2) $O(\log n \cdot \log m)$ for reachability over $\mathcal{G}(m)$.*

The $O(\log n + m)$ space bound is by a brute-force search over all certificates of length $O(m)$. Setting $m = O(\log n)$ gives a log-space algorithm. The $O(\log n \cdot \log m)$ bound is obtained by first converting the nondeterministic algorithm to a layered graph with $\mathrm{p}\,oly(n)$ vertices and m layers and then applying Savitch's algoritm on this layered graph. The second bound leads to a deterministic algorithm that beats Savitch's bound for reachability over DAGS with $2^{o(\log n)}$ sources. However, if we are aiming for deterministic *l*og-space algorithms, the above theorem could not handle asymptotically more than $\log n$ sources. In this chapter we extend the number of sources to $2^{O(\sqrt{\log n})}$ while maintaining log-space computability. We also extend our results to graphs embedded on higher genus surfaces. In addition, techniques of this chapter also leads to new results on simultaneous time-space bounds for reachability which are not implied by [126].

Investigating tree-width restricted graphs has also resulted in new space-efficient algorithms. Elberfeld, Jakoby, and Tantau present a log-space algorithm for reachability (and other problems) over graphs with constant tree-width [39]. Another interesting class of reachability problems for which we know an algorithm that beats Savitch's bound is the class of *reach-unique* problems – digraphs G with vertex pair u, v where there is at most one path from u to any other vertex. Allender and Lange showed that reachability in reach-unique graphs can be solved in deterministic $O(\log^2 n / \log\log n)$ space [3]. Recently, reach-unique reachability algorithms were shown to be strong enough to also compute *reach-poly* reachability problems, where there are at most a polynomial number of paths from u to any other vertex [48].

Designing algorithms for reachability with simultaneous time and space bound is another important direction that has been of considerable interest in the past. Since a depth first search can be implemented in linear time and $O(n)$ space, the goal here is to improve the space bound while maintaining a polynomial running time. The most significant result here is Nisan's $O(\log^2 n)$ space, $n^{O(1)}$ time bound for RL [99]. The best upper bound for general directed reachability is the $O(n/2^{\sqrt{\log n}})$ space, $n^{O(1)}$ time algorithm due to Barnes, Buss, Ruzzo and Schieber [12].

Our Results

We consider the reachability problem over a large class of DAGs embedded on surfaces. Since the graph is acyclic, there exist vertices with no incoming edge, called *sources*. Define $n = n(G)$ to be the number of vertices in the input graph. Let $\mathcal{G}(m, g)$ denote the class of DAGs with at most $m = m(n)$ source vertices embedded on a surface (orientable or non-orientable) of genus at most $g = g(n)$. Our main technical contribution is the following log-space reduction that *c*ompresses an instance of reachability for such surface-embedded DAGs.

Theorem 14.2. *There is a log-space reduction that given an instance $\langle G, u, v \rangle$ where $G \in \mathcal{G}(m, g)$ and u, v vertices of G, outputs an instance $\langle G', u', v' \rangle$ where G is a directed graph, u' and v' are vertices of G', and*

(a) there is a path from u to v in G if and only if there is a path from u' to v' in G',

(b) G' has $O(m + g)$ vertices.

By a direct application of Savitch's theorem on the reduced instance we get the following result.

Theorem 14.3. *The reachability problem for graphs in $\mathcal{G}(m, g)$ can be decided in deterministic $O(\log n + \log^2(m + g))$ space.*

Compare Theorem 14.3 to Theorem 14.1 to see that the space bound has improved from $O(\log n + m)$ or $O(\log n \cdot \log m)$ to $O(\log n + \log^2 m)$. By setting $m = g = 2^{O(\sqrt{\log n})}$ we get a deterministic log-space algorithm for reachability in $\mathcal{G}(2^{O(\sqrt{\log n})}, 2^{O(\sqrt{\log n})})$.

Corollary 14.4. *The reachability problem for directed acyclic graphs with $2^{O(\sqrt{\log n})}$ sources embedded on surfaces of genus $2^{O(\sqrt{\log n})}$ can be decided in deterministic logarithmic space.*

A more relaxed setting of parameters leads to deterministic algorithms that asymptotically beat the Savitch's bound of $O(\log^2 n)$. By setting m and g to be $n^{o(1)}$ we get the following.

Corollary 14.5. *The reachability problem for directed acyclic graphs embedded on surfaces with sub-polynomial genus and with sub-polynomial number of sources can be decided in deterministic space* $o(\log^2 n)$*.*

Combining our reduction with a simple depth-first search gives us better simultaneous time-space bound for reachability over a large class of graphs than known before.

Theorem 14.6. *The reachability problem for graphs in* $\mathcal{G}(m, g)$ *can be decided in polynomial time using* $O(\log n + m + g)$ *space.*

Note that Theorem 14.6 has a space bound which matches to $O(\log n + m)$ space bound of Theorem 14.1, except it guarantees polynomial time, where the previous bound gave $2^{O(m)}\,\mathrm{poly}(n)$ running time. In particular, for any $\epsilon < 1$, we get a polynomial time algorithm for reachability over graphs in $\mathcal{G}(O(n^\epsilon), O(n^\epsilon))$ that uses $O(n^\epsilon)$ space. This beats the Barnes *et al.* upper bound of polynomial time and $O(n/2^{\sqrt{\log n}})$ space for this class of graphs.

Corollary 14.7. *For any* ϵ *with* $0 < \epsilon < 1$*, the reachability problem for graphs in* $\mathcal{G}(O(n^\epsilon), O(n^\epsilon))$ *can be decided in polynomial time using* $O(n^\epsilon)$ *space.*

We note that the upper bound on space given in Theorem 14.6 can be slightly improved to $O\left((m+g)2^{-\sqrt{\log(m+g)}}\right)$ by using Barnes *et al.* algorithm instead of depth-first search, which will give a $o(n^\epsilon)$ space bound in the above corollary.

Theorem 14.8. *The reachability problem for graphs in* $\mathcal{G}(m, g)$ *can be decided in deterministic polynomial time using* $O\left(\log n + \frac{m+g}{2^{\sqrt{\log(m+g)}}}\right)$ *space.*

14.0.1 Outline

Theorem 14.2 is proven in several parts. We begin in Section 14.1 by reviewing some concepts of topological embeddings including log-space algorithms on embedded graphs. In Section 14.2, we present a simple structural decomposition called the *forest decomposition* of the given directed acyclic graph. Based on this decomposition, we classify the edges as local and global. We present log-space algorithms of Allender, Barrington, Chakraborty, Datta, and Roy [2] to decide reachability using local edges. In order to control how the global edges interact, we define the notion of *t*opological equivalence among global edges in Section 14.3. We show that the number of possible equivalence classes is bounded by $O(m+g)$. Then, Section 14.4 describes a finite list of *patterns* that characterize how paths use edges in these equivalence classes. We also analyze the structure of these patterns. In particular, for each pattern type we identify a pair of log-space computable edges in the corresponding equivalence class that has certain canonical properties. In Section 14.5, we describe a graph on $O(m+g)$ vertices called the *pattern graph* whose vertices are described by patterns on equivalence classes. The edges in the pattern graph are defined by a very restricted reachability condition between equivalence classes. We finally show that this pattern graph is computable in log-space and preserves reachability between a given pair of vertices.

Before we begin, we note that throughout this chapter certain known log-space primitives are frequently used as subroutines without explicit reference to them. In particular, Reingold's log-space algorithm for undirected reachability is often used, for example to identify connected components in certain undirected graphs.

14.0.2 Notation

We mainly deal with directed graphs. A directed edge $e = xy$ has the direction from x to y and we call x the *tail* denoted by $\mathrm{Tail}(e)$, and y the *head* denoted by $\mathrm{Head}(e)$.

We assume that the given graph is acyclic. Lemma 14.9 gives a technique for converting a source-bounded reachability algorithm on graphs promised to be acyclic into a cycle-detection algorithm without asymptotically increasing the space requirement.

Lemma 14.9. *Let $s(n, m, g) = \Omega(\log n)$. If there exists an $O(s(n, m, g))$-space bounded algorithm for testing uv-reachability over graphs in $\mathcal{G}(m, g)$ then there exists an $O(s(n, m, g))$-space bounded algorithm to test if a graph is acyclic, given that it has at most m sources and is embedded in a surface of genus at most g.*

Proof. Let $A(G, u, v)$ be the algorithm for testing uv-reachability on $G \in \mathcal{G}(m, g)$. Fix an incoming edge at each non-source vertex, making a set $F \subseteq E(G)$. By taking reverse walks from each vertex, it can be verified that F has no cycles.

Order the edges $E(G)$ as $\{e_1, \ldots, e_{|E(G)|}\}$. For each $i \in \{0, 1, \ldots, |E(G)|\}$, let G_i be the subgraph of G where an edge e_j is present in G_i if $e_j \in F$ or $j \leq i$. Iterate through all such i and test if $A(G_i, \mathrm{Head}(e_{i+1}), \mathrm{Tail}(e_{i+1}))$ ever returns with success. If any returns True, then there is a cycle including the edge e_{i+1}. Note that A gives the correct response, since G_0 was cycle free and by iteration, G_i is cycle free. Each G_i is acyclic for $i \in \{1, \ldots, |E(G)|\}$ if and only if G is acyclic and all queries $A(G_i, \mathrm{Head}(e_{i+1}), \mathrm{Tail}(e_{i+1}))$ return False. □

14.1 Topological Embeddings and Algorithms

We assume that the input graph G is embedded on a surface S where every face is homeomorphic to an open disk. Such embeddings are called *2-cell embeddings*. We assume that such an embedding is presented as a *combinatorial embedding* where for each vertex v the circular ordering of the edges incident to v is specified. In the case of a non-orientable surface, the signature of an edge is also given, specifying if the orientation of the rotation switches across this edge. Since computing or approximating a low-genus embedding of a non-planar graph is an NP-complete problem [136, 28], we require the embedding to be given as part of the input and we consider reachability in $\mathcal{G}(m, g)$ to be a promise problem. In the case of genus zero, we can compute a planar embedding in log-space and the promise condition can be removed.

Let G be a graph with n vertices and e edges embedded on a surface S with f faces, then by the well known *Euler's Formula* we have $n - e + f = \chi_S$, where χ_S is the Euler characteristic of the surface S. The number of faces in a graph is log-space computable from a combinatorial embedding (for a proof, see [77]), so χ_S is also computable in log-space. The genus g_S of the surface S is given by the equation $\chi_S = 2 - 2g_S$ for orientable surfaces and $\chi_S = 2 - g_S$ for non-orientable surfaces.

Let C be a simple closed curve on S given by a cycle in the underlying undirected graph of G. C is called *surface separating* if the removal of C disconnects S. A surface separating curve C is called *contractible* if removal of the nodes in C disconnects G where at least one of the connected components has an induced embedding homeomorphic to a disc.

In order to perform log-space algorithms on curves in the graph, we must be

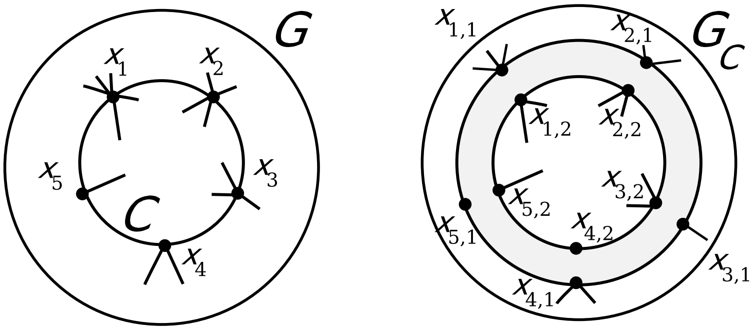


Figure 14.1: Splitting G at a curve C.

able to represent these curves in log-space. A curve C is *log-space walkable* if there is a log-space algorithm which outputs the edges of C in order. Examples of such curves are given in the following section. Given a log-space walkable curve C, it is possible to detect the type (separating, contractible, or neither) of C in log-space.

First, note that if C is not orientable (i.e. there are an odd number of negatively-signed edges in C) then C cannot be separating or contractible. By first checking the parity of such edges, we can assume that C is orientable.

Given an orientable curve $C = x_1x_2 \dots x_k$ (indices taken modulo k), we can create (in log-space) an auxilliary graph G_C where each vertex x_i is copied to two vertices $x_{i,1}, x_{i,2}$ with edges x_ix_{i+1} copied to two edges $x_{i,1}x_{i+1,1}$ and $x_{i,2}x_{i+1,2}$. However, an edge from a vertex y in $V(G) \setminus C$ to a vertex x_i in C maps to one of two edges:

1. yx_i maps to $yx_{i,1}$ if yx_i appears between $x_{i-1}x_i$ and x_ix_{i+1} in the clockwise order about x_i.
2. yx_i maps to $yx_{i,1}$ if yx_i appears between x_ix_{i+1} and $x_{i-1}x_i$ in the clockwise order about x_i.

There is a natural combinatorial embedding of G_C induced from the embedding of G by using the same cyclic relations for vertices $y \in V(G) \setminus C$ and for split vertices

$x_{i,1}$ and $x_{i,2}$, use the orientation of x_i but skip the edges which are not incident to the new vertex. See Figure 14.1 for an example of such a split. The following properties are simple to prove:

1. C is separable if and only if G_C is disconnected. In this case, G_C has two components.
2. C is contractible if and only if G_C is disconnected and at least one of the components is embedded with characteristic zero.

Moreover, using Reingold's undirected reachability algorithm we can detect that C is separable. Given a vertex $y \notin C$, we can also detect which connected component of G_C contains y. We shall exploit both of these properties in the following two sections as we partition the edge set using topological information.

14.2 Forest Decomposition

A simple structural decomposition, called a *f*orest decomposition, of a directed acyclic graph forms the basis of our algorithm. This forest decomposition has been utilized in previous works [2, 126].

Let G be a directed acyclic graph and let u, v be two vertices. Our goal is to decide whether there is a directed path from u to v. Let $u, s_1, \ldots, s_m$ be the sources of G. If u is not a source, make it a source by removing all the incoming edges. This will not affect uv-reachability, increases the number of sources by at most one, and only reduces the genus of the embedding.

Definition 14.10 (Forest Decomposition)**.** Let A be a deterministic log-space algorithm that on input of a non-source vertex x, outputs an incoming edge yx (for example, selecting the lexicographically-first vertex y so that yx is an edge in G).

This algorithm defines a set of edges $F_A = \{yx : x \in V(G) \setminus \{u, v, s_1, \ldots, s_m\},\ y = A(x)\}$, called a *forest decomposition* of G.

Since G is acyclic, the reverse walk $x_1, x_2, \ldots$, where $x_1 = x$ and $x_{i+1} = A(x_i)$, must terminate at a source s_j, u, or v, so the edges in F_A form a forest subgraph. For the purposes of the forest decomposition, v is treated as a source since no incoming edge is selected. If a vertex x is in the tree with source v, then all non-tree edges entering x are deleted. This will not affect uv-reachability, since G is acyclic and does not increase the number of sources or the genus of the surface. Each connected component in F_A is a tree rooted at a source vertex, called a *source tree*. The forest forms a typical *ancestor* and *descendant* relationship within each tree. For the remainder of this work, we fix an acyclic graph $G \in \mathcal{G}(m, g)$ embedded on a surface S (defined by the combinatorial embedding) and $F = F_A$ a log-space computable forest decomposition.

Definition 14.11 (Tree Curves)**.** Let x and y be two vertices in some source tree T of F. The *tree curve* at xy is the curve on S formed by the unique undirected path in T from x to y. If xy is an edge, then the closed curve formed by xy and the tree curve at xy is called the *closed tree curve* at xy.

Definition 14.12 (Local and Global Edges)**.** Given an S-embedded graph G and a forest decomposition F, an edge xy in $E(G) \setminus F$ is classified as *local*[1] if (a) x and y are on the same tree in F, (b) the closed tree curve at xy is contractible (i.e. the curve cuts S into a disk and another surface), and (c) No sources lie on the interior of the surface which is homeomorphic to a disk. If S is the sphere, then the curve cuts S into two disks and xy is local if one of the disks contains no source in the interior. Otherwise, the edge xy is *global*.

---

[1] This definition of *local* differs from the use in [2] and [126].

14.2.1 Paths within a single tree

Definition 14.13 (Region of a tree)**.** Let T be a connected component in the forest decomposition F along with the local edges between vertices in T. The *region* of T, denoted $\mathcal{R}[T]$ is the portion of the surface S given by the faces enclosed by the tree and local edges in T.

The faces that compose $\mathcal{R}[T]$ are together homeomorphic to a disk, since $\mathcal{R}[T]$ can contract to the source vertex by contracting the disks given by the local edges into the tree, and then contracting the tree into the source vertex. This disk is oriented using the combinatorial embedding at the source by the right-hand rule. Reachability in such subgraphs T can be decided using the SMPD algorithm [2], in log-space. Note that the restriction of a 2-cell embedding implies all global edges are incident to vertices on the outer curve of the disk $\mathcal{R}[T]$. Our figures depict source trees as circles, with the source placed in the center, with tree edges spanning radially away from the source[2]. We can also assign a clockwise or counter-clockwise direction to all local edges in a source tree region $\mathcal{R}[T_{s_j}]$.

Definition 14.14 (Rotational Direction within $\mathcal{R}[T]$)**.** For a local edge xy, the closed tree curve at xy is cyclicly oriented by the direction of xy. The edge xy is considered clockwise (counter-clockwise) if this cyclic orientation is clockwise (counter-clockwise) with respect to the orientation of $\mathcal{R}[T]$.

Definition 14.15 (Irreducible Path)**.** A path $P = x_1 x_2 \dots x_k$ in G is *F-irreducible* if for each $i < j$ so that x_i is an F-ancestor of x_j, then $x_i x_{i+1} \dots x_{j-1} x_j$ is the path in F from x_i to x_j. We say P is *irreducible* when the forest decomposition F is implied from context.

---

[2] This visualization of source trees was crucial to the development of this work, and is due to [2].

Lemma 14.16. *If there is a path from x to y in G, there is an F-irreducible path from x to y.*

Proof. Replace the violating subpaths with the given tree paths. $\square$

A very useful property of irreducible paths is that they travel in a single rotational direction within each source tree.

Lemma 14.17. *Let P be an irreducible local path from x to y in a source tree T, where y is on the boundary of $\mathcal{R}[T]$. There is a unique direction (clockwise or counter-clockwise) so that all non-tree edges of P follow this direction.*

Proof. Let e be the first local edge in P. Without loss of generality, we assume it takes a clockwise orientation. Assume for the sake of contradiction there exists a local edge in P that takes a counterclockwise orientation. Let f be the first such edge. Consider how P travels from the head of e to reach the tail of f. Note that all non-tree edges in this path have a clockwise orientation. This gives three cases:

Case 1: P passes through the ancestor path of $\text{Head}(f)$ at a vertex a. In this case, P is not irreducible, since f is not a tree edge and an irreducible path would take the tree edges from a to $\text{Head}(f)$.

Case 2: P passes through the descendants of $\text{Head}(f)$ at a vertex b. In this case, following P from a to $\text{Head}(f)$ then the tree path from $\text{Head}(f)$ to a creates a cycle, contradicting that G is a DAG.

Case 3: P travels around the descendants of $\text{Head}(f)$ using a local edge e'. Now, $\text{Head}(f)$ is properly contained within the tree cycle given by e'. In order for P to reach y on the boundary of $\mathcal{R}[T]$, P must cross this curve. This must cross the descendants of $\text{Tail}(e')$ or $\text{Head}(e')$, creating a cycle, contradicting that G is acyclic.

Therefore, such an f does not exist and all edges take the same orientation. $\square$

14.2.2 Reachability within a single tree

We now focus on the reachability problem within a single tree T_{s_j}. By the definition of local edges, we have the subgraph given by local edges within a single tree is a single-source multiple-sink planar DAG. Allender *et al.* [2] solved the reachability problem in this class of graphs. We review their method as well as adapt the method to test directional reachability.

Definition 14.18 (Step and Jump Edges)**.** A local edge $e \notin F$ is a *jump* edge if the tree curve C_e partitions $V(G) \setminus C_e$ into two non-trivial parts. Otherwise, e is a *step* edge.

First, we discuss how to solve reachability when restricted to tree and step edges.

Theorem 14.19 (Allender *et al.* [2])**.** *Let s_j be a source in G. Reachability within $\mathcal{R}[T_{s_j}]$ using tree and step edges is log-space computable.*

Proof. Here, we consider the subgraph in $\mathcal{R}[T_{s_j}]$ given by the tree and step edges to be a planar graph with a single source. Since we have removed the jump edges in $\mathcal{R}[T_{s_j}]$, all sinks in this graph are on the boundary of $\mathcal{R}[T_{s_j}]$. By adding a new global sink t to the outer face, the graph $\mathcal{R}[T_{s_j}] + t$ becomes a Single-source Single-sink Planar DAG (SSPD).

The cyclic orientation of edges at each vertex must have the outgoing edges and incoming edges in two consecutive blocks. If not, suppose that the edges e_1, e_2, e_3, e_4 appear in clockwise order at a vertex x, with e_1, e_3 are outgoing edges and e_2, e_4 are incoming edges. Since there is a single source s_j, there are paths P_2 and P_4 from s_j to x using the edges e_2 and e_4, respectively. Likewise, there are paths P_1 and P_3 from x to t starting with edges e_1, and e_3, respectively. This gives two closed curves C_1 (composed of P_1 and P_3) and C_2 (composed of P_2 and P_4) which cross at x. Thus, they must cross at another point y. By following C_1 from x to y and C_2 from y to x, there is a cycle in G, a contradiction.

Given that the outgoing edges at any vertex x are in a single block of the cyclic orientation, we can define the notion of *left-most* and *right-most* outgoing edges of x as those appearing as the first and last (respectively) outgoing edges of the block with respect to the clockwise ordering. This defines a *left-most walk* and a *right-most walk* from a vertex x by following the left-most and right-most edges, starting at x and terminating at t. The left-most and right-most walks define a closed curve C_x that includes x and t.

A vertex y is inside this curve C_x if and only if it is reachable from x: if y is within C_x, any path from s_j to y must cross the curve C_x, creating a path from x to y, and if y is reachable from x via a path P, the edges of P must appear between the left-most and right-most walks from x. Hence, by splitting $\mathcal{R}[T_{s_j}] + t$ along C_x and computing if y is within C_x, we can detect reachability. $\square$

Using the step-reachability algorithm as a subroutine, we now discuss directional reachability using all local edges.

Theorem 14.20 (Allender *et al.* [2]). *Given vertices x, y on the boundary of $\mathcal{R}[T_{s_j}]$ and a direction d (left or right), reachability from x to y in $\mathcal{R}[T_{s_j}]$ using local edges using an irreducible path in direction d is log-space computable.*

Proof. We shall define a log-space data structure called an *explored region* which in turn defines a set of vertices in $\mathcal{R}[T]$. The crucial property of these vertices is that all jump edges with tail in the set and head outside the set are reachable from x. We will then use these edges to modify the explored region while maintaining this property. When complete, the explored region will contain y if and only if y is reachable from x via an irreducible path with rotational direction d, with respect to the orientation of the source s_j.

We shall assume that the direction d is Right (clockwise). The other direction follows by symmetry.

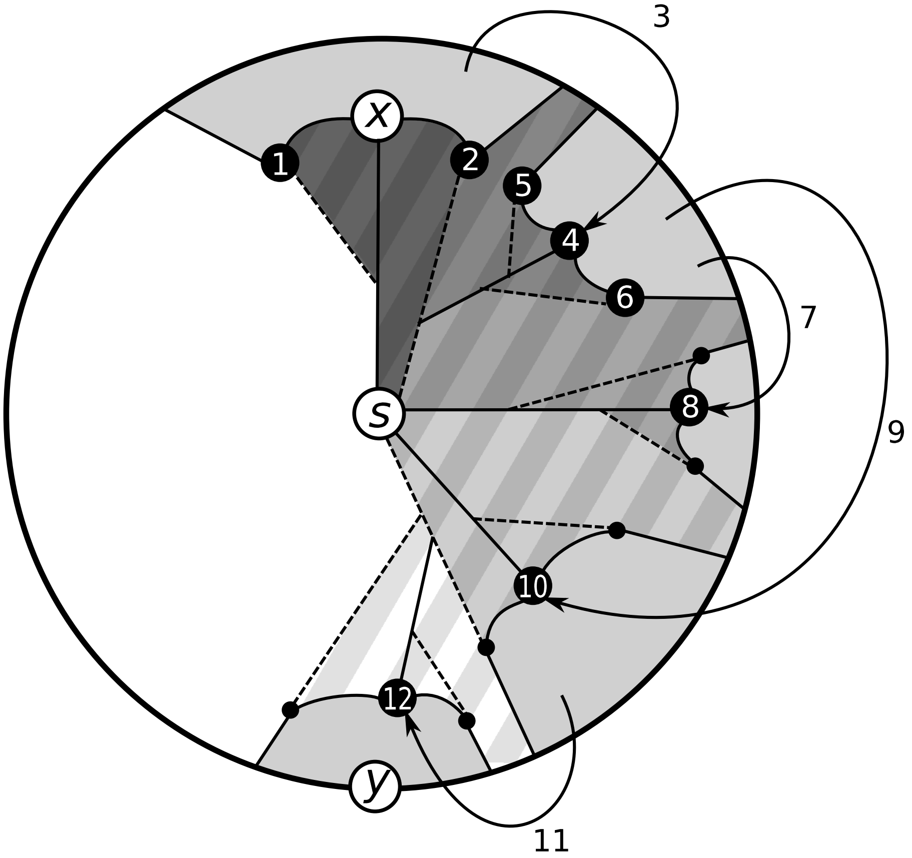
Given a vertex w in T_{s_j}, define $\text{ReachStep}(w)$ to be the vertices in T_{s_j}, reachable from w by tree and step edges. Define functions $\text{StepLeft}(w)$ and $\text{StepRight}(w)$ to be the vertices within $\text{ReachStep}(w)$ which appear most counter-clockwise and clockwise, respectively, breaking ties by selecting vertices closer to the source s_j along T.

We shall define two log-size variables ReachLeft and ReachRight and initialize them as $\text{StepLeft}(x)$ and $\text{StepRight}(x)$. These two variables store enough information for the explored region. The vertex set $\text{Between}(\text{ReachLeft}, \text{ReachRight})$ is defined as the vertices which are strictly between ReachLeft and ReachRight in the clockwise order of T_{s_j} and the descendants of ReachLeft and ReachRight. Note that this does *not* include the ancestors of ReachLeft and ReachRight.

Of particular interest to the explored region are jump edges with tail in the explored region $\text{Between}(\text{ReachLeft}, \text{ReachRight})$ and head *not* in the explored region. We call these edges *exiting* edges. Note that a jump edge e is exiting if and only if the tree curve at e contains ReachRight.

Since each d-directional exiting edge contains ReachRight, the exiting edges form a linear order $e_1, e_2, \ldots, e_r$ where e_i is contained within the tree curve on e_j if and only if $i < j$. We shall extend the explored region by using the minimal exiting edge, denoted e_{jump}, and setting ReachRight to $\text{StepRight}(\text{Head}(e_{\text{jump}}))$.

Proceed to extend the explored region until one of two situations arise: if the vertex y is within $\text{ReachStep}(\text{Head}(e_{\text{jump}}))$, we return True; if there are no exiting edges, stop and return False. This process is detailed in Algorithm 14.1.



1. StepLeft(x)
2. StepRight(x)
3. $e_{\text{jump}}^{(1)}$
4. Head$(e_{\text{jump}}^{(1)})$
5. StepLeft(Head$(e_{\text{jump}}^{(1)})$)
6. StepRight(Head$(e_{\text{jump}}^{(1)})$)
7. $e_{\text{jump}}^{(2)}$
8. Head$(e_{\text{jump}}^{(2)})$
9. $e_{\text{jump}}^{(3)}$
10. Head$(e_{\text{jump}}^{(3)})$
11. $e_{\text{jump}}^{(4)}$
12. Head$(e_{\text{jump}}^{(4)})$

The shaded region is the explored region. The flat gray areas are reachable while the striped areas are not. The striped area is darker depending on how many iterations that region was in the explored region.

Figure 14.2: An example execution of ReachLocal(x, y, R).

Algorithm 14.1 ReachLocal(x, y, d) — Returns True if and only if y reachable from x

```
ReachLeft $\leftarrow$ StepLeft$(x)$
ReachRight $\leftarrow$ StepRight$(x)$
$i \leftarrow 1$
while there exists a $d$-directional exiting edge do
    $e^{(i)}_{\text{jump}} \leftarrow$ the minimal $d$-directional exiting edge
    if $y \in$ ReachStep(Head$(e^{(i)}_{\text{jump}})$) then
        return True
    else if $d =$ Right then
        ReachRight $\leftarrow$ StepRight(Head$(e^{(i)}_{\text{jump}})$)
    else if $d =$ Left then
        ReachLeft $\leftarrow$ StepLeft(Head$(e^{(i)}_{\text{jump}})$)
    end if
    $i \leftarrow i + 1$
end while
return False
```

The correctness of ReachLocal(x, y, d) requires the following claim regarding the explored region.

Claim 14.21. *At every stage of Algorithm 14.1, every exiting edge e has* Tail(e) *reachable from x using a d-directional irreducible path.*

Proof of Claim. Without loss of generality, we assume $d = \text{R}$. We proceed by induction on the number of iterations in the execution of ReachLocal(x, y, d). When ReachLeft and ReachRight are initialized, the explored region consists of vertices within ReachStep(x) and vertices strictly within the curve given by concatenating the following paths:

$$s_j \xrightarrow{T} \text{ReachLeft} \overset{\text{(local)}}{\longrightarrow} x \overset{\text{(local)}}{\longrightarrow} \text{ReachRight} \xrightarrow{T} s_j.$$

If a jump edge e has tail within the explored region, then either (1) it is within ReachStep(x) and is reachable, or (2) it is bound by the curve and must not be an exiting edge. Thus, the claim holds for the first iteration.

Assume the claim holds for the kth iteration. Consider the next iteration's selection of e_{jump} and let e be a jump edge with tail within the new explored region. If the tail of e is in the previous explored region, the induction step shows the claim holds. Otherwise, there are only two cases. First, the tail of e is within $\text{ReachStep}(\text{Head}(e_{\text{jump}}))$ and e is reachable since e_{jump} was reachable by induction. Second, the tail of e is strictly within the curve given by concatenating the following paths:

$$s_j \xrightarrow{T} \text{Tail}(e_{\text{jump}}) \xrightarrow{e_{\text{jump}}} \text{Head}(e_{\text{jump}}) \xrightarrow{(local)} \text{ReachRight} \xrightarrow{T} s_j,$$

and hence the edge e is not exiting. This proves the claim. □

Given the above claim, observe that when ReachLocal(x, y, d) returns True it is correct, as there is some subset of the e_{jump} edges which can be combined with local paths to create a path from x to y.

To finish, we must prove that if there is a d-directional irreducible path from x to y in $\mathcal{R}[T_{s_j}]$, then ReachLocal(x, y, d) returns True. Fix a path from x to y that uses the minimum number jump edges and consider the sequence $e_1, \ldots, e_t$ of jump edges within this path. The minimum number of jump edges guarantees that $\text{Tail}(e_i) \in \text{ReachStep}(\text{Head}(e_{i-1}))$ and $\text{Tail}(e_{i+1}) \notin \text{ReachStep}(\text{Head}(e_{i-1}))$ for all suitable $i \in \{2, \ldots, t-1\}$. The first jump edge e_1 is an exiting edge for the first explored region.

We claim that at each iteration where y is not in $\text{ReachStep}(\text{Head}(e_{\text{jump}}))$, there is an edge e_i of the path that is an exiting edge. This is given by the choice of e_{jump} as the mimimal d-directional exiting edge. In the previous iteration, there was some e_i that was exiting. If e_i was selected as e_{jump}, then $\text{Tail}(e_{i+1})$ is within

ReachStep(e_{jump}) and Head(e_{i+1}) is not. Since all jump edges are d-directional, the edge e_{i+1} is an exiting edge and the claim holds for another iteration.

Suppose that e_{jump} was not selected to be e_i. Then, the tree curve at e_{jump} is contained within the tree curve at e_i. This provides two cases:

1. Head$(e_i) \notin$ ReachStep(Head(e_{jump})) and e_i is still an exiting edge, or
2. Head$(e_i) \in$ ReachStep(Head(e_{jump})) and hence Tail(e_{i+1}) is in ReachStep(Head(e_{jump})).

In the latter case it is not immediate that e_{i+1} is an exiting edge, but some edge $e_{i'}$ with $i' > i$ will be an exiting edge, since y is not in ReachStep(Head(e_{jump})). $\square$

14.3 Topological Equivalence

The following notion of topological equivalence plays a central role in our algorithms. It was originally presented in [126] for planar graphs, but we extend it to arbitrary surfaces.

Definition 14.22 (Topological Equivalence)**.** Let G be a graph embedded on a surface S. Let F be a forest decomposition of G. We say two (undirected) global edges xy and wz are *topologically equivalent* if the following two conditions are satisfied: (a) They span the same source trees in F (assume x and w are on the same tree), (b) The closed curve in the underlying undirected graph formed by (1) the edge xy, (2) the tree curve from y to z, (3) the edge zw, and (4) the tree curve from w to x bounds a connected portion of S, denoted $D(xy, wz)$, that is homeomorphic to a disk and no source lies within $D(xy, wz)$.

Topological equivalence is an equivalence relation. For the sake of the reflexive property, we take as convention that a single edge is topologically equivalent to itself. The symmetry of the definition is immediate. Transitivity is implied by the following lemma, which is immediate from the definitions.

Lemma 14.23. *Let* e_1, e_2 *be topologically equivalent global edges and* e_3 *a global edge.*

1. *If* e_3 *has an endpoint in* $D(e_1, e_2)$*, then* e_3 *is equivalent to both* e_1 *and* e_2*.*
2. *If* e_3 *is equivalent to* e_2*, then one of the following cases holds:*
    - *a)* e_1 *is in* $D(e_2, e_3)$*.*
    - *b)* $D(e_1, e_2)$ *and* $D(e_2, e_3)$ *intersect at the curve given by* e_2 *and the ancestor paths from its endpoints to their respective sources, and* $D(e_1, e_3) = D(e_1, e_2) \cup D(e_2, e_3)$*.*

   In both cases (a) and (b), e_1 *is topologically equivalent to* e_3*.*

Let E be an equivalence class of global edges containing an edge e, where e spans two different source trees. Consider the subgraph of G given by the vertices in the source trees containing the endpoints of e, along with all local edges in those trees and the edges in E. This subgraph is embedded in a disk on S, as given in the following corollary.

Corollary 14.24. *Given an equivalence class E of global edges, let* $S_E = \bigcup_{e_1, e_2 \in E} D(e_1, e_2)$*. The surface* S_E *is a disk.*

Proof. Lemma 14.23, implies that for any triple $e_1, e_2, e_3 \in E$ and any pair of the disks $D(e_1, e_2)$, $D(e_1, e_3)$, and $D(e_2, e_3)$ are either adjacent or have a containment relationship. There is an ordering $e_1, \ldots, e_k$ of the edges of E so that the disks $D(e_i, e_{i+1})$ pairwise intersect only at boundaries. Gluing the disks $D(e_{i-1}, e_i)$ and $D(e_i, e_{i+1})$ along e_i constructs S_E as a disk. □

We shall make explicit use of this locally-planar embedding. For an equivalence class of global edges spanning vertices in the same tree, a similar subgraph and embedding is formed by considering the ends of the equivalence class to be different copies of that source tree.

The lexicographically-least edge e in a topological equivalence class of global edges is log-space computable. By counting how many global edges which are lexicographically smaller than e and are the lexicographically-least in their equivalence classes, the equivalence class containing e is assigned an index i. The class E_i is the ith equivalence class in this ordering. We shall use this notation to label the equivalence classes.

Definition 14.25 (The Region of an Equivalence Class)**.** Let E_i be an equivalence class of global edges. Define the *region enclosed by* E_i as $\mathcal{R}[E_i] = \bigcup_{e_1, e_2 \in E_i} D(e_1, e_2)$.

The region $\mathcal{R}[E_i]$ has some properties which are quickly identified. There are two edges $e_a, e_b \in E_i$ so that $\mathcal{R}[E_i] = D(e_a, e_b)$. These outer edges define the *sides* of $\mathcal{R}[E_i]$. The *boundary* of $\mathcal{R}[E_i]$ is given by these two edges and their ancestor paths in F on all four endpoints. All vertices in a source tree T are contained in the region $\mathcal{R}[T]$. Let T_A and T_B be the two source trees containing the tail and head, respectively, of the representative edge in E_i. The vertices within the boundary of $\mathcal{R}[E_i]$ are within $\mathcal{R}[T_A]$ and $\mathcal{R}[T_B]$. The vertices in $\mathcal{R}[E_i]$ are partitioned into two *ends*, A and B, where the vertices are placed in an end determined by containment in $\mathcal{R}[T_A] \cap \mathcal{R}[E_i]$ and $\mathcal{R}[T_B] \cap \mathcal{R}[E_i]$ when the trees T_A and T_B different or by the two connected components of $\mathcal{R}[T_A] \cap \mathcal{R}[E_i]$ when the trees T_A and T_B are equal. Note that the endpoints of edges in E_i lie on the boundary of the regions $\mathcal{R}[T_A]$ and $\mathcal{R}[T_B]$. There is an ordering $e_a = e_1, e_2, \ldots, e_k = e_b$ of E_i so that the endpoints of the e_j on the A-end appear in a clockwise order in that tree. Two regions $\mathcal{R}[E_i]$ and $\mathcal{R}[E_j]$ on different classes E_i and E_j intersect only on the boundary paths. The vertices on the boundary are not considered *inside* the region, since they may be in multiple regions.

Since global edges appear on the boundary of $\mathcal{R}[T]$ for a given source tree T, there is a natural clockwise ordering on these edges, with respect to the orientation of T. Further, we can order the incident equivalence classes (with possibly a single repetition, in the case of global edges with both endpoints in T) by the clockwise order the ends $\mathcal{R}[E_i] \cap \mathcal{R}[T]$ appear on the boundary of $\mathcal{R}[T]$.

The resource bounds we prove directly depends on the number of equivalence classes. The following lemma bounds the number of equivalence classes.

Lemma 14.26. *Let G be a graph embedded on a surface S with Euler characteristic χ_S with a forest decomposition F with m sources. There are at most $3(m + |\chi_S|)$ topological equivalence classes of global edges. If g_S is the genus of S, $|\chi_S| = O(g_S)$ and there are $O(m + g_S)$ equivalence classes of global edges.*

Proof. Consider a graph G which has a maximal number of equivalence classes and remove all but one representative of each class. Create a new multigraph H on the m sources with edges given by the representatives of each class, with the edges embedded in S by following the undirected path composed of the tree path from the first source to the edge, the edge, then the tree path from the edge to the second source. There are m vertices, and let e be the number of edges, f the number of faces. Subdivide these edges twice to get a simple graph embedded in S. Note that Euler's formula holds in this graph on $m + 2e$ vertices, $3e$ edges, and f faces. Hence,

$$\begin{aligned} \chi_S &= (m + 2e) - (3e) + f \\ &= m - e + f. \end{aligned}$$

Moreover, each face must have at least three equivalence classes, and each edge is incident to two faces, so $2e \leq 3f$ and $f \leq \frac{2}{3}e$. This gives

$$\chi_S = m - e + f \leq m - \frac{1}{3}e$$
$$\Rightarrow e \leq 3m - 3\chi_S \leq 3(m + |\chi_S|). \qquad \square$$

Now that all tree and local edges are embedded in disks of the form $\mathcal{R}[T]$ and global edges are in $O(m+g)$ disks of the form $\mathcal{R}[E_i]$, we are able to abandon all other portions of S. The important information from S is that the ends of regions incident to a given source tree appear in a clockwise order on the boundary of $\mathcal{R}[T]$ and that there are $O(m+g)$ equivalence classes of global edges. Each source tree looks like a disk ($\mathcal{R}[T]$) with strips ($\mathcal{R}[E_i]$ for incident classes E_i) stretching radially away from it (as long as the other end of the strip $\mathcal{R}[E_i]$ is not considered). Hence, the regions $\mathcal{R}[T_{s_j}]$ and $\mathcal{R}[E_i]$ form a ribbon graph, which encodes the entire surface but has only m vertices and $O(m+g)$ edges.

Consider an equivalence class E_i between source trees T_A and T_B, a rotational direction d (clockwise or counterclockwise), and a vertex x in T_A outside the region $\mathcal{R}[E_i]$. We say that the vertex x *fully reaches* E_i in the direction d if there is an irreducible d-directional local path from x to an endpoint of each edge in E_i. If x does not fully reach E_i in direction d, but there is a local path from x to an endpoint of some edge of E_i, then we say x *partially reaches* E_i in this direction. If such a path is irreducible, then the path follows a clockwise or counter-clockwise direction within T_A and we say x fully (or partially) reaches E_i *using a clockwise (or counter-clockwise) rotation*.

Lemma 14.27. *Let x be a vertex in a source tree T_A. For each rotational direction (clockwise or counter-clockwise), there is an ordering $E_{i_0}, E_{i_1}, \ldots, E_{i_\ell}$ of the edge classes reachable* *via irreducible paths in that direction so that*

1. *x fully reaches each E_{i_j} for $j \in \{1, \dots, \ell - 1\}$.*
2. *x either fully or partially reaches E_{i_0} and E_{i_ℓ}.*
3. *If x is not in the interior of $\mathcal{R}[E_{i_0}]$, x fully reaches E_{i_0}.*

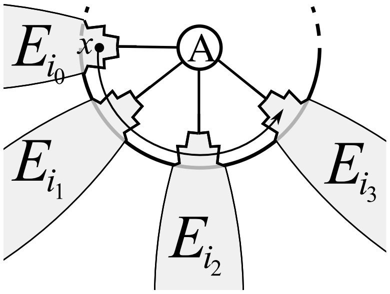


Figure 14.3: A vertex x with three counter-clockwise reachable classes, E_{i_1}, E_{i_2}, and E_{i_3}, as in Lemma 14.27.

Proof. Construct the list using all reachable classes in the given rotational direction and order by their appearance. The irreducible path P from x to the class E_{i_ℓ} must intersect the tree paths from the source to the edges in each class E_{i_j} for all $j < \ell$, with $x \notin \mathcal{R}[E_{i_j}]$, since the edges in P lie in $\mathcal{R}[T]$, but the endpoints of the edges in E_{i_j} are on the boundary of $\mathcal{R}[T]$. Hence, x fully reaches these classes. $\square$

14.4 Global Patterns

At this point, we take a very different approach than [126]. The algorithm described in [126] focused on reachability within the regions $\mathcal{R}[T]$ on the source trees T. Here, we focus on reachability within and between equivalence classes E_i. We create a constant number of vertices derived from each equivalence class. This constant is given by the number of distinct ways a path can enter the region $\mathcal{R}[E_i]$, use edges in E_i, then leave the region $\mathcal{R}[E_i]$. We call these *patterns*.

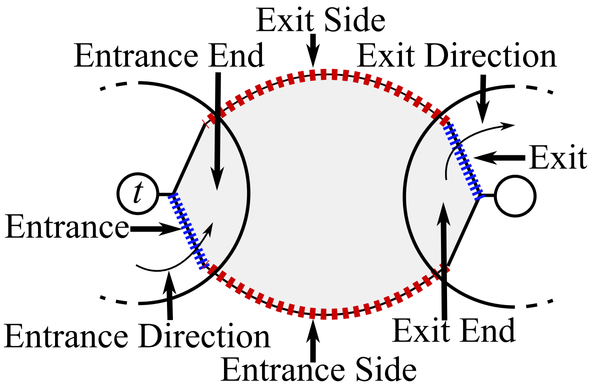


Figure 14.4: Terminology for the entrance and exit of a pattern and the modifiers of *direction*, *end*, and *side*. This example is an LXR pattern.

Definition 14.28 (The Pattern Set)**.** Let E_i be an equivalence class of global edges. An irreducible path P that involves an edge of the class E_i *induces* a pattern on E_i defined by abc with $a, c \in \{\text{L}, \text{R}\}$, $b \in \{\text{S}, \text{X}\}$ where a is the clockwise (R) or counter-clockwise (L) direction the path takes as it enters $\mathcal{R}[E_i]$, c is the direction the path takes as it leaves $\mathcal{R}[E_i]$, and if $b = \text{S}$, the path enters and leaves $\mathcal{R}[E_i]$ on the same end and if $b = \text{X}$, the path enters and leaves $\mathcal{R}[E_i]$ on opposite ends. [3]. Define the *pattern set*, $\mathcal{P} = \{\text{RSR}, \text{LSL}, \text{RXR}, \text{RXL}, \text{LXR}, \text{LXL}\}$.

Let E_i be an edge class and $\mathcal{R}[E_i]$ be the enclosed region. Let t be an end of $\mathcal{R}[E_i]$ (either A or B) and fix an orientation on that end and a pattern p that involves E_i. Then the *entrance* (*exit*) of the pattern at the t-end is the ancestor path on the boundary of $\mathcal{R}[E_i]$ on the t-end that a path must cross *b*efore (respectively, *a*fter) using the edges in E_i that induce the pattern p with the given orientation. (See Figure 14.4 for a visual representation of the entrance and exit of a pattern.)

We can now define *p*attern descriptions which are the vertices of the pattern graph that we will define in the next section.

---

[3] The interested reader will find the notation for patterns derived from move sequences in the Coin Crawl Game of [126].

Definition 14.29 (Pattern Descriptions)**.** Let k be the number of topological equivalence classes of edges of G. A *pattern description* is a tuple $\mathbf{x} = (i, t, o, p)$ where $i \in \{1, \ldots, k\}$, $t \in \{A, B\}$, $o \in \{+1, -1\}$, and $p \in \mathcal{P}$. Here i represents the equivalence class E_i, t represents the end of $\mathcal{R}[E_i]$ that contains the entrance, $o \in \{+1, -1\}$ specifies if the orientation of the path is in agreement with (or opposite to, respectively) the local orientation of the tree on the t-side of E_i, and $p \in \mathcal{P}$ represents the pattern used in E_i. The set $\{1, \ldots, k\} \times \{A, B\} \times \{+1, -1\} \times \mathcal{P}$ of all pattern descriptions is denoted by V_{P}.

For example, the description $(i, B, +1, \mathrm{RXL})$ is an element in V_{P} corresponding to a RXL pattern, using at least one edge of the class E_i starting at the B-side and leaving the A-side, oriented to agree with the B-side. Lemma 14.26 implies the number of descriptions is $O(m + g_S)$ where m is the number of sources and g the genus of the surface. A pattern description can be represented with $\lceil \log k \rceil + 5 = O(\log(m + g_S))$ bits[4].

We now investigate some properties of paths that induce these pattern descriptions. We focus on a path which uses local edges and global edges in a single equivalence class and induces a single pattern on that class. These single-pattern paths will be concatenated to make larger paths once the structure of the shorter paths is understood.

An important property of these patterns is that if the pattern is of full type or the equivalence class is fully reachable, we can assume without loss of generality that the path used two special edges, which we call the *canonical edge pair*.

Definition 14.30 (Canonical Edge Pair)**.** Let $\mathbf{x} = (i, t, o, p)$ be a pattern description centered at the edge class E_i. There are two edges (*incoming* and *outgoing*) in E_i,

---

[4] This bland fact is in fact very important for the later use of Savitch's Theorem.

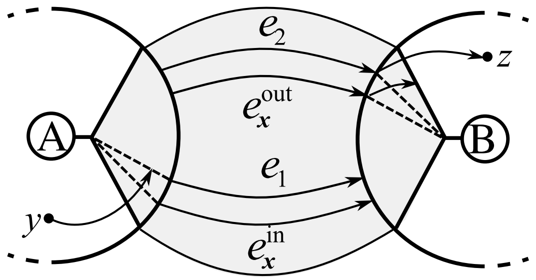


Figure 14.5: The edges used in the proof of Lemma 14.31 in an LXR pattern.

called the *canonical edge pair* for $\mathbf{x}$. The *outgoing edge*, $e_{\mathbf{x}}^{\text{out}}$, is the edge $e \in E_i$ with head on the exit end that is farthest from the exit side so that there exists a local path from $\text{Head}(e)$ to the exit of $\mathcal{R}[E_i]$. The *incoming edge*, $e_{\mathbf{x}}^{\text{in}}$, is the edge $e \in E_i$ with the tail on the entrance end that is closest to the entrance side so that either $e = e_{\mathbf{x}}^{\text{out}}$ or $\text{Tail}(e_{\mathbf{x}}^{\text{out}})$ is reachable from $\text{Head}(e)$ using local paths and edges in E_i.

14.4.1 Full Patterns

Full patterns are named so because a path which induces a full pattern intersects the ancestor path of at least one endpoint of every edge in the class. Hence, every edge is reachable. This leads to the property that if an irreducible path induces such a pattern, then the path might as well use the canonical edges in the corresponding equivalence class.

Lemma 14.31. *Let* $\mathbf{x}$ *be a pattern description of full type centered at an edge class* E_i*. Let* $y, z \in V(G)$ *be vertices not inside* $\mathcal{R}[E_i]$*, where* y *is in the source tree on the entrance end of* $\mathbf{x}$ *and* z *is in the source tree on the exit end of* $\mathbf{x}$*. Then there is a path from* y *to* z *in* G *using only local paths and edges of the class* E_i *that induces the pattern* $\mathbf{x}$ *if and only if* $\text{Tail}(e_{\mathbf{x}}^{\text{in}})$ *is reachable from* y *using a local path in the entrance direction of* $\mathbf{x}$ *and* z *is reachable from* $\text{Head}(e_{\mathbf{x}}^{\text{out}})$ *using a local path in the exit direction of* $\mathbf{x}$*.*

Proof. Note that if the tail of $e_{\mathbf{x}}^{\text{in}}$ is reachable from y using a local path in the entrance direction, and z is reachable from the head of $e_{\mathbf{x}}^{\text{out}}$ using a local path in the exit direction, then there is a path from y to z that induces the pattern $\mathbf{x}$ using the path between $e_{\mathbf{x}}^{\text{in}}$ and $e_{\mathbf{x}}^{\text{out}}$ given by the definition of the canonical pair.

If a path exists from y to z that induces the pattern $\mathbf{x}$, then there is at least one edge of the class E_i in the path. Let e_1 be the first edge of class E_i used in the path and e_2 be the last. Consider where e_1 and e_2 are in comparison to the canonical pair $(e_{\mathbf{x}}^{\text{in}}, e_{\mathbf{x}}^{\text{out}})$ in the ordering of the edges in E_i. An example of the edges e_1 and e_2 are shown in Figure 14.5.

If e_1 is closer to the entrance side of E_i compared to $e_{\mathbf{x}}^{\text{in}}$, then (by the definition of $e_{\mathbf{x}}^{\text{in}}$) there is no path from the head of e_1 to the tail of $e_{\mathbf{x}}^{\text{out}}$ using local paths and edges in E_i. Hence, a path from e_1 that leaves $\mathcal{R}[E_i]$ in the exit direction can not cross the ancestor path of the tail of $e_{\mathbf{x}}^{\text{out}}$, so it must cross the ancestor path of the head of $e_{\mathbf{x}}^{\text{out}}$. This implies there is an edge e in E_i in the direction of $e_{\mathbf{x}}^{\text{out}}$ that is farther from the exit direction and whose head reaches the head of $e_{\mathbf{x}}^{\text{out}}$. This contradicts the definition of $e_{\mathbf{x}}^{\text{out}}$, since there is now a local path from the head of e_1 that reaches the boundary of $\mathcal{R}[E_i]$ in the exit direction.

Therefore, the edge e_1 appears after $e_{\mathbf{x}}^{\text{in}}$ in the order on E_i starting from the entrance side. This implies that y has a local path that crosses the ancestor path from the tail of $e_{\mathbf{x}}^{\text{in}}$ and hence reaches the tail of $e_{\mathbf{x}}^{\text{in}}$. If $e_{\mathbf{x}}^{\text{out}}$ is on the exit side of E_i compared to e_2, then by the definition of $e_{\mathbf{x}}^{\text{out}}$, there is no local path from the head of e_2 that reaches the boundary of $\mathcal{R}[E_i]$ in the exit direction. So, e_2 is on the exit side of E_i compared to $e_{\mathbf{x}}^{\text{out}}$. The local path that reaches the boundary of $\mathcal{R}[E_i]$ from the head of $e_{\mathbf{x}}^{\text{out}}$ crosses the ancestor path to the head of e_2, so z is reachable from the head of $e_{\mathbf{x}}^{\text{out}}$ using a local path. □

Lemma 14.32. *Let* $\mathbf{x}$ *be a pattern description of full type. The canonical edge pair* $(e_{\mathbf{x}}^{\text{in}}, e_{\mathbf{x}}^{\text{out}})$ *is log-space computable.*

Proof. The outgoing edge, $e_{\mathbf{x}}^{\text{out}}$, is computed by enumerating the set of edges in the class E_i with head on the exit end of $\mathcal{R}[E_i]$ which reach the boundary of the region $\mathcal{R}[E_i]$ using local edges in the exit direction of the pattern.

The incoming edge is computed by an iterative procedure. Store two edge pointers, e_1 and e_2. These edges will always be in the class E_i or null. The edge e_1 will have tail in the entrance end of $\mathcal{R}[E_i]$ and e_2 with have tail in the exit end of $\mathcal{R}[E_i]$. Initialize $e_1 = e_{\mathbf{x}}^{\text{out}}$ and set e_2 to be null.

Proceed by iterating through the edges in E_i starting at $e_{\mathbf{x}}^{\text{out}}$ to the last edge in E_i on the entrance side of $\mathcal{R}[E_i]$. Each edge is a candidate to update e_1 and e_2.

If the tail is in the entrance side of $\mathcal{R}[E_i]$, check if the head reaches the tail of e_2 or $e_{\mathbf{x}}^{\text{out}}$ using a local path. If so, then update e_1 to this edge.

If the tail is in the exit side of $\mathcal{R}[E_i]$, check if the head reaches the tail of e_1 or $e_{\mathbf{x}}^{\text{out}}$ using a local path. If so, then update e_2 to this edge.

After all edges have been tested, set $e_{\mathbf{x}}^{\text{in}} = e_1$. There is a path from e_1 to $e_{\mathbf{x}}^{\text{out}}$ using local paths and edges in E_i by considering the reverse sequence of e_1 and e_2 updates that allowed $\text{Tail}(e_{\mathbf{x}}^{\text{out}})$ to be reachable from $\text{Head}(e_1)$. Further, no edge beyond e_1 in the proper direction can reach $e_{\mathbf{x}}^{\text{out}}$ because it must cross the ancestor paths from e_1 to the sources on each endpoint. $\square$

14.4.2 Nesting Patterns

Nesting patterns are named so because irreducible paths which induce such patterns use exactly one edge of this class, and we may assume that the edge used is the one farthest from the entrance that is reachable (and that a local path exists from its head to the exit). The following lemmas describe properties of nesting patterns.

Lemma 14.33. *If an irreducible path using local paths and edges in a global edge class E_i induces a nesting pattern, then the path uses exactly one edge in the class E_i.*

Proof. Let x and y be vertices outside E_i with a path from x to y that induces a nesting pattern on E_i. Let e_1 be the first edge in E_i used and e_2 be the second. Note that e_2 cannot be closer to the entrance direction than e_1, or else the head of e_2 is a descendant of the local path from x to the tail of e_1, contradicting irreducibility. Also, e_2 cannot be farther from the entrance direction than e_1 or else the path from the head of e_2 to y must cross the ancestor path at the head of e_1, creating a cycle, contradicting that the graph is acyclic. □

Lemma 14.34. *Let $\mathbf{x}$ be a pattern description of nesting type centered at a global edge class E_i. Then, $e_{\mathbf{x}}^{\text{in}} = e_{\mathbf{x}}^{\text{out}}$, and $e_{\mathbf{x}}^{\text{out}}$ is log-space computable.*

Proof. By the definition of $e_{\mathbf{x}}^{\text{out}}$, there is a local path P from the head of $e_{\mathbf{x}}^{\text{out}}$ to the boundary of $\mathcal{R}[E_i]$ in the exit direction (which is also the entrance direction). All edges in E_i closer to the boundary in the entrance direction from $e_{\mathbf{x}}^{\text{out}}$ have at least one endpoint reachable from P. If any of these edges could reach $e_{\mathbf{x}}^{\text{out}}$, then there would be a cycle. Therefore, $e_{\mathbf{x}}^{\text{in}} = e_{\mathbf{x}}^{\text{out}}$.

Iterate through the edges in E_i starting on the exit side. Then, $e_{\mathbf{x}}^{\text{out}}$ is the last edge in this order with a local path from the head to the boundary of $\mathcal{R}[E_i]$ in the exit direction. □

Lemma 14.35. *Let $\mathbf{x}$ be a nesting pattern centered at an edge class E_i. Let y and z be vertices not inside $\mathcal{R}[E_i]$. If there exists an irreducible path from y to z using local paths and edges in the global edge class E_i which induces $\mathbf{x}$, then z is reachable from* Head $(e_{\mathbf{x}}^{\text{out}})$.

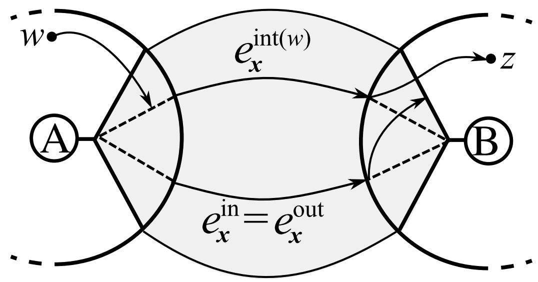


Figure 14.6: The most-interior edge from a vertex w in a pattern description $\mathbf{x}$ with an RXR pattern.

While it would be useful to have a property similar to Lemma 14.31 for nesting patterns, there may exist a vertex w from which there are paths that induce a nesting pattern without reaching the canonical incoming edge. We can define a new edge in the class that is similarly canonical, except with respect to the vertex w.

Definition 14.36 (Most-Interior Edge)**.** Let $\mathbf{x} = (i, t, o, p)$ be a pattern description of nesting type and w be a vertex not in the interior of $\mathcal{R}[E_i]$. The *most-interior* edge of $\mathbf{x}$ reachable from w, denoted $e_{\mathbf{x}}^{\mathrm{int}(w)}$, is the edge e in the class E_i that is farthest from the entrance side of $\mathcal{R}[E_i]$ so that (a) there is a local path from w to $\mathrm{Tail}(e)$ in the entrance direction, and (b) there is a local path from $\mathrm{Head}(e)$ to the exit boundary of $\mathcal{R}[E_i]$.

Lemma 14.37. *Let* $\mathbf{x}$ *be a pattern description of nesting type and* w *a vertex not in the interior of* $\mathcal{R}[E_i]$*. The most-interior edge,* $e_{\mathbf{x}}^{\mathrm{int}(w)}$*, is log-space computable. For any vertex* z *not in* $\mathcal{R}[E_i]$*, there is a path from* w *to* z *that induces the pattern* $\mathbf{x}$ *if and only if there is an irreducible local path from* $\mathrm{Head}\left(e_{\mathbf{x}}^{\mathrm{int}(w)}\right)$ *to* z *in the exit direction of* $\mathbf{x}$*. If* w *fully reaches* E_i*, then* $e_{\mathbf{x}}^{\mathrm{int}\,w} = e_{\mathbf{x}}^{\mathrm{out}}$*.*

Proof. The edges in the class E_i have an order using the rotation given by the entrance direction of the pattern description $\mathbf{x}$, where two edges in E_i can be compared using this order in log-space. Let $e_{\mathbf{x}}^{\mathrm{int}(w)}$ be the edge e of class E_i farthest from the entrance side of $\mathcal{R}[E_i]$ with tail reachable from w and the head has a local path reaching the exit boundary of $\mathcal{R}[E_i]$ in the exit direction of $\mathbf{x}$. Note that this edge is computable in log-space using the SMPD algorithm and pairwise comparison of the rotational order of edges.

Consider an irreducible path P from w that induces the pattern description $\mathbf{x}$ to reach a vertex z outside $\mathcal{R}[E_i]$. By Lemma 14.33, the path P uses exactly one edge e of the class E_i. The edge cannot farther from the entrance side of $\mathcal{R}[E_i]$ than $e_{\mathbf{x}}^{\mathrm{int}(w)}$ or else either w does not reach $\mathrm{Tail}(e)$ or $\mathrm{Head}(e)$ does not reach the exit of $\mathcal{R}[E_i]$. The path that exits the class E_i from the head of $e_{\mathbf{x}}^{\mathrm{int}(w)}$ must pass through the tree path from the source to the head of e. Therefore, the head of e is reachable from the head of $e_{\mathbf{x}}^{\mathrm{int}(w)}$ and so is anything reachable from the head of e, including z.

Since $\mathrm{Tail}(e_{\mathbf{x}}^{\mathrm{int}(w)})$ is reachable from w using a local path in the entrance direction, anything reachable from $\mathrm{Head}(e_{\mathbf{x}}^{\mathrm{int}(w)})$ using a local path in the exit direction is reachable from w using a path that induces the pattern description $\mathbf{x}$. □

14.5 The Pattern Graph

We now describe a graph on $O(m + g_S)$ vertices that preserves uv-reachability.

Definition 14.38 (The Pattern Graph)**.** Given G and F as above, the *pattern graph*, denoted $\mathrm{P}(G, F) = (V'_{\mathrm{P}}, E'_{\mathrm{P}})$ is a directed graph defined as follows. The vertex set $V'_{\mathrm{P}} = \{u', v'\} \cup V_{\mathrm{P}} = \{u', v'\} \cup (\{1, \ldots, k\} \times \{A, B\} \times \{+1, -1\} \times \mathcal{P})$. For two pattern descriptions $\mathbf{x}, \mathbf{y} \in V_{\mathrm{P}}$, an edge $\mathbf{x} \to \mathbf{y}$ is in E'_{P} if and only if there exists a (possibly empty) list of nesting pattern descriptions $\mathbf{z}_1, \ldots, \mathbf{z}_\ell$ (called an *adjacency certificate*), so that the following two conditions hold:

1. There is an irreducible path from $\mathrm{Head}(e_{\mathbf{x}}^{\mathrm{out}})$ to $\mathrm{Tail}(e_{\mathbf{y}}^{\mathrm{in}})$ which induces the

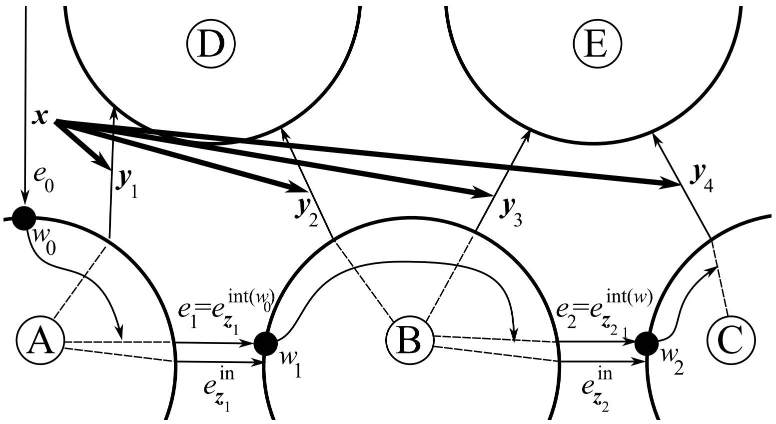


Figure 14.7: The nesting patterns $\mathbf{z}_1$ and $\mathbf{z}_2$ satisfy the adjacency conditions in Definition 14.38 from $\mathbf{x}$ to each $\mathbf{y}_j$. The pattern adjacencies are enumerated during the algorithm of Lemma 14.40 where e is assigned to e_0, e_1, and e_2, sequentially. Note that $e_0 = e_{\mathbf{x}}^{\text{out}}$, $e_1 = e_{\mathbf{z}_1}^{\text{int}(\text{Head}(e_0))}$, and $e_2 = e_{\mathbf{z}_2}^{\text{int}(\text{Head}(e_1))}$. The pattern $\mathbf{y}_1$ is reachable from w_0 with no internal nesting patterns. The patterns $\mathbf{y}_2$ and $\mathbf{y}_3$ are reachable from w_0 using the nesting pattern $\mathbf{z}_1$. The pattern $\mathbf{y}_4$ is reachable from w_0 using the nesting patterns $\mathbf{z}_1$ and $\mathbf{z}_2$. The algorithm from Lemma 14.40 terminates at e_2, since e_2 does not give a partially-reachable class.

sequence $\mathbf{z}_1, \ldots, \mathbf{z}_\ell$ of nesting pattern descriptions.

2. For each $j \in \{1, \ldots, \ell\}$, $\text{Tail}(e_{\mathbf{z}_j}^{\text{in}})$ is not reachable from $\text{Head}(e_{\mathbf{x}}^{\text{out}})$ using irreducible paths that induce the pattern descriptions $\mathbf{z}_1, \ldots, \mathbf{z}_{j-1}$.

In addition, for a description $\mathbf{x} = (i, t, o, p)$ there is an edge $u' \to \mathbf{x}$ in E'_{P} if and only if $\mathbf{x}$ has the t-end in the tree T_u. Also, for a pattern description $\mathbf{x} = (i, t, o, p)$ there is an edge $\mathbf{x} \to v'$ in E'_{P}, if and only if the class E_i is incident to v, t is the other end of the class, and $p \in \{\mathrm{RXL}, \mathrm{LXR}\}$.

Theorem 14.39. *There exists a path from u to v in G if and only if there exists a path from u' to v' in* $\mathrm{P}(G, F)$*.*

Proof. ($\Rightarrow$) Let P be an irreducible path from u to v in G. P induces a sequence of pattern descriptions $\mathbf{x}_1, \ldots, \mathbf{x}_\ell$. Note that $\mathbf{x}_1$ is centered at an edge class that is incident to T_u and the entrance end is on T_u. Note also that $\mathbf{x}_\ell$ is centered at an edge class where the edges have head v. Thus, in $\mathrm{P}(G, F)$, $u' \to \mathbf{x}_1$ and $\mathbf{x}_\ell \to v'$ are edges.

For full pattern descriptions $\mathbf{x}_i$, Lemma 14.31 implies that we may assume the first edge in the global edge class of $\mathbf{x}_i$ used by P is $e^{\text{in}}_{\mathbf{x}_i}$ and the last such edge is $e^{\text{out}}_{\mathbf{x}_i}$.

Fix $i \in \{1, \ldots, \ell - 1\}$ and let $\mathbf{x}_j$ be the next full pattern induced after $\mathbf{x}_i$. If $j = i + 1$, then the path P takes a local path between the edges that induce the patterns $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$. By Lemma 14.31, $e^{\text{in}}_{\mathbf{x}_j}$ is reachable from $e^{\text{out}}_{\mathbf{x}_i}$ by a local path and an adjacency exists from $\mathbf{x}_i$ to $\mathbf{x}_{i+1}$ in $\mathrm{P}(G, F)$, using an empty list of nesting patterns as the adjacency certificate.

Otherwise, $j > i + 1$ and there are $j - i$ nested patterns between $\mathbf{x}_i$ and $\mathbf{x}_j$. Rename the nesting patterns between $\mathbf{x}_i$ and $\mathbf{x}_j$ as $\mathbf{z}_1, \ldots, \mathbf{z}_{j-i}$ where $\mathbf{z}_{i'} = \mathbf{x}_{i+i'}$. If $\mathbf{z}_1, \ldots, \mathbf{z}_{j-i}$ compose an adjacency certificate for $\mathbf{x}_i \to \mathbf{x}_j$, then this edge exists in $\mathrm{P}(G, F)$. Otherwise, there exists such a k that violates the adjacency condition between $\mathbf{x}_i$ and $\mathbf{x}_j$, then let i' be the smallest such index. There is an edge in $\mathrm{P}(G, F)$ from $\mathbf{x}_i$ to the nesting pattern description $\mathbf{z}_{i'}$, since $\mathrm{Tail}(e^{\text{in}}_{\mathbf{z}_{i'}})$ is reachable from $\mathrm{Head}(e^{\text{out}}_{\mathbf{x}_i})$ by a path using the nesting patterns $\mathbf{z}_1, \ldots, \mathbf{z}_{i'-1}$ as the adjacency certificate. By Lemma 14.37, $\mathrm{Tail}(e^{\text{in}}_{\mathbf{x}_j})$ is reachable from $\mathrm{Head}(e^{\text{out}}_{\mathbf{z}_{i'}})$ using an irreducible path which induces the patterns $\mathbf{z}_{i'+1}, \ldots, \mathbf{z}_{j-i}$. By iteration, there is a path from $\mathbf{z}_{i'}$ to $\mathbf{x}_j$ in $\mathrm{P}(G, F)$, and hence a path from $\mathbf{x}_i$ to $\mathbf{x}_j$ in $\mathrm{P}(G, F)$. Connecting all of the edges between the full patterns in $\mathbf{x}_1, \ldots, \mathbf{x}_\ell$ gives a path from u' to v' in $\mathrm{P}(G, F)$.

($\leftarrow$) Given a path $P = u', \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_\ell, v'$ in $\mathrm{P}(G, F)$, let $\mathbf{x}_j = (i_j, t_j, o_j, p_j)$ for each $j \in \{1, \ldots, \ell\}$. Since $u' \to \mathbf{x}_1$ in $P(G)$, E_{i_1} is a class incident to T_u and all edges are reachable from u. Specifically, there is a tree path P_0 from u to $e^{\text{out}}_{\mathbf{x}_1}$. Similarly, since $\mathbf{x}_\ell \to v'$ in $\mathrm{P}(G, F)$, E_{i_k} is a class incident to T_v and all edges have v as a head. For each $j \in \{1, \ldots, \ell - 1\}$, Lemmas 14.31 and 14.37 imply there is an irreducible path P_i in G from the head of $e^{\text{out}}_{\mathbf{x}_j}$ to the tail of $e^{\text{in}}_{\mathbf{x}_{j+1}}$ that is either a local path or induces a list of nesting pattern descriptions which form an adjacency certificate. Also, by Definition 14.30, there exist (possibly empty) paths Q_j from $e^{\text{in}}_{\mathbf{x}_j}$ to $e^{\text{out}}_{\mathbf{x}_j}$ using local paths and edges of the class E_{i_j}. These paths concatenate to a path $uP_0e^{\text{out}}_{\mathbf{x}_1}P_1e^{\text{in}}_{\mathbf{x}_2}Q_2e^{\text{out}}_{\mathbf{x}_2}P_2e^{\text{in}}_{\mathbf{x}_3}\ldots e^{\text{out}}_{\mathbf{x}_{\ell-1}}P_{\ell-1}e^{\text{in}}_{\mathbf{x}_\ell}v$ from u to v in G. □

Lemma 14.40. *The pattern graph* $\mathrm{P}(G, F)$ *is log-space computable.*

Proof. Given a pattern description $\mathbf{x}$, we describe a log-space algorithm for enumerating the pattern descriptions reachable by an edge in $\mathrm{P}(G, F)$. It is simple to find the pattern descriptions $\mathbf{x}, \mathbf{y}$ so that $u \to \mathbf{x}$ and $\mathbf{y} \to v$.

A necessary subroutine takes a global edge e and enumerates all pattern descriptions reachable from $\mathrm{Head}(e)$ using local paths in the exit direction of $\mathbf{x}$. By Lemma 14.27, there is an ordered list of topological equivalence classes $E_{i_0}, E_{i_1}, \ldots, E_{i_\ell}$ reachable by local paths from the head of e. E_{i_0} is the class containing e, so e is in $\mathcal{R}[E_{i_0}]$. All other classes E_{i_j} (for $j \geq 1$, except possibly $j = \ell$) are fully reachable. Hence, each pattern description $\mathbf{y}$ centered at a class E_{i_j} with $j \in \{1, \ldots, \ell - 1\}$ (where the entrance direction of $\mathbf{y}$, orientation, and end all match the exit direction of $\mathbf{x}$) has $e^{\text{in}}_{\mathbf{y}}$ reachable from $\mathrm{Head}(e)$ using a local path. Each pattern description $\mathbf{y}$ with entering direction the same as the exit direction of $\mathbf{x}$ and centered at E_{i_ℓ} can be checked if $e^{\text{in}}_{\mathbf{y}}$ is reachable from e. The only pattern that could be used without having $e^{\text{in}}_{\mathbf{y}}$ reachable is a nesting pattern.

To enumerate all neighbors of $\mathbf{x}$ in $\mathrm{P}(G, F)$, perform the above subroutine on $e^{\text{out}}_{\mathbf{x}}$, adding edges from $\mathbf{x}$ to each reachable pattern description $\mathbf{y}$. If the nesting pattern $\mathbf{z}$ on E_{i_ℓ} is not fully reachable (i.e. there is no local path from e to $e^{\text{in}}_{\mathbf{z}}$ in the proper direction) then compute the most-interior edge $e_{\mathbf{z}}^{\mathrm{int}(\mathrm{Head}(e))}$. Repeat the subroutine on this edge, continuing until the class E_{i_ℓ} is fully reachable (or the list is empty). In the jth iteration, let $w_{j-1} = \mathrm{Head}(e)$ and $\mathbf{z}_j = \mathbf{z}$. See Figure 14.7 for an example of this iterative procedure.

It is clear this algorithm takes log-space. It enumerates all neighbors of $\mathbf{x}$ in $\mathrm{P}(G,F)$, since a neighbor $\mathbf{y}$ requires a list of nesting classes $\mathbf{z}_1,\ldots,\mathbf{z}_\ell$ so that there is an irreducible path from $\mathbf{x}$ to $\mathbf{y}$ inducing these classes. Each class $\mathbf{z}_j$ has the edge $e_{\mathbf{z}_j}^{\mathrm{in}}$ *not* reachable from $\mathbf{x}$ using the patterns $\mathbf{z}_1,\ldots,\mathbf{z}_{j-1}$. This means that the pattern $\mathbf{z}_j$ is centered at the class E_{i_ℓ} computed by the iteration of the subroutine on the edge $e_{\mathbf{z}_{j-1}}^{\mathrm{int}(w_{j-1})}$. Moreover, $\mathbf{y}$ appears as a reachable class from the most-interior edge computed at $\mathbf{z}_\ell$, so $\mathbf{y}$ is enumerated. Finally, any pattern enumerated by this procedure can reconstruct the list of $\mathbf{z}_1,\ldots,\mathbf{z}_\ell$ by using the nesting patterns used in the subroutine iterations. $\square$

Theorem 14.41 (Main Theorem)**.** *There is a log-space reduction that given an instance* $\langle G,u,v\rangle$ *where* $G \in \mathcal{G}(m,g)$ *and* u,v *vertices of* G*, outputs an instance* $\langle G',u',v'\rangle$ *where* G *is a directed graph and* u',v' *vertices of* G'*, so that*

- *(a) there is a directed path from* u *to* v *in* G *if and only if there is a directed path from* u' *to* v' *in* G'*,*
- *(b)* G' *has* $O(m+g)$ *vertices.*

Proof. Fix a forest decomposition F and let G' be the pattern graph $\mathrm{P}(G,F)$. Theorem 14.39 shows that there is a path from u to v in G if and only if there is a path from u' to v' in $\mathrm{P}(G,F)$ if and only if there is a path from u' to v' in $\mathrm{P}(G,F)$. Lemma 14.40 gives that G' is log-space computable. By Lemma 14.26, there are at most $O(m+g)$ equivalence classes in G (with respect to F), and there is a constant multiple of pattern descriptions per equivalence class, so G' has $O(m+g)$ vertices. $\square$

14.6 Discussion

We have succeeded in enlarging the class of surface-embedded DAGs which admit deterministic log-space algorithms for reachability. By extending the concept of topological equivalence from [126], we have shown that this is a useful algorithmic construct. Perhaps the structures built in this chapter have application to other problems. Placing planar DAG reachability within ł will likely require significant new ideas since the source-to-genus tradeoff hints that an algorithm for m-source planar DAGs will also apply to m-genus DAGs.

Further, the algorithms developed in this work improve upper bounds for the class $\mathcal{G}(m, g)$ for sub-polynomial values of m and g. See Table 14.1 for a list of space bounds of different algorithms for reachability in certain classes of graphs. Table 14.2 describes which results give which space bounds with simultaneous polynomial-time algorithms.

Ealier known graph class	Space bound s	New graph class given by Theorem 14.3
Undirected Graphs [109] SMPD[4] [2] LMPD[5] [126]	$O(\log n)$	$\mathcal{G}\left(2^{O(\sqrt{\log n})}, 2^{O(\sqrt{\log n})}\right)$
Poly-mixing time [110, 116]	$O\left(\log^{\frac{3}{2}} n\right)$	$\mathcal{G}\left(2^{O(\log^{\frac{3}{4}} n)}, 2^{O(\log^{\frac{3}{4}} n)}\right)$
Reach-poly graphs [3, 48]	$O\left(\frac{\log^2 n}{\log\log n}\right)$	$\mathcal{G}\left(2^{O\left(\frac{\log n}{\sqrt{\log\log n}}\right)}, 2^{O\left(\frac{\log n}{\sqrt{\log\log n}}\right)}\right)$
	$o(\log^2 n)$	$\mathcal{G}(n^{o(1)}, n^{o(1)})$
All directed graphs [117]	$O(\log^2 n)$	

Table 14.1: A table of graph classes (old and new) for which reachability can be solved using space s, for various interesting values of s.

Earlier known graph class	Space bound s with poly-time	New graph class given by Theorem 14.8
Poly-mixing time [110, 99] Reach-poly graphs[6] [80, 32]	$O(\log^2 n)$	
	$2^{O\left(\log^{\frac{1}{2}+\varepsilon} n\right)}$	$\mathcal{G}\left(2^{O\left(\log^{\frac{1}{2}+\varepsilon} n\right)}, 2^{O\left(\log^{\frac{1}{2}+\varepsilon} n\right)}\right)$
	$o(n^{\varepsilon})$	$\mathcal{G}(O(n^{\varepsilon}), O(n^{\varepsilon}))$.
All directed graphs [12]	$O\left(\frac{n}{2^{\sqrt{\log n}}}\right)$	

Table 14.2: A table of graph classes (old and new) with simultaneous time-space bound $(n^{O(1)}, s)$ for reachability for various values of s.

---

[4] SMPD: Single-source Multiple-sink Planar DAG

[5] LMPD: Log-source Multiple-sink Planar DAG

[6] It is a quick observation that reachability in reach-poly graphs is decidable by a LogDCFL machine.

Bibliography

[1] E. Allender. NL-printable sets and nondeterministic Kolmogorov complexity. *Theor. Comput. Sci.*, 355(2):127–138, 2006.

[2] E. Allender, D. A. M. Barrington, T. Chakraborty, S. Datta, and S. Roy. Planar and grid graph reachability problems. *Theory of Computing Systems*, 45(4):675–723, 2009.

[3] E. Allender and K.-J. Lange. $\mathsf{RUSPACE}(\log n) \subseteq \mathsf{DSPACE}(\log^2 n / \log\log n)$. *Theory of Computing Systems*, 31:539–550, 1998. Special issue devoted to the 7th Annual International Symposium on Algorithms and Computation (ISAAC'96).

[4] E. Allender, K. Reinhardt, and S. Zhou. Isolation, Matching, and Counting Uniform and Nonuniform Upper Bounds. *Journal of Computer and System Sciences*, 59(2):164–181, 1999.

[5] N. Alon and S. Friedland. The maximum number of perfect matchings in graphs with a given degree sequence. *Electronic Journal of Combinatorics*, 14(1), 2008. Note 13.

[6] C. Àlvarez and B. Jenner. A very hard log-space counting class. *Theoret. Comput. Sci.*, 107:3–30, 1993.

[7] S. Arora and B. Barak. *Computational complexity: a modern approach*, volume 1. Cambridge University Press, 2009.

[8] A. Atamtürk, G. L. Nemhauser, and M. W. P. Savelsbergh. Conflict graphs in solving integer programming problems. *European Journal of Operational Research*, 121(1):40 – 55, 2000.

[9] L. Babai. On the minimum order of graphs with given group. *Canadian Mathematical Bulletin*, 17:467–470, 1974.

[10] L. Babai. Automorphism groups, isomorphism, reconstruction. In *Handbook of combinatorics, Vol. 1, 2*, pages 1447–1540. Elsevier, Amsterdam, 1995.

[11] R. Baker, G. Ebert, J. Hemmeter, and A. Woldar. Maximal cliques in the paley graph of square order. *Journal of statistical planning and inference*, 56(1):33–38, 1996.

[12] G. Barnes, J. F. Buss, W. L. Ruzzo, and B. Schieber. A sublinear space, polynomial time algorithm for directed s-t connectivity. In *Structure in Complexity Theory Conference, 1992., Proceedings of the Seventh Annual*, pages 27–33, 1992.

[13] M. Bašić and A. Ilić. On the clique number of integral circulant graphs. *Applied Mathematics Letters*, 22(9):1406–1411, 2009.

[14] E. Berlekamp. A construction for partitions which avoid long arithmetic progressions. *Canad. Math. Bull*, 11(1968):409–414, 1968.

[15] A. Blokhuis. On subsets of gf (q2) with square differences. 87(4):369–372, 1984.

[16] B. Bollobás. *Modern graph theory*, volume 184. Springer Verlag, 1998.

[17] B. Bollobás. *Random graphs*, volume 73 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, second edition, 2001.

[18] B. Bollobás. *Extremal graph theory*. Dover Pubns, 2004.

[19] J. A. Bondy. A graph reconstructor's manual. In *Surveys in combinatorics, 1991 (Guildford, 1991)*, volume 166 of *London Math. Soc. Lecture Note Ser.*, pages 221–252. Cambridge Univ. Press, Cambridge, 1991.

[20] C. Bourke, R. Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *ACM Transactions on Computation Theory*, 1(1):1–17, 2009.

[21] L. M. Brègman. Some properties of nonnegative matrices and their permanents. *Soviet Math. Dokl.*, 14:945–949, 1973.

[22] I. Broere, D. Döman, and J. Ridley. The clique numbers and chromatic numbers of certain paley graphs. *Quaestiones Mathematicae*, 11(1):91–93, 1988.

[23] J. Brown and R. Hoshino. Proof of a conjecture on fractional ramsey numbers. *Journal of Graph Theory*, 63(2):164–178, 2010.

[24] T. Brown, P. Erdös, and A. Freedman. Quasi-progressions and descending waves.(in english). *J. Comb. Theory, Ser. A*, 53(1):81–95, 1990.

[25] G. Buntrock, C. Damm, U. Hertrampf, and C. Meinel. Structure and importance of logspace-mod class. *Mathematical Systems Theory*, 25(3):223–237, 1992.

[26] G. Buntrock, L. A. Hemachandra, and D. Siefkes. Using inductive counting to simulate nondeterministic computation. *Information and Computation*, 102(1):102–117, 1993.

[27] G. Buntrock, B. Jenner, K.-J. Lange, and P. Rossmanith. Unambiguity and fewness for logarithmic space. In *Proceedings of the 8th International Conference on Fundamentals of Computation Theory (FCT'91)*, Volume 529 Lecture Notes in Computer Science, pages 168–179. Springer-Verlag, 1991.

[28] J. Chen, S. Kanchi, and A. Kanevsky. A note on approximating graph genus. *Information processing letters*, 61(6):317–322, 1997.

[29] M. Chudnovsky. Berge trigraphs. *J. Graph Theory*, 53(1):1–55, 2006.

[30] S. Cohen. Clique numbers of paley graphs. *Quaestiones Mathematicae*, 11(2):225–231, 1988.

[31] D. Collins, J. Cooper, B. Kay, and P. S. Wenger, 2011. personal communication.

[32] S. Cook. Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In *Proceedings of the eleventh annual ACM Symposium on Theory of Computing*, pages 338–345. ACM, 1979.

[33] K. Coolsaet, J. Degraer, and E. Spence. The strongly regular $(45, 12, 3, 3)$ graphs. *The Electronic Journal of Combinatorics*, 13(R32):1, 2006.

[34] J. Cooper, J. Lenz, T. LeSaulnier, P. Wenger, and D. West. Uniquely c 4-saturated graphs. *Graphs and Combinatorics*, pages 1–9, 2010.

[35] I. CPLEX. ILOG CPLEX 10.0, January 2006.

[36] J. Cutler and A. J. Radcliffe. An entropy proof of the Kahn-Lovász Theorem. *Electronic Journal of Combinatorics*, 18(1), January 2011. P10.

[37] R. Diestel. Graph theory (graduate texts in mathematics vol 173), 2000.

[38] A. Dudek and J. Schmitt. On the size and structure of graphs with a constant number of 1-factors. *Discrete Mathematics*, 2012. to appear.

[39] M. Elberfeld, A. Jakoby, and T. Tantau. Logspace versions of the theorems of bodlaender and courcelle. In *FOCS '10: Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, 2010.

[40] P. Erdős. Problems in number theory and combinatorics. In *Proc. 6th Manitoba Conference on Numerical Mathematics, Congress Numer*, volume 18, pages 35–58, 1976.

[41] P. Erdős and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. *Infinite and finite sets*, 10:609–627, 1975.

[42] P. Erdős and A. Stone. On the structure of linear graphs. *Bull. Amer. Math. Soc*, 52:1087–1091, 1946.

[43] P. Erdös, A. Hajnal, and J. Moon. A problem in graph theory. *The American Mathematical Monthly*, 71(10):1107–1110, 1964.

[44] P. Erdos and M. Simonovits. A limit theorem in graph theory. *Studia Sci. Math. Hungar*, 1(51-57):51, 1966.

[45] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31(3):538–544, 1984.

[46] S. Friedland. An upper bound for the number of perfect matchings in graphs, 6 March 2008. arXiv: 0803.0864v1.

[47] R. Frucht. Herstellung von Graphen mit vorgegebener abstrakter Gruppe. *Compositio Math.*, 6:239–250, 1939.

[48] B. Garvin, D. Stolee, R. Tewari, and N. V. Vinodchandran. ReachUL = ReachFewL. *17th Annual International Computing and Combinatorics Conference*, 2011.

[49] D. Geller and B. Manvel. Reconstruction of cacti. *Canad. J. Math.*, 21:1354–1360, 1969.

[50] W. Gowers. A new proof of szemerédi's theorem. *Geometric and Functional Analysis*, 11(3):465–588, 2001.

[51] R. Graham. Some of my favorite problems in ramsey theory. *Integers: Electronic Journal of Combinatorial Number Theory*, 7(2):A15, 2007.

[52] B. Green*. Counting sets with small sumset, and the clique number of random cayley graphs. *Combinatorica*, 25(3):307–326, 2005.

[53] B. Green and T. Tao. The primes contain arbitrarily long arithmetic progressions. *Ann. Math*, 167(208):481–547, 2008.

[54] D. L. Greenwell. Reconstructing graphs. *Proc. Amer. Math. Soc.*, 30:431–433, 1971.

[55] D. L. Greenwell and R. L. Hemminger. Reconstructing graphs. In *The Many Facets of Graph Theory (Proc. Conf., Western Mich. Univ., Kalamazoo, Mich., 1968)*, pages 91–114. Springer, Berlin, 1969.

[56] D. Gross, N. Kahl, and J. T. Saccoman. Graphs with the maximum or minimum number of 1-factors. *Discrete Math.*, 310:687–691, 2010.

[57] R. K. Guy. *Unsolved problems in number theory*. Problem Books in Mathematics. Springer-Verlag, third edition edition, 2004.

[58] P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 1(1):26–30, 1935.

[59] S. G. Hartke, H. Kolb, J. Nishikawa, and D. Stolee. Edge-reconstruction for small 2-connected graphs, 2009. unpublished.

[60] S. G. Hartke, H. Kolb, J. Nishikawa, and D. Stolee. Automorphism groups of a graph and a vertex-deleted subgraph. *Electron. J. Combin.*, 17(1):Research Paper 134, 8, 2010.

[61] S. G. Hartke and A. Radcliffe. Mckay's canonical graph labeling algorithm. In *Communicating Mathematics*, volume 479 of *Contemporary Mathematics*, pages 99–111. American Mathematical Society, 2009.

[62] S. G. Hartke and D. Stolee. Uniquely K_r-saturated graphs, 2012. preprint.

[63] S. G. Hartke, D. Stolee, D. B. West, and M. Yancey. On extremal graphs with a given number of perfect matchings, 2011. preprint.

[64] A. Hoffman and R. Singleton. On moore graphs with diameters 2 and 3. *IBM Journal of Research and Development*, 4:497–504, 1960.

[65] R. Hoshino. *Independence polynomials of circulant graphs*. Citeseer, 2007.

[66] S. G. Hwang. A note on system of distinct representatives. *Kyungpook Math. J.*, 35:513–516, 1996.

[67] N. Immerman. Nondeterministic space is closed under complementation. In *Structure in Complexity Theory Conference, 1988. Proceedings., Third Annual*, pages 112–115. IEEE, 1988.

[68] A. Jakoby, M. Liśkiewicz, and R. Reischuk. Space efficient algorithms for directed series-parallel graphs. *Journal of Algorithms*, 60(2):85–114, 2006.

[69] A. Jakoby and T. Tantau. Logspace algorithms for computing shortest and longest paths in series-parallel graphs. In *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science*, pages 216–227, 2007.

[70] A. Jobson, A. Kézdy, H. Snevily, and S. C. White. Ramsey functions for quasi-progressions with large diameter, 2010. preprint.

[71] A. Jobson, A. Kézdy, and D. Stolee. A new variant of van der Waerden numbers, 2012. in preparation.

[72] P. Kaski and P. Ostergard. The steiner triple systems of order 19. *Mathematics of Computation*, 73(248):2075–2092, 2004.

[73] P. Kaski and P. R. J. Östergård. *Classification Algorithms for Codes and Designs*. Number 15 in Algorithms and Computation in Mathematics. Springer-Verlag, Berlin Heidelberg, 2006.

[74] W. Klotz and T. Sander. Some properties of unitary cayley graphs. *Electronic Journal of Combinatorics*, 14, 2007.

[75] M. Kouril and J. Paul. The van der waerden number w (2, 6) is 1132. *Experimental Mathematics*, 17(1):53–61, 2008.

[76] E. Kupin, B. Reiniger, and D. Stolee. Counting chains in width-two posets with few cover edges, 2012. in preparation.

[77] J. Kynčl and T. Vyskočil. Logspace reduction of directed reachability for bounded genus graphs to the planar case. *ACM Transactions on Computation Theory*, 1(3):1–11, 2010.

[78] C. Lam, L. Thiel, and S. Swiercz. The non-existence of finite projective planes of order 10. *Canad. J. Math*, 41(6):1117–1123, 1989.

[79] B. Landman. Ramsey functions for quasi-progressions. *Graphs and Combinatorics*, 14(2):131–142, 1998.

[80] K.-J. Lange. An unambiguous class possessing a complete set. In *STACS '97: Proceedings of the 14th Annual Symposium on Theoretical Aspects of Computer Science*, pages 339–350, 1997.

[81] J. Lauri and R. Scapellato. *Topics in graph automorphisms and reconstruction*, volume 54 of *London Mathematical Society Student Texts*. Cambridge University Press, Cambridge, 2003.

[82] M. W. Liebeck. On graphs whose full automorphism group is an alternating group or a finite classical group. *Proceedings of the London Mathematical Society*, 3:337–362, 1983.

[83] V. Linek, 2011. via D. B. West.

[84] L. Lovász. A note on the line reconstruction problem. *J. Combinatorial Theory Ser. B*, 13:309–310, 1972.

[85] L. Lovasz. Ear-decompositions of matching-covered graphs. *Combinatorica*, 3(1):105–117, 1983.

[86] L. Lovász and M. D. Plummer. On bicritical graphs. *Infinite and finite sets*, 10:1051–1079, 1975.

[87] L. Lovász and M. D. Plummer. *Matching Theory*. AMS Chelsea Publishing Series. AMS Bookstore, 2009.

[88] B. Manvel. Reconstruction of trees. *Canad. J. Math.*, 22:55–60, 1970.

[89] B. Manvel. On reconstructing graphs from their sets of subgraphs. *Journal of Combinatorial Theory, Series B*, 21(2):156–165, 1976.

[90] R. Martin and J. Smith. Induced saturation number, 2011. preprint.

[91] B. D. McKay. Small graphs are reconstructible. *Australas. J. Combin.*, 15:123–126, 1997.

[92] B. D. McKay. Isomorph-free exhaustive generation. *J. Algorithms*, 26(2):306–324, 1998.

[93] B. D. McKay. nauty user's guide (version 2.4). *Dept. Computer Science, Austral. Nat. Univ.*, 2006.

[94] B. D. McKay and E. Spence. Classification of regular two-graphs on 36 and 38 vertices. *Australasian Journal of Combinatorics*, 24:293–300, 2001.

[95] S. Micali and V. V. Vazirani. An $o(v^{1/2}e)$ algorithm for finding maximum matching in general graphs. *Proc. 21st Symp. on Foundations of Computer Science*, pages 17–27, 1980.

[96] S. D. Monson. The reconstruction of cacti revisited. *Congr. Numer.*, 69:157–166, 1989. Eighteenth Manitoba Conference on Numerical Mathematics and Computing (Winnipeg, MB, 1988).

[97] V. Müller. The edge reconstruction hypothesis is true for graphs with more than $n \cdot \log_2 n$ edges. *J. Combinatorial Theory Ser. B*, 22(3):281–283, 1977.

[98] G. L. Nemhauser, M. W. P. Savelsbergh, and G. C. Sigismondi. MINTO, a Mixed INTeger Optimizer. *Operations Research Letters*, 15:47–58, 1994.

[99] N. Nisan. RL $\subseteq$ SC. In *In Proceedings of the Twenty Fourth Annual ACM Symposium on Theory of Computing*, pages 619–623, 1995.

[100] S. Niskanen and P. R. J. Östergård. *Cliquer* user's guide, version 1.0. *Technical Report*, T48, 2003.

[101] P. A. Ostrand. Systems of distinct representatives, ii. *J. Math. Anal. Appl.*, 32:1–4, 1970.

[102] J. Ostrowski, J. Linderoth, F. Rossi, and S. Smriglio. Orbital branching. In *IPCO '07: Proceedings of the 12th international conference on Integer Programming and Combinatorial Optimization*, volume 4513 of *LNCS*, pages 104–118, Berlin, Heidelberg, 2007. Springer-Verlag.

[103] J. Ostrowski, J. Linderoth, F. Rossi, and S. Smriglio. Constraint orbital branching. In A. Lodi, A. Panconesi, and G. Rinaldi, editors, *Integer Programming and Combinatorial Optimization, 13th International Conference, IPCO 2008, Bertinoro, Italy, May 26-28, 2008, Proceedings*, volume 5035 of *Lecture Notes in Computer Science*. Springer, 2008.

[104] J. Ostrowski, J. Linderoth, F. Rossi, and S. Smriglio. Solving large steiner triple covering problems. *Technical Reports of the Computer Sciences Department, University of Wisconsin-Madison*, (1663), 2009.

[105] R. E. A. C. Paley. On orthogonal matrices. *J. Math. Physics*, 12:311–320, 1933.

[106] A. Pavan, R. Tewari, and N. V. Vinodchandran. On the power of unambiguity in logspace. *Computational Complexity*, 2010. to appear.

[107] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, et al. The Open Science Grid. In *Journal of Physics: Conference Series*, volume 78, pages 12–57. IOP Publishing, 2007.

[108] J. Radhakrishnan. An Entropy Proof of Brégman's Theorem. *Journal of Combinatorial Theory, Series A*, 77(1):161–164, 1997.

[109] O. Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4), 2008.

[110] O. Reingold, L. Trevisan, and S. Vadhan. Pseudorandom walks on regular digraphs and the RL vs. L problem. In *STOC '06: Proceedings of the thirty-eighth annual ACM Symposium on Theory of Computing*, pages 457–466, New York, NY, USA, 2006. ACM.

[111] K. Reinhardt and E. Allender. Making nondeterminism unambiguous. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 244–253. IEEE, 2002.

[112] R. Robinson. Tables. available at http://www.cs.uga.edu/ rwr/publications/tables.pdf.

[113] R. Robinson. Enumeration of non-separable graphs*. *Journal of Combinatorial Theory*, 9(4):327–356, 1970.

[114] G. Sabidussi. On the minimum order of graphs with a given automorphism group. *Monatsh. Math.*, 63:124–127, 1959.

[115] G. Sabidussi. On the minimum order of graphs with given automorphism group. *Monatshefte für Mathematik*, 63(2):124–127, 1959.

[116] M. Saks and S. Zhou. $\mathrm{BP}_H\mathrm{SPACE}(S) \subseteq \mathrm{DSPACE}(S^{3/2})$. *Journal of Computer and System Sciences*, 58(2):376–403, 1999.

[117] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[118] J.-P. Serre. *Trees*. Springer-Verlag, Berlin, 1980. Translated from the French by John Stillwell.

[119] N. J. A. Sloane. A001349: Number of connected graphs with n nodes, 2000. http://oeis.org/A002218.

[120] N. J. A. Sloane. A002218: Number of unlabeled nonseparable (or 2-connected) graphs (or blocks) with n nodes, 2000. http://oeis.org/A002218.

[121] E. Spence. The strongly regular (40, 12, 2, 4) graphs. *Electr. J. Combin*, 7:R22, 2000.

[122] D. Stolee. TreeSearch users guide, 2010.

[123] D. Stolee. Generating p-extremal graphs, 2011. preprint.

[124] D. Stolee. Isomorph-free generation of 2-connected graphs with applications. Technical Report #120, University of Nebraska–Lincoln, Computer Science and Engineering, 2011.

[125] D. Stolee. Automorphism groups and adversarial vertex deletions, 2012. preprint.

[126] D. Stolee, C. Bourke, and N. V. Vinodchandran. A log-space algorithm for reachability in planar acyclic digraphs with few sources. *25th Annual IEEE Conference on Computational Complexity*, pages 131–138, 2010.

[127] D. Stolee and N. V. Vinodchandran. Space-efficient algorithms for reachability in surface-embedded graphs. *27th Annual IEEE Conference on Computational Complexity*, 2012. to appear.

[128] Z. Szabo. An application of lovász'local lemma-a new lower bound for the van der waerden number. *Random Structures & Algorithms*, 1(3):343–360, 1990.

[129] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

[130] E. Szemerédi. On sets of integers containing no four elements in arithmetic progression. *Acta Mathematica Hungarica*, 20(1):89–104, 1969.

[131] E. Szemerédi. On sets of integers containing no k elements in arithmetic progression. *Acta Arithmetica*, 27:199Ð245, 1975.

[132] E. Szemerédi. Regular partitions of graphs. Technical report, DTIC Document, 1975.

[133] Z. Szigeti. The two ear theorem on matching-covered graphs. *J. Combin. Theory Ser. B*, 74(1):104–109, 1998.

[134] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[135] T. Thierauf and F. Wagner. Reachability in $K_{3,3}$-free graphs and K_5-free graphs is in unambiguous log-space. In *FCT*, pages 323–334, 2009.

[136] C. Thomassen. The graph genus problem is NP-complete. *Journal of Algorithms*, 10(4):568–576, 1989.

[137] P. Turán. On an extremal problem in graph theory. *Mat. Fiz. Lapok*, 48:436–452, 1941.

[138] W. T. Tutte. The factorization of linear graphs. *J. London Math. Soc.*, 22:107–111, 1947.

[139] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[140] B. van der Waerden. Beweis einer baudetschen vermutung. *Nieuw Arch. Wisk*, 15:212–216, 1927.

[141] S. Vijay. On a variant of van der waerden's theorem. *Integers*, 10(2):223–227, 2010.

[142] T. Walsh and E. Wright. The k-connectedness of unlabelled graphs. *Journal of the London Mathematical Society*, 2(3):397, 1978.

[143] D. J. Weitzel. *Campus Grids: A framework to facilitate resource sharing*. Masters thesis, University of Nebraska - Lincoln, 2011.

[144] P. S. Wenger. Uniquely c_k-saturated graphs. in preparation.

[145] P. S. Wenger, 2011. personal communication.

[146] D. B. West. *Introduction to Graph Theory*. Prentice-Hall, second edition, 2001.

[147] H. Whitney. Congruent Graphs and the Connectivity of Graphs. *Amer. J. Math.*, 54(1):150–168, 1932.

[148] L. Xu, Z. Xia, and Y. Yang. Some results on the independence number of circulant graphs $c(n;\{1,k\})$. *OR Trans.*, 13(4):65?70, 2009.

[149] Y. Z. Yang. The reconstruction conjecture is true if all 2-connected graphs are reconstructible. *J. Graph Theory*, 12(2):237–243, 1988.

[150] Q. Yu and G. Liu. *Graph factors and matching extensions*. Springer, 2009.

Appendix A

Symbols

Symbol	Meaning
$\exists$	Existential Quantifier
$\forall$	Universal Quantifier
$\wedge$	And
$\vee$	Or
$\equiv$	Equality Comparison
$\leftarrow$	Assignment
$\Rightarrow$	Implication
$\xleftarrow{\max}$	Assignment, when the input value is larger than the current value.

Table A.1: Symbols

Appendix B

TreeSearch User Guide

B.1 Introduction

The computation path of a dynamic search frequently takes the form of a rooted tree. One important property of each node in this tree is that the computation at that node depends only on the previous nodes in the ancestral path leading to the root of the computation. If the search is implemented in the usual way, subtrees operate independently.

For a search of this type, all search nodes at a given depth can be generated by iterating through the search tree, but backtracking once the target depth is reached. Each of the subtrees at this depth can be run independently, and hence it is common to run these jobs concurrently (See [73] Chapter 5 for more information). Since the subtrees are independent, no communication is necessary for these jobs, and the jobs can be run on a distributed machine such as a cluster or grid.

The *TreeSearch* library was built to maximize code reuse for these types of search. It abstracts the structure of the tree and the recursive nature of the search into custom components available for implementation by the user. Then, the ability to generate a list of jobs, run individual jobs, and submit the list of jobs to a cluster are available with minimal extra work.

TreeSearch is intended for execution on a distributed machine using Condor [134], a job scheduler that uses idle nodes of a cluster or grid. Condor was chosen as its original development was meant for installation in computer labs and office machines at the University of Wisconsin–Madison to utilize idle computers.

The C++ portion of *TreeSearch* is independent of Condor. The Python scripts which manage the input and output files as well as modifying the submission script are tied to Condor, but could be adapted for use in other schedulers.

B.1.1 Acquiring *TreeSearch*

The latest version of *TreeSearch* and its documentation is publicly available on GitHub [**?**] at the address http://www.github.com/derrickstolee/TreeSearch/.

B.2 Strategy

Let us begin by describing the general structure and process of an abstract tree-based search. There is a unique root node at depth zero. Each node in the tree searches in a depth-first, recursive manner. There are a number of children to select at each node. One may select this child through iteration or selecting via a numerical label. Before searching below the child, a pruning procedure may be called to attempt to rule out the possibility of a solution below that child. Another procedure may be used to find if this node is a solution. Now, the search recurses at this node until its children are exhausted and the search continues back to its parent.

B.2.1 Subtrees as Jobs

This tree structure allows for search nodes to be described via the list of children taken at each node. Typically, the breadth of the search will be small and these descriptions take very little space. This allows for a method of describing a search node independently of what data is actually stored by the specific search application. Moreover, the application may require visiting the ancestor search nodes in order to have consistent internal data. With the assumption that each subtree is computationally independent of other subtrees at the same level, one can run each subtree in a different process in order to achieve parallelization. These path descriptions make up the input for the specific processes in this scheme.

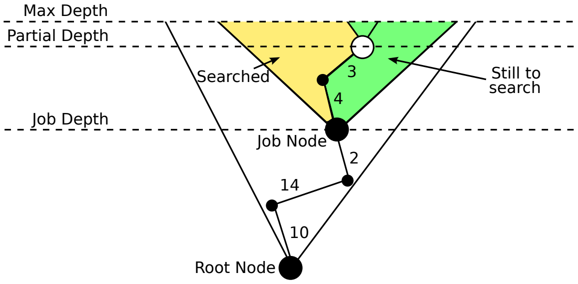


Figure B.1: A partial job description.

Each path to a search node qualifies as a single job, where the goal is to expand the entire search tree below that node. A collection of nodes where no pair are in an ancestor-descendant relationship qualifies as a list of independent jobs. Recognizing that the amount of computation required to expand the subtree at a node is not always a uniform value, *TreeSearch* allows a maximum amount of time within a given job. In order to recover the state of the search when the execution times out, the concept of *partial jobs* was defined. A partial job describes the path from the root to the current search node. In addition, it describes which node in this path is the original job node. The goal of a partial job is to expand the remaining nodes in the subtree of the job node, without expanding any nodes to the left of the last node in this path. See Figure B.1 to an example partial job and its position in the job subtree.

B.2.2 Job Descriptions

The descriptions of jobs and partial jobs are described using text files in order to minimize the I/O constraints on the distributed system. The first is the standard job, given by a line starting with the letter `J`. Following this letter are a sequence of numbers in hexadecimal. The first two should be the same, corresponding to the depth of the node. The remaining numbers correspond to the child values at each depth from the root to the job node.

A partial job is described by the letter `P`. Here, the format is the same as a standard job except the first number describes the depth of the job node and the second number corresponds to the depth of the current node. For example, the job and partial job given in Figure B.1 are described by the strings below:

```
J 3 3 10 14 2
P 3 5 10 14 2 4 3
```

B.2.3 Customization

The *TreeSearch* library consists of an iterative implementation of the abstract search. The corresponding actions for a specific application are contacted via extending the `SearchManager` class and implementing certain virtual functions. The list of functions available are given in Table B.1.

In addition to supplying the logic behind these functions, protected members of the `SearchManager` class can be modified to change the operation of the search.

`LONG_T  pushNext()`	Deepen the search to the next child of the current node.
`LONG_T  pushTo(LONG_T child)`	Deepen the search to the specified child of the current node.
`LONG_T  pop()`	Remove the current node and move up the tree.
`int     prune()`	Perform a check to see if this node should be pruned.
`int     isSolution()`	Perform a check to see if a solution exists at this point.
`char*   writeSolution()`	Create a buffer that contains a description of the solution.
`char*   writeStatistics()`	Create a buffer that contains custom statistics.

Table B.1: List of virtual functions in the `SearchManager` class.

These parameters are listed in Table B.2.

B.3 Integration with *TreeSearch*

This section details the specific interfaces for implementation with *TreeSearch*.

It is important to understand the order of events when the search is executing. The search begins when the `doSearch()` methd is called. The first call initializes the search, including starting the kill timer. Then, each recursive call expands the current search node at the top of the stack. Figure B.3 describes the actions taken by the recursive `doSearch()` method takes at each search node.

B.3.1 Virtual Functions

The two most important methods are the `pushNext()` and `pushTo(LONG_T child)` methods. Both deepen the search, manage the stack, and control the job descriptions. Each returns a child description (of type `LONG_T`)

Type	Name	Option	Description
`int`	`maxdepth`	`-m [N]`	The maximum depth the search will go. In generate mode, a job will be output with job description given by the current node.
`int`	`killtime`	`-k [N]`	Number of seconds before the search is halted. If the search has not halted naturally, a partial job is output at the current node.
`int`	`maxSolutions`	`-maxsols [N]`	The maximum number of solutions to output. When this number of solutions is reached, a partial job is output and the search halts.
`int`	`maxJobs`	`-maxjobs [N]`	The maximum number of jobs to output (generate mode). When this number of jobs is reached, a partial job is output and the search halts.
`bool`	`haltAtSolutions`	`-haltatsols [yes/no]`	If `true`, the search will stop deepening if `isSolution()` signals a solution. If `false`, the search will continue until specified by `prune()` or `maxdepth`.

Table B.2: List of members in the `SearchManager` class.

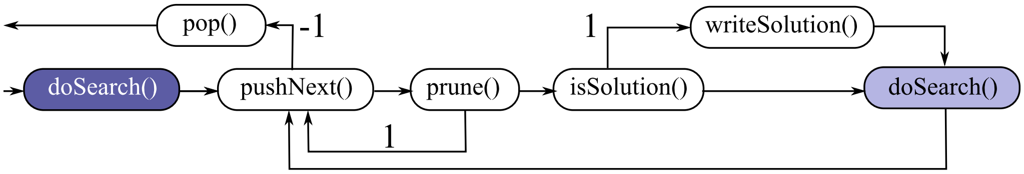


Figure B.2: The conceptual operation of the `doSearch()` method.

`pushNext()`: Advance the search stack using the next augmentation available at the current node. Return a label of `LONG_T` type which describes the augmentation. Return -1 if there are no more augmentations to attempt at this stage, signaling a `pop()` operation.

`pushTo(LONG_T child)`: Advance the search stack using the augmentation specified by `child`. If the augmentation fails as specified, return -1, and the search

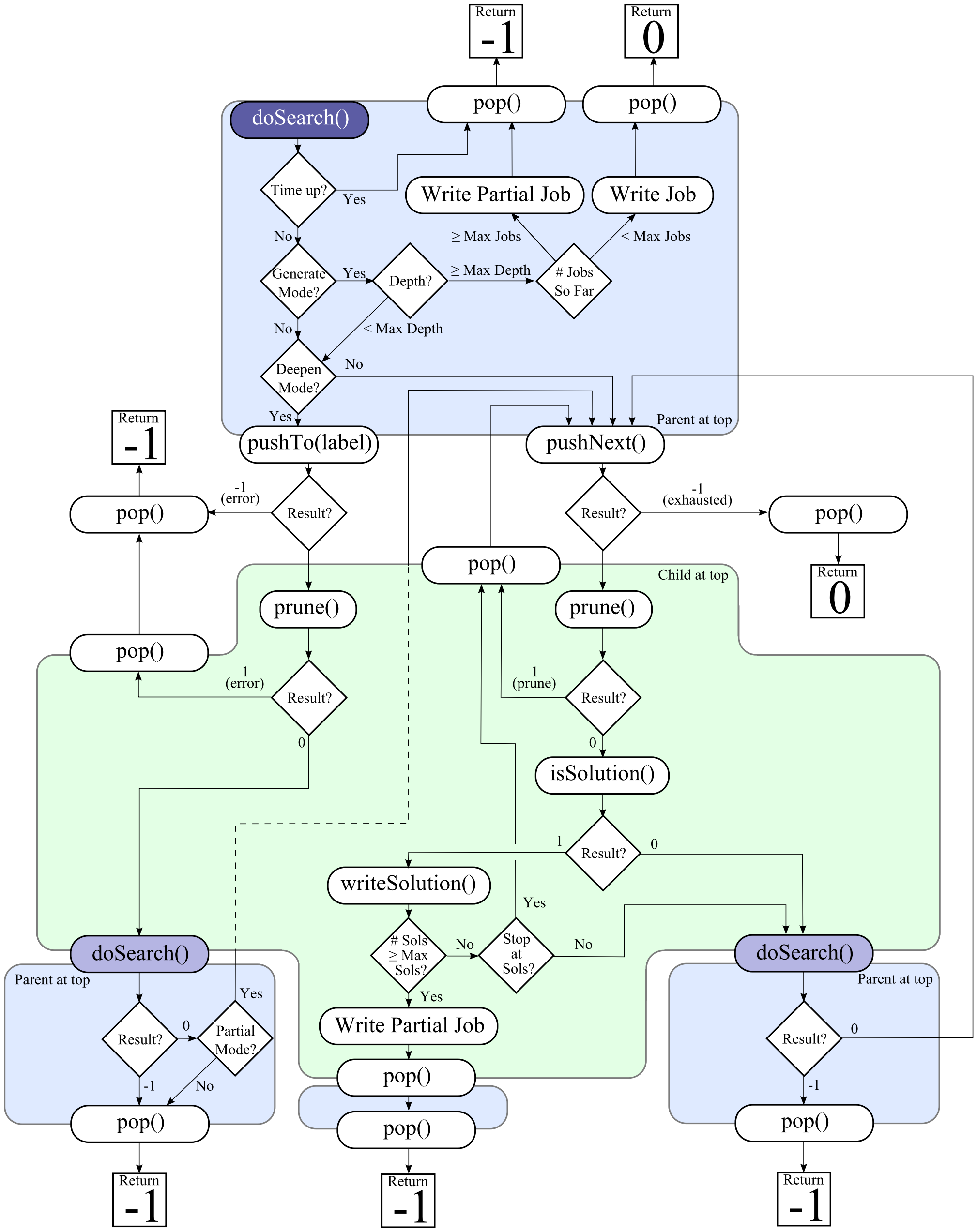


Figure B.3: The full operation of the `doSearch()` method.

will terminate in error. **Note:** If this method is called during the partial portion of a job description, the later augmentations will be called using the `pushNext()` method, so the current augmentation must be stored for later.

`pop()`: Clean up any memory used for the current level and/or revert any data structures to the previous level.

`prune()`: Check if the current node should be pruned (i.e. detect if there are no solutions reachable from the current node by performing the specified augmentation). Return 1 if the node should be pruned, 0 otherwise. A prune signal will be followed by the `pop()` method.

`isSolution()`: Check if there is a solution at this node. If there is a solution, store the solution data and return 1. The `writeSolution()` method will be called to pass the solution data to output. If the `haltAtSolutions` option is set, a solution will trigger a `pop()` method. Otherwise, the node will be used for augmentations until the maximum depth is reached or the `prune()` method signals a prune for all later nodes.

`writeSolution()`: Return a buffer containing the solution data for the output. This data will be sent to standard out along with the job description of the current node (with an '`S`' prefix). If your data is prohibitively large for sending over the network, this job description can be used to generate the current node where the solution data can be recovered. **Note:** The buffer must be allocated using `malloc`, as it will be deallocated using `free`.

`writeStatistics()`: Write a buffer of custom statistics information. Each line must follow the format

T [TYPE] [ID] [VALUE]

where [TYPE] is one of “SUM", “MAX", or “MIN", [ID] is a specified name, and

[VALUE] is a number. These statistics are combined by the `compactjobs` script using the [TYPE] specifier (either sum the values or store the maximum or minimum) and the statistics are placed in the `allstats.txt` file. The `combinestats` script converts the `allstats.txt` file into a comma separated value file, grouping by depth over variables with [ID] of the form [SUBID]_AT_[#]. This allows per-depth statistics tracking.

B.3.2 Helper Methods

The following methods are useful when constructing a *TreeSearch* application.

`importArguments(int argc, char** argv):` Take the command line arguments and set the standard search options. The options from Table B.2 such as `killtime`, `maxJobs`, and `maxdepth` are all set in this way.

`readJob(FILE* file):` Read a job from the given input source, such as standard in. It will read only one line, and prepare the manager for the `doSearch()` method.

`doSearch():` This method starts the search on the current job as well as returns the status of the search: 1 if completed with a solution, 0 if completed with no solution, and -1 if halted early due to error or time constraints.

B.3.3 Compilation

To compile *TreeSearch*, run `make` in the source directory. This command compiles the object file `SearchManager.o` which must be linked into your executable. Moreover, it compiles the example application presented in Section B.5. Your code must reference the header file `SearchManager.hpp` and link the object file `SearchManager.o`.

B.4 Execution and Job Management

To execute a single process, simply run your executable with the proper arguments. However, to run a distributed job via Condor, a set of scripts were created to manage the input and output files, the Condor submission file, and monitor the progress of the submission during execution.

B.4.1 Management Scripts

The *TreeSearch* library works best with independent subtrees and hence does not suffer from scaling issues when the parallelism is increased. However, managing these large lists of jobs requires automation.

B.4.1.1 Expanding jobs before a run

When the generation step is run, a list of jobs is presented in a single file. Condor requires a separate input and output file for each process. The role of the `expandjobs` script is to split the jobs into individual files and to set up the Condor submission file for the number of jobs that are found.

There are a few customizable options for this script.

- `-f [folder]` – change the folder where the jobs are created. Default is `./`.
- `-m [maximum]` – set the maximum number of jobs allowed. Default is unlimited.
- `-g [groupsize]` – set the number of jobs per process. Default is 1.

Inside the specified folder, the file `condorsubmit.sub.tmp` is modified and copied to `condorsubmit.sub` with the proper queue size based on the number of jobs found. Any remaining jobs that did not fit within the maximum are held as back jobs. They will be added to the job pool when the jobs are completed.

B.4.1.2 Collecting data after a run

Once Condor has completed the requested jobs, the output must be collected to discover which jobs completed fully, which are partially complete, and how many solutions have been found. The script `compactjobs` was built for this purpose.

This script takes the output files and reads all new jobs that may have been generated using the staging feature, finds if the input job completed or is partial, and reports on the total number of jobs of each type. Moreover, it will find and store the solutions found, along with the corresponding data.

Finally, it compiles statistics from each run. Using the `writeStatistics` method, the application may report statistics by starting the line with a "T" followed by the type (MAX, MIN, SUM), variable name, and variable value. These are collected using the specified type and compiled with existing statistics from previous batches.

B.5 Example Application

An example application is given in the file `example.cpp`. This example application takes an extra option `-bits [k]`, where k is an integer no more than the maximum depth specified (m). The solutions are the incidence vectors for all subsets of $[m]$ which have exactly k elements. The augmentation procedure places a 0 or 1 in the next bit of the incidence vector, corresponding to the choice of placing d in the set (0 it is out, 1 it is in) where d is the current depth. The `prune()` method prunes when there are more than k elements selected.

This example should highlight a few nuances when working with the TreeSearch library, specifically how the root node is used to hold the latest label of the first node, and how the `SearchNode` class is extended to include necessary information for the current node so that the `prune()` and `isSolution()` methods do not need to access more than one position in the stack.

B.6 Example Workflow

When managing a distributed search, there are several choices to make as the user. What depth should I generate to? How many jobs should I run? How long should I set the kill time? These questions are answered based on your application and experience. This section guides you through the use of the management scripts as well as strategies for different situations.

B.6.1 Create the Submission Template

The file `condorsubmit.sub` contains the necessary information for submission to the Condor scheduler. The number of processes is automatically managed by the `expandjobs` script. However, you may modify the arguments of the run depending on the type of job you want to run. More on this later.

B.6.2 Generate initial jobs

To create a beginning list of jobs, run a single process in `generate` mode with a reasonably small maximum depth. Send the output to a file called “`out.0`" in order to have it viewed by the `compactjobs` script. To start at the root search node, use the job description “`J 0 0`". The jobs created could be reasonably small.

B.6.3 Compact data

After any run, use the `compactjobs` script to combine the list of results, partial jobs, and new jobs from the output files into a collection of data files. This will also give you the number of jobs which completed, failed, are partial, or are new.

B.6.4 Evaluate Number of Jobs

Based on this number of jobs, you have a few options.

1. You have a lot of jobs ($\geq 20,000$ for instance). This is probably too many to run each job as a single process, so they must be grouped together. When using the `expandjobs` script, use the `-g` flag to group jobs together into processes, so that there are a reasonable number of total processes. For example, if there were $100,000$ jobs, using `expandjobs -g 10` would result in a list of $10,000$ processes. After running `expandjobs`, modify the generated `condorsubmit.sub` script to set the killtime so that all of the jobs in the process can complete. For example, if I want to run the previous list of $10,000$ processes for an hour each, I want to set the per-job killtime to six minutes, or 360 seconds.

   Hopefully, many of your processes will complete in this short time interval, and you can run the remaining processes for a longer period. If this does not occur, and you still have many jobs remaining, perhaps using the `-m` flag on the `expandjobs` script to bound the total number of jobs will allow fewer jobs per process while storing the remaining jobs for execution later.

2. You have a decent number of jobs (between $2,500$ and $20,000$). This is a good number for running each as an individual process. Running `expandjobs`

with no grouping will allow a bijection between processes and jobs. Modifying the generated `condorsubmit.sub` file for a per-process killtime of one hour (3600 seconds) will allow a reasonable amount of computation per job.

3. You have very few jobs (below $2,500$) which did not terminate in an hour of computation. With the number of jobs, just repeating another hour-per-job submission will not take full advantage of parallelism. Use the `expandjobs` script (possibly with grouping) to generate a list of input files for your jobs. Modify the `condorsubmit.sub` script to be in `generate` mode with a maximum depth beyond your current job-depth. Depending on the density of your application's branching, this could be between one to ten or more levels beyond the current job depth. It is usually a good goal to create a large number ($\geq 20,000$) of jobs which can be run with grouping at a small computation time in order to quickly remove small subtrees of the search. This hopefully isolates the "hard cases" that kept the current list of jobs from completing.

B.6.5 Submit Script

Using the `condor_submit` script, submit the processes using the `condorsubmit.sub` submission script. Now, wait for the processes to complete. You can monitor progress by using two Condor commands:

1. `condor_status -submitters` will give you a list of running/idle/held jobs for each submitter. This is a good way to watch your queue when many jobs are running.

2. `condor_q [-submitter username]` or `condor_q [clusterid]` will return a per-process list of run times, statuses, and other useful information.

> This is not recommended when there are many processes running, but when there are less than 100 processes, this can help to find processes that are not completing or when you should expect the processes to complete. Use the `-currentrun` flag to see how long the processes have run since their last eviction or suspension.

If the above methods are not telling you the information you want, view the tail of the log file (as specified in your submission script). This contains the most up-to-date information including the amount of memory each process is using and the reasons for evictions or other failures.

If your processes are not completing, or you have found that you incorrectly set the submission script, remove the jobs using the `condor_rm [clusterid]` command.

B.7 Summary

You should now have the necessary information to develop your own applications using the TreeSearch library. For support questions or bug reports, please email the author at s-dstolee1@math.unl.edu.

B.8 Acknowledgements

This software was developed with the support of the National Science Foundation grant DMS-0914815 under the advisement of Stephen G. Hartke.

The author thanks the faculty and staff at the Holland Computing Center, especially David Swanson, Brian Bockleman, and Derek Weitzel for their extremely helpful advice during the design and development of this library.

Appendix C

ChainCounting User Guide

C.1 Acquiring *ChainCounting*

The latest version of *ChainCounting* and its documentation is available online as part of the *SearchLib* collection at the address

`http://www.math.unl.edu/~s-dstolee1/SearchLib/`

ChainCounting is made available open-source under the GPL 3.0 license.

To complile *ChainCounting*, use a terminal to access the `ChainCounting/src/` folder and type `make`. The executables will be placed in `ChainCounting/bin/`

C.1.1 Acquiring Necessary Libraries

There is one *SearchLib* project used by *ChainCounting*.

1. *TreeSearch* is a project in *SearchLib* that abstracts the structure of a backtrack search in order to allow for parallelization. *TreeSearch* is available on the same web site as *ChainCounting*. Consult the *TreeSearch* documentation [122] for details about the arguments and execution processes.

C.1.2 Full Directory Structure

For proper compilation, place the different dependencies in the following directory structure:

- `SearchLib/` – The *SearchLib* collection.
  - `ChainCounting/` – The *ChainCounting* project.
    - `bin/` – The final binaries are placed here.
    - `docs/` – This folder contains documentation.
    - `src/` – Contains source code. Compilation occurs here.
  - `TreeSearch/` – A support project from *SearchLib*.

C.2 Execution

The *ChainCounting* project uses a single executable: `chains.exe`. This executable evaluates a given formula $f_C(\mathbf{a};\mathbf{b})$ for some configuration C of a certain size. These formulas are hard-coded into the source files, but they were generated automatically using the methods described in [**?**].

Appendix D

Progressions User Guide

D.1 Acquiring *Progressions*

The latest version of *Progressions* and its documentation is available online as part of the *SearchLib* collection at the address

`http://www.math.unl.edu/~s-dstolee1/SearchLib/`

Progressions is made available open-source under the GPL 3.0 license.

To complile *Progressions*, use a terminal to access the `Progressions/src/` folder and type `make`. The executables will be placed in `Progressions/bin/`

D.1.1 Acquiring Necessary Libraries

There are two *SearchLib* projects used by *Progressions*.

1. *TreeSearch* is a project in *SearchLib* that abstracts the structure of a backtrack search in order to allow for parallelization. *TreeSearch* is available on the same web site as *Progressions*. Consult the *TreeSearch* documentation for details about the arguments and execution processes.

2. *Utilities* is a project in *SearchLib* containing useful objects and functions necessary by other projects in *SearchLib*. *Utilities* is available on the same web site as *Progressions*.

D.1.2 Full Directory Structure

For proper compilation, place the different dependencies in the following directory structure:

- `SearchLib/` – The *SearchLib* collection.
    - `Progressions/` – The *Progressions* project.
        - `bin/` – The final binaries are placed here.
        - `docs/` – This folder contains documentation.
        - `src/` – Contains source code. Compilation occurs here.
    - `TreeSearch/` – A support project from *SearchLib*.
    - `Utilities/` – A support project from *SearchLib*.
        - `src/` – Type `make` in this directory to compile the *Utilities* project.

D.2 Execution

The main executable is `progressions.exe`.

D.2.1 Progessions-Specific Arguments

- `-mode [quasi|pseudo]` — Select which type of progression to avoid: Quasi-Arithmetic or Pseudo-Arithmetic.
- `-R #` — The number of colors to use (default: 2).

- `-n #` — The minimum length of a good coloring to report. Will be used by constraint propagation to prune the search space.
- `-N #` — The maximum number of elements to color. Propagation and coloring will not extend beyond this value.
- `-K #` — The length of the progressions.
- `-D #` — The diameter of the progressions.
- `-I #` — The diameter as $d = k - i$. (**W**arning: Must follow the argument of `-K #`).
- `-skew-symmetric` — If present, the colorings will be restricted to skew-symmetric colorings. In this case, the colorings span $\{-n, \ldots, -1, 0, 1, \ldots, n-1\}$.
- `-backward [on|off]` — Specify if the backward propagation should be enabled.
- `-forward [on|off]` — Specify if the forward propagation should be enabled. If enabled, the backward propagation will be enabled as well.

Appendix E

EarSearch User Guide

E.1 Introduction

The EarSearch library implements the generation algorithm of [92] to generate families of 2-connected graphs. It is based on the TreeSearch library [122]. The class `EarSearchManager` extends the class `SearchManager` and manages the search tree, using ear augmentations to generate children. It automates the canonical deletion selection in order to remove isomorphs.

E.2 Acquiring *EarSearch*

The latest version of *EarSearch* and its documentation is available online as part of the *SearchLib* collection at the address

`http://www.math.unl.edu/~s-dstolee1/SearchLib/`

E.2.1 Acquiring Necessary Libraries

There are two *SearchLib* projects and an external library used by *EarSearch*.

1. *TreeSearch* is a project in *SearchLib* that abstracts the structure of a backtrack search in order to allow for parallelization. *TreeSearch* is available on the same web site as *EarSearch*. Consult the *TreeSearch* documentation for details about the arguments and execution processes.
2. *Utilities* is a project in *SearchLib* containing useful objects and functions necessary by other projects in *SearchLib*. *Utilities* is available on the same web site as *EarSearch*.
3. *nauty* performs isomorphism and automorphism calculations. *nauty* was written by Brendan McKay and is available at

   `http://cs.anu.edu.au/~bdm/nauty/`

E.2.2 Full Directory Structure

For proper compilation, place the different dependencies in the following directory structure:

- `SearchLib/` – The *SearchLib* collection.
  - `EarSearch/` – The *EarSearch* project.
    - `bin/` – The final binaries are placed here.
    - `docs/` – This folder contains documentation.
    - `src/` – Contains source code. Compilation occurs here.
  - `TreeSearch/` – A support project from *SearchLib*.
  - `Utilities/` – A support project from *SearchLib*.
    - `src/` – Type `make` in this directory to compile the *Utilities* project.
  - `nauty/` – The *nauty* library must be placed and compiled here.

E.3 Data Management

E.3.1 Graphs

Graphs are stored using the `sparsegraph` structure from the `nauty` library.

During the course of computation, these graphs are modified using edge and vertex deletions. To delete the ith vertex, set the `v` array to -1 in the ith position. To delete the edge between the i and j vertices, set the `e` array to -1 in two places: in the list of neighbors for i where j was listed and in the list of neighbors for j where i was listed. To place the vertices or edges back, place the previous values into those places.

E.3.2 Augmentations and Labels

The labels for each augmentation use two 32-bit integers. The first is the order of the augmented ear. The second is the index of the pair orbit which is used for the endpoints of the ear.

E.3.3 `EarNode`

Each level of the search tree is stored in a stack, where all data is stored in an `EarNode` object. All of the members of `EarNode` are public, in order to easily add data structures and flags that are necessary for each application. All pointers are initialized to 0 in the constructor and are checked to be non-zero before freeing up any memory in the destructor.

The core data necessary for `EarSearchManager` is stored in the following members:

- ear_length – the length of the augmented ear.

- ear – the byte-array description of the augmented ear.
- num_ears – the number of ears in the graph.
- ear_list – the list of ears in the graph (-1 terminated).
- graph – the graph at this node.
- max_verts – the maximum number of vertices in all supergraphs. Default to max_n from `EarSearchManager`.
- reconstructible – TRUE if detectably reconstructible
- numPairOrbits – the number of pair orbits for this graph.
- orbitList – the list of orbits, in a an array of arrays. Each array `orbitList[i]` contains pair-indices for pairs in orbit and is terminated by `-1`.
- canonicalLabels – the canonical labeling of the graph, stored as an integer array of values for each vertex
- solution_data – the data of a solution on this node.
- violatingPairs – A set of pair indices which cannot be endpoints of an ear.

E.4 Pruning

The interface `PruningAlgorithm` has an abstract method for pruning nodes of the search tree. The method `checkPrune` takes two `EarNode` objects: one for the parent and another for the child. Using this data, the method decides if no solution exists by augmenting beyond the child node. Since the pruning algorithm is called before the canonical deletion algorithm, this can also remove nodes which cannot possibly be canonical augmentations.

E.5 Canonical Deletion

The interface `EarDeletionAlgorithm` has an abstract method for finding a canonical ear deletion. The method `getCanonical` takes two `EarNode` objects for the parent and child and returns the array corresponding to the canonical ear. The `EarSearchManager` will determine if this canonical ear is in orbit with the augmented ear.

E.6 Solutions

The interface `SolutionChecker` is an abstract class which contains methods for finding solutions given a search node, storing the solution data, reporting on these solutions, and reporting application-specific statistics.

The method `isSolution` takes the parent, child, and depth and reports if there is a solution at the child node. It returns a non-null string if and only if there is a solution, and that string is a buffer containing the solution data. This buffer will be deallocated with `free()` by the `EarSearchManager`.

The method `writeStatisticsData()` returns a string of statistics (using the TreeSearch format) to be reported at the end of a job.

E.7 Example 0: 2-Connected Graphs

To enumerate all 2-connected graphs, the interfaces were implemented to only prune by number of vertices and possibly by number of edges. The search space is defined by three inputs: $N, e_{\min}$, and $e_{\max}$. These implementations are give by the following classes:

- `EnumeratePruner` will prune a graph if it has more than N vertices or more than $e_{\max}$ edges. Also, if $e(G) + (N - n(G) + 1) > e_{\max}$, it will prune since we cannot add the remaining $N - n(G)$ edges without surpassing $e_{\max}$ edges.
- `EnumerateDeleter` implements the default deletion algorithm: over all ears e in G so that $G - e$ is 2-connected, find one of minimum length, then use the canonical labels to select the canonical ear.
- `EnumerateChecker` detects “solutions" as any graph with exactly N vertices and between $e_{\min}$ and $e_{\max}$ edges.

E.8 Example 1: Unique Saturation

The input consists of two numbers r and N, and we are searching for uniquely K_r-saturated graphs of order N. The unique saturation problem utilizes the deletion algorithm in `EnumerateDeleter`, but adds some data to `EarNode` in order to track the constraints. The `SaturationAlgorithm` class implements both the `PruningAlgorithm` and `SolutionChecker` interfaces.

Note: The `SaturationAlgorithm` class is implemented only for $r \in \{4, 5, 6\}$ in order to use compiler optimizations for the nested loop structure.

E.8.1 Application-Specific Data

The following fields were added to `EarNode` for tracking constraints during the search. Most information is tracked in `adj_matrix_data`, which stores information as an adjacency matrix. The others are boolean flags which mark different properties of the current graph. These flags are set during the `checkPrune` method, and are accessed by the `isSolution` method.

- `adj_matrix_data` – Data on the (directed) edges. For unique saturation, this gives -1 for edges, and for non-edges counts the number of copies of H given by adding that edge. Values are in $\{0, 1, 2\}$, since when 2 is listed, then there are too many copies of H.
- `any_adj_zero` – A boolean flag: are any of the cells in adj_matrix_data zero?
- `any_adj_two` – A boolean flag: are any of the cells in adj_matrix_data at least two?
- `dom_vert` – A boolean flag: is there a dominating vertex?
- `copy_of_H` – A boolean flag: is there a copy of H?

E.9 Example 2: Edge Reconstruction

The Edge Reconstruction application takes an integer N and searches over all 2-connected graphs of order up to N and up to $1 + \log_2 N!$ edges. The deletion is built to make graphs with the same deck be siblings. Then, all siblings which are not detectably edge reconstructible are checked to have different edge decks.

The following three classes implement the interfaces:

- `ReconstructionPruner` implements the `PruningAlgorithm` interface and prunes any graph with more than N vertices or more than $1 + \log_2 N!$ edges.
- `ReconstructionDeleter` implements the `EarDeletionAlgorithm` interface and performs two different deletions:

1. If the graph is detectably edge reconstructible, the deletion can be independent of the application and utilizes the standard deletion algorithm from `EnumerateDeleter`.
2. If the graph is NOT detectably edge reconstructible, the canonical ear is selected by using only the edge deck. Further, if the deletion is canonical, the graph is stored in the parent `EarNode` for later comparison of edge decks. The `GraphData` class was implemented specifically for storing these children within the parent `EarNode`.

- `ReconstructionChecker` implements the `SolutionChecker` interface and compares the current graph's edge deck against all previous siblings. This is done using three levels of comparison, which are implemented in the `GraphData` class.

E.9.1 Application-Specific Data

The `GraphData` class stores all information for a child graph. It implements three levels of comparison, which are checked in order within the `compare` method.

1. `computeDegSeq` computes and stores the standard degree sequence for the current graph.
2. `computeInvariant` calculates and stores a more complicated function based on the degree sequence and the degrees of the neighborhood for each vertex.
3. `computeCanonStrings` computes canonical strings for every edge-deleted subgraph and sorts the list. These are then compared, card-for-card.

In order to store these `GraphData` objects, the following members were added to the `EarNode` class:

- child_data – the GraphData objects for immediate children, used for pairwise comparison.
- num_child_data – the number of GraphData objects currently filling the data.
- size_child_data – the number of pointers currently allocated.

E.10 Example 3: p-Extremal Graphs

This problem is investigated in [123] and is the most involved of all applications. See [38] and [63] for background on this problem. The input is given as $P_{\min}$, $P_{\max}$, C, and N. The search is for elementary graphs with p perfect matchings (for $P_{\min} \leq p \leq P_{\max}$) with excess at least C and at most N vertices. The search actually runs over 1-extendable and almost 1-extendable graphs, which are the graphs reachable by the ear augmentations. A second stage adds forbidden edges to maximize excess without increasing the number of perfect matchings.

The following classes implement the EarSearch interfaces:

- `MatchingPruner` implements the `PruningAlgorithm` interface. Graphs are pruned for three reasons:

    1. There are an odd number of vertices. By the Lovász Two Ear Theorem, we know that every ear augmentation has an even number of internal vertices.
    2. There are more than $P_{\max}$ perfect matchings.
    3. The parent graph was not 1-extendable, and neither is the current graph. By the Lovász Two Ear Theorem, we can always go from 1-extendable to 1-extendable using at most two ear augmentations.

4. Let c be the maximum excess of an elementary supergraph of the current graph, which is of order n, and let p be the current number of perfect matchings. If $c + 2(P_{\max} - p) - \frac{1}{4}(n' - n)(n - 2) < C$ for all $n \leq n' \leq N$, then prune. Otherwise, maximize the n' so that the inequality $c + 2(P_{\max} - p) - \frac{1}{4}(n' - n)(n - 2) \geq C$ holds. That value of n' is then used to bound the length of future ear augmentations, since no graph reachable from the current graph can have excess at least C and more than n' vertices.

In addition to pruning, the pruning algorithm also performs the on-line algorithm for updating the list of barriers by using the current ear augmentation.

- `MatchingChecker` implements the `SolutionChecker` interface. Given a 1-extendable graph with between $P_{\min}$ and $P_{\max}$ perfect matchings, forbidden edges are added in all possible ways and the elementary supergraphs with excess at least C are printed to output. If any are found, the `isSolution` method returns with success. The algorithm for enumerating all elementary supergraphs is implemented in the `BarrierSearch.cpp` file.
- `MatchingDeleter` implements the `EarDeletionAlgorithm` interface. The following sequence of choices describe the method for selecting a canonical ear to delete from a graph H:

1. If H is almost 1-extendable, we need to delete an ear e' so that $H - e'$ is 1-extendable. By the definition of almost 1-extendable, there is a unique such choice.
2. If H is 1-extendable, check if there exists an ear e' so that $H - e'$ is 1-extendable. If one exists, select one of minimum length and break ties

using the canonical labels of the endpoints.

3. If H is 1-extendable and no single ear e' makes $H - e'$ 1-extendable, then find an ear e so that there is a disjoint ear f with $H - e$ is almost 1-extendable and $H - e - f$ is 1-extendable. Out of these choices for e, choose one of minimum length and break ties using the canonical labels of the endpoints.

E.10.1 Application-Specific Data

The following members were added to `EarNode` to help the perfect matchings application.

- `extendable` – A boolean flag: is the graph 1-extendable?
- `numMatchings` – The number of perfect matchings for this graph.
- `barriers` – The list of barriers of the graph, given as an array of `Set` pointers. This barrier list is updated at each level by an on-line algorithm.
- `num_barriers` – the number of barriers in the graph.

E.10.2 Perfect Matching Algorithms

There are a few algorithms that are implemented in order to solve certain subproblems, such as counting perfect matchings or enumerating independent sets. These are computationally complex problems, but the implementations are very fast for these small instances. The algorithms are mostly un-optimized and rely on simple instructions and low overhead in order to be run many many times during the course of the search.

- `countPM`(G, P) counts the number of perfect matchings in a graph G, with an upper bound of P. It operates recursively, selecting an edge e in G and attempts to extend the current matching using e and not using e. When a perfect matching is found, the counter increases. There are two shortcutting strategies:

  1. If there is ever a vertex with no available edges, the recursion is halted with a count of zero perfect matchings, since the current matching does not extend to a perfect matching.
  2. If the current count of perfect matchings ever surpasses P, then the current value is returned. During the search, we only care about graphs with at most P_{max} perfect matchings, so graphs with many more will only be pruned.

- `isExtendable`(G) tests if the given graph is 1-extendable. This is done by storing an array of boolean flags for each edge, marking each as they are found to be in perfect matchings. This algorithm is explicitly used in the deletion algorithm. During the pruning algorithm, where a specific augmentation is given, we can detect 1-extendability by asking if there is a perfect matching using the proper alternating path within the augmented ear.

- `enumerateAllBarrierExtensions`($G, \mathcal{B}, C$) and `searchAllBarrierExtensions`($G, \mathcal{B}$) are two methods which take a 1-extendable graph G with barrier list $\mathcal{B}$ and attempts to add forbidden edges to G to attain the maximum excess. The difference is that `enumerateAllBarrierExtens` will output any graphs with excess at least C, while `searchAllBarrierExtensions` will simply return the largest excess. The algorithm essentially enumerates all independent sets within the barrier conflict graph $\mathcal{B}$, where conflicts are computed on the fly. The enumeration is recursive, simply testing if the next available barrier should be added to the current independent set. As each set is added, it tests which barriers with larger index are in conflict with this graph. These barriers are then not considered in deeper recursive calls. Due to the low overhead for each independent set, this simple algorithm runs fast enough for the search to be feasible.

Appendix F

Saturation User Guide

F.1 Acquiring *Saturation*

The latest version of *Saturation* and its documentation is available online as part of the *SearchLib* collection at the address

`http://www.math.unl.edu/~s-dstolee1/SearchLib/`

Saturation is made available open-source under the GPL 3.0 license.

To compile *Saturation*, use a terminal to access the `Saturation/src/` folder and type `make`. The executables will be placed in `Saturation/bin/`

F.1.1 Acquiring Necessary Libraries

There are two external libraries and two *SearchLib* projects used by *Saturation*.

1. *nauty* performs isomorphism and automorphism calculations. *nauty* was written by Brendan McKay [93] and is available at

   `http://cs.anu.edu.au/~bdm/nauty/`

2. *cliquer* performs clique calculations, including finding the clique number and counting the number of cliques. *cliquer* was written by Niskanen and Östergård [100] and is available at

   `http://users.tkk.fi/pat/cliquer.html`

3. *TreeSearch* is a project in *SearchLib* that abstracts the structure of a backtrack search in order to allow for parallelization. *TreeSearch* is available on the same web site as *Saturation*. Consult the *TreeSearch* documentation for details about the arguments and execution processes.

4. *Utilities* is a project in *SearchLib* containing useful objects and functions necessary by other projects in *SearchLib*. *Utilities* is available on the same web site as *Saturation*.

F.1.2 Full Directory Structure

For proper compilation, place the different dependencies in the following directory structure:

- `SearchLib/` – The *SearchLib* collection.
    - `Saturation/` – The *Saturation* project.
        - `bin/` – The final binaries are placed here.
        - `docs/` – This folder contains documentation.
        - `src/` – Contains source code. Compilation occurs here.
    - `TreeSearch/` – A support project from *SearchLib*.
    - `Utilities/` – A support project from *SearchLib*.

    - `src/` – Type `make` in this directory to compile the *Utilities* project.
  - `cliquer/` – The *cliquer* library must be placed and compiled here.
  - `nauty/` – The *nauty* library must be placed and compiled here.

F.2 Execution

There are two executables in the *Saturation* project.

- `saturation.exe` runs an orbital branching search for uniquely K_r-saturated graphs of a given order n.
- `cayley.exe` generates Cayley complements and checks if they are uniquely K_r-saturated for some r.

F.2.1 `saturation.exe`

This executable generates all uniquely K_r-saturated graphs of a given order n. It uses a customized orbital branching approach.

```
saturation.exe [TreeSearch args] -N # -r # [--cliquer]
```

- `-N #` specifies the number n of vertices to use. All uniquely K_r-saturated graphs of order n will be generated.
- `-r #` specifies the value of r to use when searching for uniquely K_r-saturated graphs.
- `--cliquer` is an option that specifies to use the *cliquer* library in the pruning steps of the search. If not specified, the search uses a tabulation method.

F.2.2 `cayley.exe`

This executable generates Cayley complements and checks if they are uniquely K_r-saturated for some r. For a fixed number of generators g, it selects a set $S = \{1 < s_2 < s_3 < \cdots < s_g\}$ and then selects integers n so that $2s_g + 1 \leq n \leq N_{\max}$. Then, it uses

To execute `cayley.exe`, use the following format of arguments:

```
cayley.exe [TreeSearch args] -N # -G # -t # [--verbose]
                              [--dihedral]
```

- `-N #` specifies $N_{\max}$, the maximum value of n to use when searching for a uniquely K_r-saturated Cayley complement $\overline{C}(\mathbb{Z}_n, S)$.
- `-G #` specifies the number of generators to place in the set S.
- `-t #` specifies the number of seconds to allow a call to the *cliquer* library run before terminating. If a call is terminated early, the graph that was being tested is output as a job (using *TreeSearch* job descriptions).
- `--verbose` is an option to output the status of the search while testing a specific Cayley complement. Not recommended for a large-scale search, but only for a long test of a specific example.
- `--dihedral` is an option that checks for uniquely K_r-saturated Cayley complements over the dihedral groups. (*Note:* We have not yet found any generator sets that create uniquely K_r-saturated Cayley complements of dihedral groups.)

F.3 *TreeSearch* Arguments

- `-k #` — The killtime: How many seconds before halting the process and reporting a partial job.
- `-m #` — The maximum depth: the maximum number of steps to go before halting (or in generation mode, a new job is written at this depth).
- `run` — Run mode: The input jobs are run until finished or the killtime is reached.
- `generate` — Generation mode: The input jobs are run and new jobs are listed when reaching the maximum depth.
- `-maxjobs #` — The maximum number of jobs to generate before halting with a partial job (default: 1000).
- `-maxsols #` — The maximum number of solutions to output before halting with a partial job (default: 100).