# Independent Sets and Closed-Shell Independent Sets of Fullerenes

by

**Sean Michael Daugherty**

B.Sc., Clemson University, 2002
M.Sc., Clemson University, 2005

A Dissertation Submitted in Partial Fulfillment
of the Requirements for the Degree of

**DOCTOR OF PHILOSOPHY**

in the Department of Computer Science

Sean Michael Daugherty, 2009

University of Victoria

*Your file Votre référence*
ISBN: 978-0-494-66829-0
*Our file Notre référence*
ISBN: 978-0-494-66829-0

# Supervisory Committee

## Independent Sets and Closed-Shell Independent Sets of Fullerenes

by

**Sean Michael Daugherty**

B.Sc., Clemson University, 2002
M.Sc., Clemson University, 2005

**Supervisory Committee**

Dr. Wendy Myrvold, Supervisor
(Department of Computer Science)

Dr. Frank Ruskey, Departmental Member
(Department of Computer Science)

Dr. Ulrike Stege, Departmental Member
(Department of Computer Science)

Dr. Gary MacGillivray, Outside Member
(Department of Mathematics and Statistics)

# Abstract

**Supervisory Committee**

Dr. Wendy Myrvold, Supervisor
(Department of Computer Science)

Dr. Frank Ruskey, Departmental Member
(Department of Computer Science)

Dr. Ulrike Stege, Departmental Member
(Department of Computer Science)

Dr. Gary MacGillivray, Outside Member
(Department of Mathematics and Statistics)

*Fullerenes* are all-carbon molecules with polyhedral structures where each atom is bonded with three other atoms and the faces of the polyhedron are pentagons and hexagons. *Fullerene graphs* model the fullerene structures and are cubic planar graphs having twelve pentagonal faces and the remaining faces are hexagonal. This work explores two models that seek to determine the maximum number of bulky addends that may bond to the surface of a fullerene.

The first model assumes that any two bulky addends are too large to bond to adjacent carbon atoms. This is equivalent to finding a graph-theoretical *maximum independent set*: a vertex subset of maximum size such that no two vertices are adjacent. The problem of determining the maximum independent set order is NP-hard for general cubic planar graphs and the complexity for the fullerene subclass was previously unknown. By extending the work of Graver, a graph-theoretical foundation is laid then used to derive a linear-time algorithm for solving the maximum independent set problem for fullerenes. A discussion of the relationship between maximum independent sets and some specific families of fullerenes follows.

The second model refines the first by adding an additional requirement that the resulting molecule is stable according to Hückel theory: the molecule exhibits a stable

distribution of $\pi$ electrons. The graph-theoretical description of this model is a *maximum closed-shell independent set*: a vertex subset of maximum size such that no two vertices are adjacent and exactly half of the eigenvalues of the adjacency matrix of the graph that results from the deletion of the vertex subset are positive. Computations for finding a maximum closed-shell independent set rely on determining whether fullerene subgraphs are *closed-shell* (satisfy the eigenvalue requirement) so a linear-time algorithm for finding the *inertia* (number of negative, zero, and positive eigenvalues) of unicyclic graphs is given. This algorithm is part of an exponential-time algorithm for finding a maximum closed-shell independent set of a fullerene molecule that is fast enough for practical use. An improved upper bound of $3n/8 + 3/2$ for the closed-shell independence number is included.

# Table of Contents

# List of Tables

# List of Figures

# List of Algorithms

# Acknowledgments

I wish to acknowledge my supervisor Dr. Wendy Myrvold for her support and endless curiosity of new aspects of my research. She provided numerous suggestions and found many more ideas to explore. She also showed me interesting approaches to developing faster algorithms. Without her, the results provided here would not have been possible. Furthermore, her development of the FuiGui software was invaluable for this work.

I also wish to thank Dr. Patrick Fowler, with whom I collaborated on many occasions during his many visits and emails. He provided helpful discussions and patiently explained and re-explained the chemistry aspects of the project.

Dr. Jack Graver laid the groundwork that enabled the linear-time algorithm for maximum independent sets. I thank him for several discussions related to the work.

I thank Dr. Ermelinda DeLaVina for running her program Graffiti.pc on data output from my programs in order to generate conjectures. One of the conjectures is mentioned in this dissertation.

Dr. Daniel Král provided me with an interesting new perspective on how to improve the lower bound for the independence number of fullerenes. His suggestions are mentioned where the related open problem is discussed.

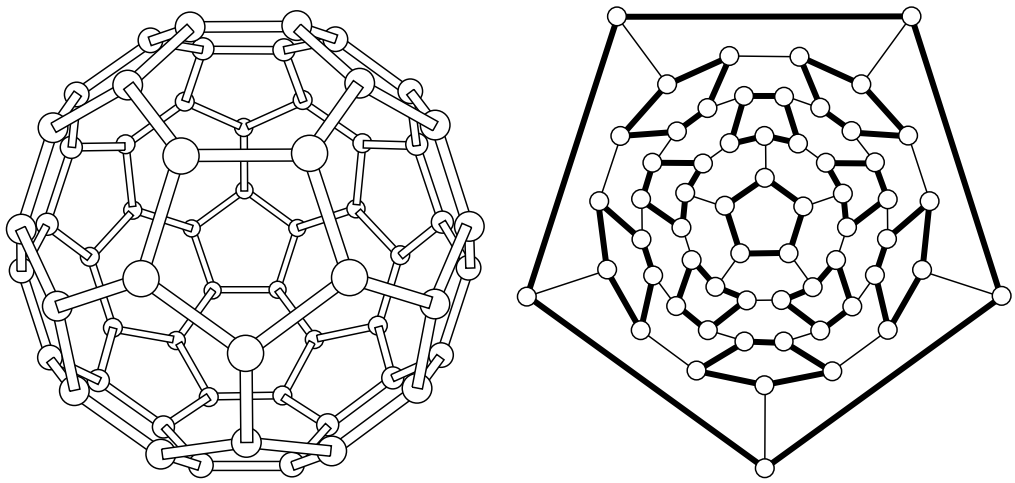
I would also like to thank my wife Rebekah for her support and for prodding me to keep writing when it became tedious.

This research was supported in part by the University of Victoria's fellowship program and indirectly by NSERC through Wendy Myrvold.

# Chapter 1
# Introduction

In 1985, Kroto, Heath, O'Brien, Curl, and Smalley discovered the existence of an all-carbon molecule that was produced as a side effect by an experiment [37]. Previously, the only known all-carbon substances were diamond and graphite. This discovery resulted in the Nobel Prize in chemistry in 1996 for several of the authors. The molecule has 60 carbon atoms with the bond structure shown in Figure 1.1(a) and was given the name $C_{60}$: Buckminsterfullerene after Richard Buckminster Fuller because its structure resembles his geodesic dome. The surface of the polyhedron represented by the structure contains twelve pentagons and twenty hexagons, which may be visualized more clearly by the 2-dimensional model in Figure 1.1(b). This is the same structure as a soccer ball, which commonly has black pentagons and white hexagons.

After its discovery in the lab, Buckminsterfullerene was discovered to occur naturally in soot [25, 1.2] along with similar structures having 70, 76, 78, 82, 84, 90, 94, or 96 carbon atoms. Each of these all-carbon molecules form polyhedral structures in which each of the atoms is directly bonded to its three neighbors and in which all faces of the polyhedron are either pentagonal or hexagonal. The collection of all such all-carbon molecules, whether observed or theoretical, is called the *fullerenes*. A general discussion of theoretical chemistry aspects of fullerene structure, isomerism



(a) A 3D model (b) A 2D model with pentagons in bold

Figure 1.1: Models of the most common fullerene, $C_{60}$: Buckminsterfullerene

and properties is given in [25]. The most common fullerene to occur in nature is Buckminsterfullerene, which is why it is arguably the most commonly studied fullerene.

A fullerene with $n$ carbon atoms is denoted by $C_n$. Fullerenes exist, at least in theory, for any even value of $n$ at least 20, excluding 22. With 60 carbon atoms, there are 1812 isomeric possibilities, but only Buckminsterfullerene can currently be manufactured in significant quantity. The number of fullerene isomers of $C_n$ is shown in Table 1.1 along with the number of isomers that satisfy the isolated pentagon rule (IPR), meaning they have no adjacent pentagons [7]. The number of $C_n$ isomers is in $\Theta(n^9)$ according to [25, p.33] that references [49].

Given a molecular-property requirement for an application of fullerenes, one may seek to determine the best fullerene isomer from those that are naturally available. Should the state-of-the-art of molecular manufacturing advance to the point where arbitrary isomers can be constructed, a method to determine the most relevant isomer from the mass of mathematical possibilities is vital. Evaluation of graph invariants is one strategy for the identification of such isomers. Hence, the development of fast algorithms for enumeration of fullerene isomers [8, 9] and computation of invariants is of practical chemical as well as theoretical computer-science interest. Such calculations produce a large data set of various invariants, which facilitates comparisons and pattern discovery that result in learning previously unknown properties of the fullerenes. Faster algorithms for calculating invariants make it easier to compute data for larger and more fullerenes. This dissertation is primarily focused on invariants (and fast algorithms to compute them) that measure the number of large addends that can simultaneously bond to the surface of the fullerene. First, Chapter 2 gives precise definitions to the terms used throughout the dissertation.

The majority of the work presented here is concerned with determining how many molecules (possibly single atoms) that are sufficiently larger than a carbon atom can bond to the surface of a fullerene. Such molecules are called *bulky addends* and under the models studied here, their size imposes a restriction that no two bulky addends may bond to adjacent carbon atoms. A popular bulky addend is bromine and the problem can be phrased as: what is the maximum number of bromine atoms that may bond to the surface of a fullerene? Chapter 3 explores this problem, including a discussion of what is known mathematically and adds some new results. The new results are used in Chapter 4 to create a linear-time algorithm that answers the question for any given fullerene, when previously only exponential-time algorithms were known. Chapter 5 discusses some theoretical results regarding the relationship

between some specific fullerene families and the maximum number of bulky addends that can bond to the surface of the fullerenes in the families.

Each carbon atom in a fullerene molecule has four valence electrons. Three of the valence electrons are used in $\sigma$ bonds, one for each of the atom's three neighbors. In chemical terms, fullerene molecules are *unsaturated*: according to Hückel theory, the fourth valence electron of each carbon atom is used to form a delocalized $\pi$ system on the surface of the polyhedron. That is, these fourth valence electrons are shared by the entire molecule in molecular orbitals.

Computations involving the bond structure of a hydrocarbon molecule indicate whether or not a molecule is stable by considering the molecular energy levels given by the $\pi$ system. The Hückel molecular orbital theory was devised for molecules that are two-dimensional, as opposed to the curved surface of a fullerene's cage structure. Due to this, Hückel theory does not work very well for predicting the stability of fullerenes in general, especially those that are not highly symmetrical [25, Ch. 4]. Other methods have been suggested in order to predict the stability of a fullerene. Among these is the invariant that counts the maximum number of bulky addends that may bond to the surface of a fullerene (the maximum independent set order) [21]. However, algorithms like the one presented here were used to compute the invariant for over ten million fullerenes, providing the necessary evidence to refute the suggestion [23].

When a bulky addend like bromine bonds to a carbon atom, the bond uses the fourth valence electron of the carbon atom and therefore the atom is no longer present in the fullerene's $\pi$ system. When a large number of addends have bonded at different locations on the fullerene's surface, the $\pi$ system of the fullerene is broken into smaller disjoint pieces composed of the subgraphs induced by the carbon atoms that are not bonded to addends. From Hückel theory, the fullerene molecule with addends is stable if each of these subgraphs is stable. (To determine if a piece is stable, a calculation involving the molecular energy levels is performed.) A natural question that arises is: if the maximum number of bulky addends are bonded to the surface of a fullerene, is the resulting molecule stable? This question is explored in Chapter 7 and the answer is a resounding "no" for the vast majority of fullerenes. This begets the question: what is the maximum number of bulky addends that can bond to the surface of a fullerene for which the result is stable? Chapter 7 gives a fast exponential-time algorithm to compute the answer to this question after some related work is established in Chapter 6.

| $n$ | Fullerenes | IPR | $n$ | Fullerenes | IPR | $n$ | Fullerenes | IPR |
|---|---|---|---|---|---|---|---|---|
| 20 | 1 | 0 | 94 | 153 493 | 134 | 168 | 41 478 338 | 1 628 029 |
| 22 | 0 | 0 | 96 | 191 839 | 187 | 170 | 46 088 148 | 1 902 265 |
| 24 | 1 | 0 | 98 | 231 017 | 259 | 172 | 51 809 018 | 2 234 133 |
| 26 | 1 | 0 | 100 | 285 914 | 450 | 174 | 57 417 255 | 2 601 868 |
| 28 | 2 | 0 | 102 | 341 658 | 616 | 176 | 64 353 257 | 3 024 383 |
| 30 | 3 | 0 | 104 | 419 013 | 823 | 178 | 71 163 435 | 3 516 365 |
| 32 | 6 | 0 | 106 | 497 529 | 1 233 | 180 | 79 538 725 | 4 071 832 |
| 34 | 6 | 0 | 108 | 604 217 | 1 799 | 182 | 87 738 289 | 4 690 880 |
| 36 | 15 | 0 | 110 | 713 319 | 2 355 | 184 | 97 841 157 | 5 424 777 |
| 38 | 17 | 0 | 112 | 860 161 | 3 342 | 186 | 107 679 684 | 6 229 550 |
| 40 | 40 | 0 | 114 | 1 008 444 | 4 468 | 188 | 119 761 030 | 7 144 091 |
| 42 | 45 | 0 | 116 | 1 207 119 | 6 063 | 190 | 131 561 725 | 8 187 581 |
| 44 | 89 | 0 | 118 | 1 408 553 | 8 148 | 192 | 145 976 654 | 9 364 975 |
| 46 | 116 | 0 | 120 | 1 674 171 | 10 774 | 194 | 159 999 441 | 10 659 863 |
| 48 | 199 | 0 | 122 | 1 942 929 | 13 977 | 196 | 177 175 662 | 12 163 298 |
| 50 | 271 | 0 | 124 | 2 295 721 | 18 769 | 198 | 193 814 634 | 13 809 901 |
| 52 | 437 | 0 | 126 | 2 650 866 | 23 589 | 200 | 214 127 713 | 15 655 672 |
| 54 | 580 | 0 | 128 | 3 114 236 | 30 683 | 202 | unknown | 17 749 388 |
| 56 | 924 | 0 | 130 | 3 580 637 | 39 393 | 204 | unknown | 20 070 486 |
| 58 | 1 205 | 0 | 132 | 4 182 071 | 49 878 | 206 | unknown | 22 606 939 |
| 60 | 1 812 | 1 | 134 | 4 787 715 | 62 372 | 208 | unknown | 25 536 557 |
| 62 | 2 385 | 0 | 136 | 5 566 948 | 79 362 | 210 | unknown | 28 700 677 |
| 64 | 3 465 | 0 | 138 | 6 344 698 | 98 541 | 212 | unknown | 32 230 861 |
| 66 | 4 478 | 0 | 140 | 7 341 204 | 121 354 | 214 | unknown | 36 173 081 |
| 68 | 6 332 | 0 | 142 | 8 339 033 | 151 201 | 216 | unknown | 40 536 922 |
| 70 | 8 149 | 1 | 144 | 9 604 410 | 186 611 | 218 | unknown | 45 278 722 |
| 72 | 11 190 | 1 | 146 | 10 867 629 | 225 245 | 220 | unknown | 50 651 799 |
| 74 | 14 246 | 1 | 148 | 12 469 092 | 277 930 | 222 | unknown | 56 463 948 |
| 76 | 19 151 | 2 | 150 | 14 059 173 | 335 569 | 224 | unknown | 62 887 775 |
| 78 | 24 109 | 5 | 152 | 16 066 024 | 404 667 | 226 | unknown | 69 995 887 |
| 80 | 31 924 | 7 | 154 | 18 060 973 | 489 646 | 228 | unknown | 77 831 323 |
| 82 | 39 718 | 9 | 156 | 20 558 765 | 586 264 | 230 | unknown | 86 238 206 |
| 84 | 51 592 | 24 | 158 | 23 037 593 | 697 720 | 232 | unknown | 95 758 929 |
| 86 | 63 761 | 19 | 160 | 26 142 839 | 836 497 | 234 | unknown | 105 965 373 |
| 88 | 81 738 | 35 | 162 | 29 202 540 | 989 495 | 236 | unknown | 117 166 528 |
| 90 | 99 918 | 46 | 164 | 33 022 572 | 1 170 157 | 238 | unknown | 129 476 607 |
| 92 | 126 409 | 86 | 166 | 36 798 430 | 1 382 953 | 240 | unknown | 142 960 479 |

Table 1.1: From [7], the number of theoretically possible fullerene isomers with $n$ atoms and the number of which satisfy the isolated pentagon rule (IPR).

# Chapter 2
# Background

Graph theory has a plethora of applications in chemistry because atoms may be represented by vertices and bonds by edges of a graph. In this way, fullerenes can be studied in a graph theoretical context. For precision and clarity, this chapter gives the graph theoretical definitions that are used throughout this dissertation. Then properties of the fullerene graphs are discussed.

## 2.1 Graph Theory

A *graph* $G$ consists of a set of *vertices* $V$ and a multi-set of *edges* $E$ such that each edge connects two vertices. A *simple graph* is a graph with no loops or multiple edges. That is, each edge $(u, v)$ connects two different vertices $u$ and $v$ and each pair of vertices has at most one edge connecting them. Two vertices are *adjacent* if an edge connects them and two edges are *adjacent* if they share a common vertex. A vertex and an edge are *incident* if the vertex is an endpoint of the edge. The *degree* of a vertex is the number of edges incident with it.

A *path* is an alternating sequence of not-necessarily-distinct vertices and edges of the form $v_0$, $(v_0, v_1)$, $v_1$, $(v_1, v_2)$, ..., $v_{k-1}$, $(v_{k-1}, v_k)$, $v_k$. A *connected graph* is a graph in which a path exists between each pair of vertices. The graphs in this dissertation are assumed to be simple and connected.

A *directed graph* consists of a set of vertices and a multi-set of *arcs* where each arc represents an ordered pair of vertices. An arc $(u, v)$ (not necessarily unique) is said to *exit* $u$ and *enter* $v$.

A *walk* is an alternating sequence of vertices and arcs of the form $v_0$, $(v_0, v_1)$, $v_1$, $(v_1, v_2)$, ..., $v_{k-1}$, $(v_{k-1}, v_k)$, $v_k$. A walk is directed and has a *start vertex* $v_0$, an *end vertex* $v_k$, and *internal vertices* $v_1$ through $v_{k-1}$. Note that a vertex or arc may appear more than once in a walk. A *circuit* is a closed walk (a walk in which $v_0 \equiv v_k$) without a specified start/end vertex.

The directed graphs are also assumed to be simple and connected. That is, each ordered pair of vertices appears in the set of edges at most once and there is a walk from any vertex to any other vertex.

An *embedding* is a mapping of a graph into a surface, drawn such that the edges do not cross. The surface of interest here is the plane. A *planar graph* is a graph that can be drawn on the plane with no edge crossings.

A *rotation system* of a directed graph is an adjacency list where the arcs associated with each vertex are listed in clockwise order according to a given embedding. A rotation system of an undirected graph is a rotation system of its corresponding directed graph, which has two arcs $(u, v)$ and $(v, u)$ for each edge $(u, v)$ of the undirected graph.

A *dual graph* $G^*$ of a planar graph $G$ has a vertex for each face of $G$ and two vertices of $G^*$ are joined by an edge if and only if the corresponding faces of $G$ share an edge. If $G^*$ is a dual graph of some graph $G$, then $G$ is called the *primal graph*. The dual graph of a dual graph is the primal graph.

## 2.2 Fullerenes

Recall that fullerenes are all-carbon molecules in which each atom bonds with three other atoms to form a polyhedron structure with face sizes five or six. The structure of a fullerene molecule can be represented by a graph by representing each atom by a vertex and joining two vertices with an edge if the corresponding atoms are bonded to one another. This yields the following definition of a fullerene graph.

**Definition 2.2.1** (Fullerene)**.** *A* fullerene graph*, henceforth* fullerene*,* $F = (V, E)$ *with vertex set* $V$ *and edge set* $E$ *is a 3-regular (all vertices have degree three) planar graph in which each face has size either five or six.*

Fullerenes are 3-connected (at least 3 vertices must be deleted to disconnect the graph). Therefore, each fullerene has a unique embedding in the plane (up to preserving face boundaries) [54], so the face sizes are uniquely defined.

Because fullerenes are 3-regular graphs, a fullerene on $n$ vertices has $3n/2$ edges. Euler's polyhedron formula [20] can be used to show that every fullerene contains exactly 12 pentagons and $n/2 - 10$ hexagons for $n/2 + 2$ faces in total. Therefore, a fullerene dual graph contains $n/2 + 2$ vertices, $3n/2$ edges, and $n$ faces. Because fullerenes are 3-regular, each face of the dual graph is a triangle.

A *face spiral* is an ordering of the faces of a fullerene such that each face in the spiral shares an edge with its immediate predecessor and successor and each face in the spiral after the second shares an edge with both (a) its immediate predecessor in the spiral and (b) the first face in the preceding spiral that still has an open edge [25, p.24]. Figure 2.1 shows an example of a face spiral. A *face spiral sequence* of a fullerene with $n$ atoms is a list of twelve unique integers that correspond with the zero-based positions of the twelve pentagons in the face spiral order. The face spiral sequence of the example is also given in Figure 2.1.

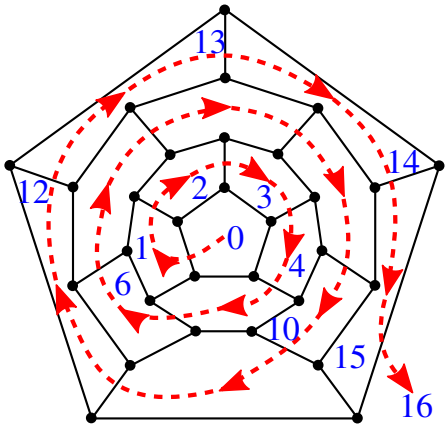



Figure 2.1: A fullerene with 30 atoms shown with a face spiral and face spiral sequence $0, 1, 2, 3, 4, 6, 10, 12, 13, 14, 15, 16$

Due to the existence of multiple isomers for values of $n > 26$ illustrated by Table 1.1, the notation $C_n$ is ambiguous because $n$ does not uniquely characterize the structure of the fullerene. A face spiral uniquely defines a fullerene isomer and isomer numbers are determined by sorting the fullerenes based on their lexicographically smallest face spiral sequence [25, 2.6]. To specify the structure of the fullerene, one provides the isomer number $k \geq 1$ with the notation $C_n{:}k$. It should be noted that not all fullerenes have a face spiral, though the smallest known fullerene without a face spiral has 380 vertices [25, 2.8].

The algorithms presented in Chapters 4 and 7 were run on all fullerenes with 120 or fewer (Chapter 4) and 100 or fewer vertices (Chapter 7). This required the generation of all such fullerenes, which was performed by running Brinkmann's fullgen program [7]. The fullerenes were then sorted by the isomer numbers.

# Chapter 3
# Maximum Independent Sets of Fullerenes

Applicability of fullerenes in functional compounds, materials and devices require the ability to tune stability and properties. Graph invariants have a part to play in rationalization of the chemistry of fullerene derivatives. Even the simplest addition reactions—of atoms of a single type X to a single fullerene $C_n$—generate a combinatorial explosion of possible isomers $C_nX_q$ differing in the number and placing of addends, and may lead to difficult experimental problems of separation and characterization. One way to avoid such problems may be to drive the reaction to completion, when the product distribution may become simpler, and the question then arises [24, 26]: what is the maximum number of addends of a given type that will ultimately attach to a given fullerene, if the addends are too bulky to occupy neighboring carbon sites? To answer this question, the graph theoretical model of *independent sets* (sometimes called *stable sets*) is considered as follows.

**Definition 3.0.2** (Independent Set)**.** *On a graph $F = (V, E)$, a vertex subset $S \subseteq V$ is called an* independent set *if $\forall u, v \in S$, $(u, v) \notin E$.*

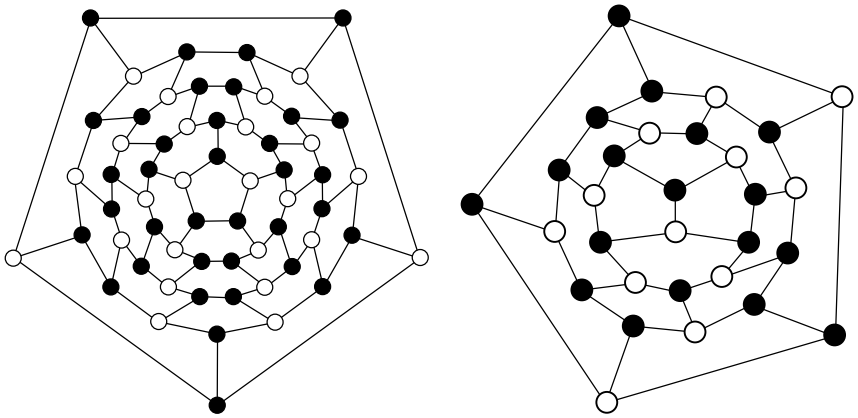
A largest such set is a *maximum independent set*. The following definition gives a corresponding invariant for a graph.

**Definition 3.0.3** (Independence Number)**.** *The* independence number *of a graph $F$ is:*

$$\alpha(F) = \max_{S \subseteq V} |S|$$

*where $S$ is an independent set.*

Under the described model in which the only restriction on addition is that no two bulky addends occupy neighboring carbon sites, the answer to the posed question will be represented by taking the X positions to be a maximum independent set of the vertices of the fullerene graph. The stoichiometry $C_nX_q$ will reflect the independence number $q$ of the graph. Figure 3.1 shows two examples of a maximum independent set.



(a) $C_{60}$:1812 (b) $C_{30}$:2

Figure 3.1: Maximum independent sets (white vertices) of two fullerenes

Although it was proposed in the literature that minimization of the independence number is a predictor of bare fullerene stability [21], it has since been shown that the independence number and fullerene stability are poorly correlated [23, 26]. Nevertheless, the problem of calculating the independence number remains interesting from a graph-theoretical perspective and provides insight about fullerene derivatives.

Independent sets also have theoretical interest. The problem of finding a maximum independent set of a graph has been studied extensively, including [12, 43, 46], as well as in the form of its sister problem, the maximum clique problem [6, 35]. (A maximum independent set corresponds to a maximum clique in the complement graph.) The problem of finding the independence number is $NP$-Hard for cubic planar graphs [28, A1.GT20], a class that contains fullerenes.

This made the question of the complexity of computing $\alpha(F)$ for fullerenes intriguing. A lower bound on the independence number of $3n/8$ has been shown for triangle-free cubic planar graphs [34], a category that includes fullerenes. An upper bound on the independence number of fullerenes is $n/2 - 2$ [27]. An exponential-time backtracking algorithm has been used to calculate $\alpha(F)$ for the 10,190,782 fullerenes on 120 or fewer vertices [23, 26]. The results are given in Table 3.1 and are plotted in Figure 3.2 along with the theoretical bounds. Although these results are known, a faster algorithm was desired.

Graver showed that for icosahedral (highly symmetric) fullerenes, the independence number can be calculated from a formula in $O(1)$ time [31]. The results here build on his ideas [31] that apply to general fullerenes to describe an algorithm for finding the independence number in $O(n)$ time.

This section concludes with some general definitions. This work makes heavy use

| $n$ | $x$ | | | | | | $n$ | $x$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | | 7 | 6 | 5 | 4 | 3 | 2 |
| 20 | - | - | - | - | - | 1 | 72 | - | 1 | 205 | 7796 | 3164 | 24 |
| 24 | - | - | - | - | 1 | - | 74 | - | - | 355 | 10254 | 3619 | 18 |
| 26 | - | - | - | - | - | 1 | 76 | - | 1 | 700 | 14314 | 4108 | 28 |
| 28 | - | - | - | - | 1 | 1 | 78 | - | 3 | 1130 | 18389 | 4561 | 26 |
| 30 | - | - | - | - | 2 | 1 | 80 | - | 4 | 2085 | 24478 | 5328 | 29 |
| 32 | - | - | - | - | 3 | 3 | 82 | - | 3 | 3245 | 30586 | 5853 | 31 |
| 34 | - | - | - | - | 5 | 1 | 84 | - | 17 | 5580 | 39418 | 6537 | 40 |
| 36 | - | - | - | - | 14 | 1 | 86 | - | 9 | 8458 | 47997 | 7268 | 29 |
| 38 | - | - | - | - | 15 | 2 | 88 | - | 28 | 13410 | 60111 | 8146 | 43 |
| 40 | - | - | - | 1 | 36 | 3 | 90 | - | 42 | 19216 | 71714 | 8899 | 47 |
| 42 | - | - | - | - | 42 | 3 | 92 | - | 102 | 28818 | 87532 | 9910 | 47 |
| 44 | - | - | - | 3 | 80 | 6 | 94 | - | 151 | 40015 | 102660 | 10617 | 50 |
| 46 | - | - | - | 10 | 102 | 4 | 96 | - | 330 | 57701 | 121976 | 11773 | 59 |
| 48 | - | - | - | 17 | 177 | 5 | 98 | - | 444 | 77104 | 140564 | 12862 | 43 |
| 50 | - | - | - | 25 | 241 | 5 | 100 | 1 | 1058 | 106173 | 164793 | 13819 | 70 |
| 52 | - | - | - | 83 | 347 | 7 | 102 | - | 1618 | 139263 | 185772 | 14933 | 72 |
| 54 | - | - | 1 | 122 | 448 | 9 | 104 | - | 3179 | 185783 | 213432 | 16551 | 68 |
| 56 | - | - | 1 | 257 | 656 | 10 | 106 | - | 4880 | 234863 | 240210 | 17504 | 72 |
| 58 | - | - | 2 | 379 | 816 | 8 | 108 | 3 | 8389 | 304350 | 272419 | 18961 | 95 |
| 60 | - | 1 | 6 | 736 | 1058 | 11 | 110 | 6 | 12857 | 378948 | 301041 | 20402 | 65 |
| 62 | - | - | 6 | 1081 | 1288 | 10 | 112 | 8 | 21735 | 477860 | 338539 | 21919 | 100 |
| 64 | - | - | 18 | 1799 | 1631 | 17 | 114 | 7 | 30083 | 583643 | 371130 | 23481 | 100 |
| 66 | - | - | 24 | 2558 | 1881 | 15 | 116 | 28 | 48970 | 720591 | 412142 | 25295 | 93 |
| 68 | - | - | 59 | 3955 | 2300 | 18 | 118 | 39 | 68788 | 863452 | 449762 | 26403 | 109 |
| 70 | - | 1 | 97 | 5395 | 2639 | 17 | 120 | 119 | 103796 | 1046902 | 494619 | 28606 | 129 |

Table 3.1: From [23], the number of fullerenes $C_n$ with independence number $n/2 - x$

of a rotation system for a planar embedding of a fullerene as well as its dual graph, each of which was defined in Chapter 2.

For a rotation system, a "*clockwise positions*" unit of measure is defined for two arcs with a common incident vertex as the number of positions in clockwise order from one arc to the other around the common vertex. Let $v_0, v_1, \ldots, v_{d-1}$ be the clockwise order of the neighbors of vertex $u$. Arcs $(u, v_i)$ and $(v_i, u)$ are $k$ clockwise positions after arcs $(u, v_j)$ and $(v_j, u)$ if and only if $i \equiv j + k \pmod{d}$.

A *primal walk* is a walk in the fullerene graph and a *dual walk* is a walk in the dual graph. When a dual walk enters a vertex $v_i$ of degree six from arc $(v_{i-1}, v_i)$, it is said to continue in the *straight* direction if it leaves $v_i$ using the arc from $v_i$ that is three clockwise positions after arc $(v_{i-1}, v_i)$. The straight direction is not defined for a walk exiting a vertex of degree five. A *straight walk* is a dual walk that exits in

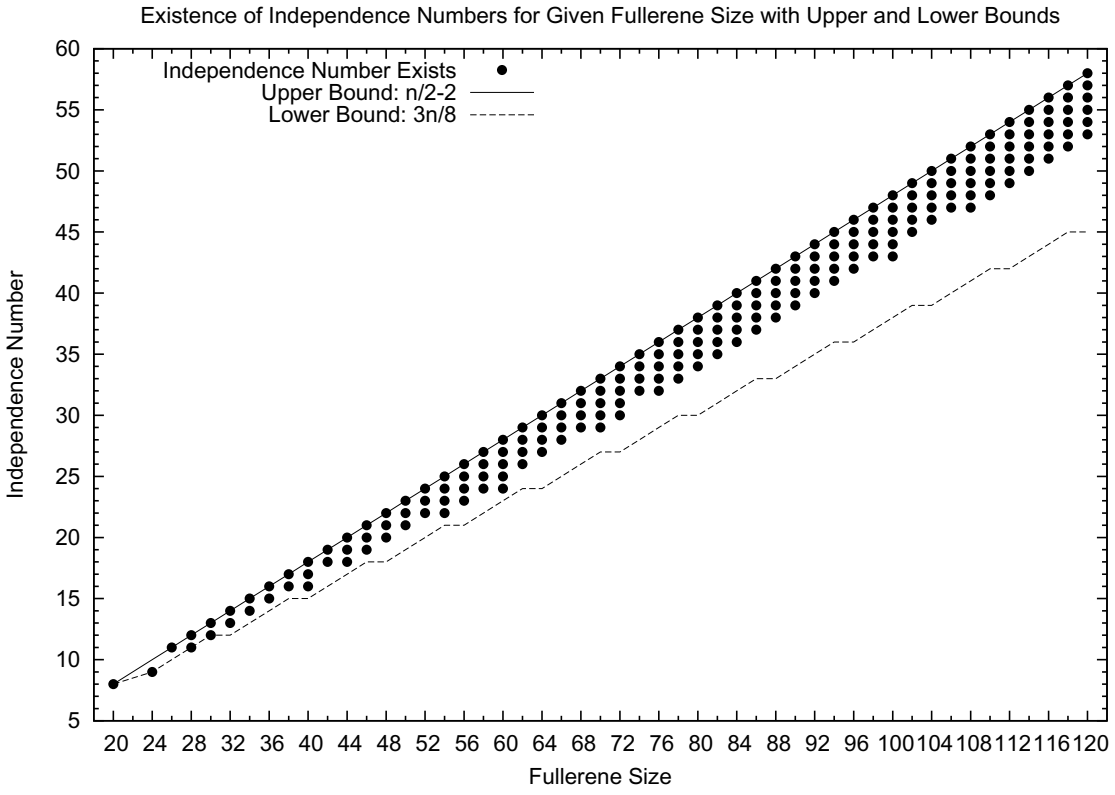



Figure 3.2: Existence of independence numbers for given fullerene size with the best-known theoretical bounds of $n/2 - 2$ and $\lceil 3n/8 \rceil$

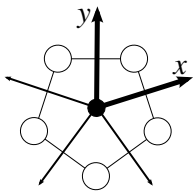



Figure 3.3: The $x$- and $y$-arcs of a quintant.

the straight direction at every internal vertex. This implies that every internal vertex of a straight walk has degree six. A walk makes a *sharp left (sharp right) turn* at vertex $v_i$ of degree five or six if it exits $v_i$ using the arc from $v_i$ that is one clockwise (counter-clockwise) position after arc $(v_i, v_{i-1})$. A *wide left (wide right) turn* occurs when the walk exits using the arc that is two clockwise (counter-clockwise) positions after arc $(v_i, v_{i-1})$.

A *quintant* of a vertex $v$ of degree five consists of a pair of arcs from $v$, the *$x$-arc* and the *$y$-arc*, such that the $x$-arc is 1 clockwise position after the $y$-arc. Figure 3.3 gives an example.

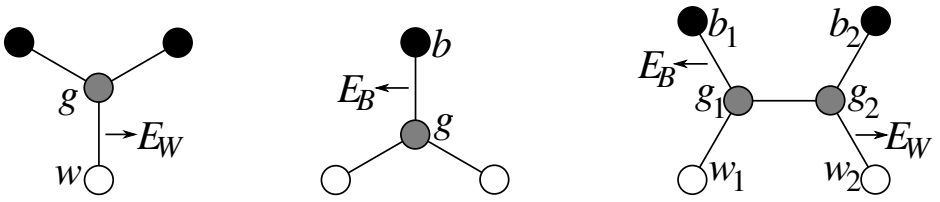



Figure 3.4: The three possible configurations involving gray vertices. [31, Fig. 1]

## 3.1 Graver's Results

This section highlights the results from Graver's work [31] that are useful here. In a maximum independent set, each hexagon ideally contributes three vertices; however, the pentagons can cause some disruptions. Each pentagon can contribute at most two vertices to an independent set. This can cause some hexagons to contribute fewer than three vertices to the set. The difficulty with calculating the independence number for fullerenes is with choosing independent set vertices from the pentagons to minimize their disruption of the hexagons. Graver showed that a maximum independent set can be obtained by finding a way to pair the pentagons where a dual walk exists from each pentagon to its paired pentagon (subject to some restrictions) such that the global disruption is minimized.

Let $W$ be a maximum independent set of a fullerene $F$, let $B$ be a maximum independent set of $F - W$ and let $G$ be the remaining vertices $(V - W - B)$. Color the vertices in $W$ white, those in $B$ black, and those in $G$ gray. The following property arises:

**Lemma 3.1.1.** *[31, Lemma 1] For every maximum independent set, every gray vertex is adjacent to a black vertex and a white vertex.*

**Definition 3.1.2** (Graver Coloring)**.** *A* Graver coloring *of a fullerene $F = (V, E)$ consists of a partitioning of the vertices into sets of white vertices $W$, black vertices $B$ and gray vertices $G$ and a partitioning of the edges into white edges $E_W$, black edges $E_B$ and gray edges $E_G$, such that $W$ is a maximum independent set of $F$, $B$ is a maximum independent set of $F - W$, $G$ is $V - W - B$, and $E_W$, $E_B$, and $E_G$ are defined as follows:*

**[Figure 3.4(a)]:** *If gray vertex $g$ is adjacent to two black vertices and one white vertex $w$, then edge $(g, w)$ is assigned to $E_W$.*

**[Figure 3.4(b)]:** *If gray vertex $g$ is adjacent to two white vertices and one black vertex $b$, then edge $(g, b)$ is assigned to $E_B$.*

**[Figure 3.4(c)]:** *For each pair of adjacent gray vertices, arbitrarily label one as $g_1$ and the other as $g_2$. Vertex $g_1$ is adjacent to one white vertex $w_1$ and one black vertex $b_1$ and $g_2$ is adjacent to one white vertex $w_2$ and one black vertex $b_2$. Assign $(g_1, b_1)$ to $E_B$ and $(g_2, w_2)$ to $E_W$.*

*The edge set $E_G$ is defined to be $E - E_W - E_B$.*

With the sets defined in this manner, each gray vertex is incident with exactly one edge of either $E_W$ or $E_B$. Note that this definition of $E_G$ differs from Graver's set $E_G$ in [31] in order to simplify some of the discussion.

**Lemma 3.1.3.** *[31, Lemma 2] For any Graver coloring, $|E_W| + |E_B| = |G|$ (where $G$ is the set of gray vertices) and no two edges in $E_W \cup E_B$ have a common endpoint.*

**Lemma 3.1.4.** *[31, Lemma 3] A Graver coloring will satisfy the following:*

*(i) Each pentagonal face contains exactly one edge from $E_W \cup E_B$.*

*(ii) Each hexagonal face contains either exactly two edges from $E_W \cup E_B$ or no edges from $E_W \cup E_B$. If two edges from $E_W \cup E_B$ are opposite one another on a hexagon, then they are either both from $E_W$ or both from $E_B$. If two edges from $E_W \cup E_B$ are on a hexagon and are not opposite one another, then one is from $E_W$ and one is from $E_B$.*

An important relationship exists between $|W|$, $|E_W|$ and $|E_B|$, which is shown by the next lemma.

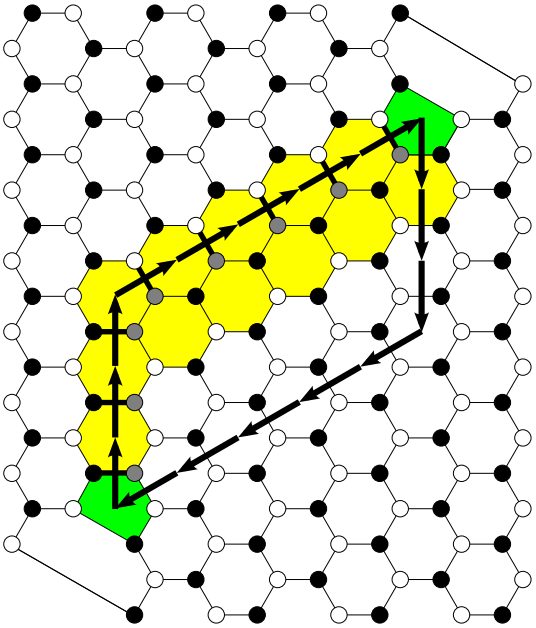
**Lemma 3.1.5.** *[31, Lemma 4] Given a Graver coloring, the independence number can be computed as*

$$|W| = |V|/2 - (2|E_W| + |E_B|)/3. \tag{3.1}$$

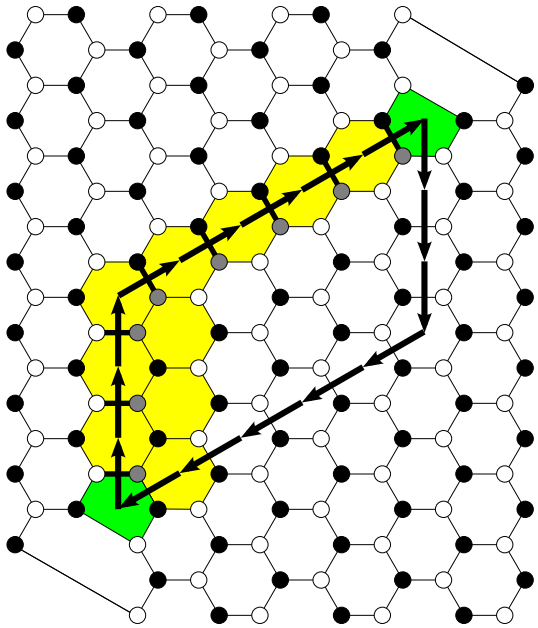
The next theorem allows one to view a correspondence between a maximum independent set and a pairing of the twelve pentagons.

**Theorem 3.1.6.** *[31, Theorem 1] Given a Graver coloring, the dual subgraph induced by the dual edges corresponding to the primal edges in $E_W \cup E_B$ is disconnected with six components, each of which is a simple path between a different pair of vertices of degree five.*

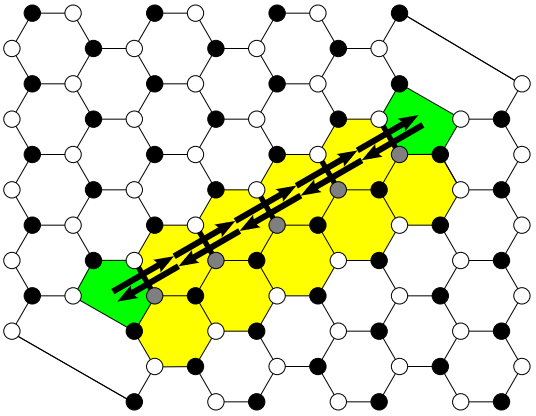
The paths in Theorem 3.1.6 cannot make a sharp turn at any hexagon due to Lemma 3.1.4(ii). The next lemma introduces a restriction on the wide turns that can be made.



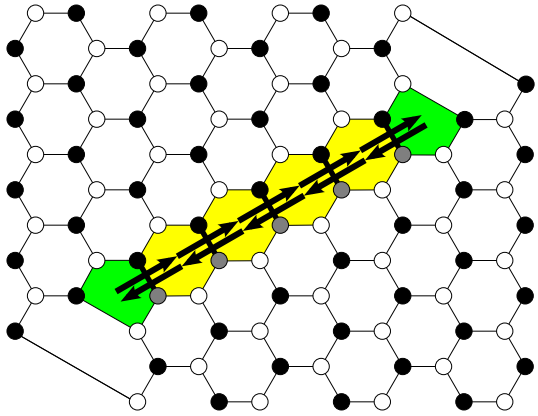
(a) A white $(5,3)$ clear field: penalty is $2(5) + 3 = 13$



(b) A black $(5,3)$ clear field: penalty is $5+2(3) = 11$



(c) A white $(5,0)$ clear field: penalty is $2(5)+0 = 10$



(d) A black $(5,0)$ clear field: penalty is $5+2(0) = 5$

Figure 3.5: Examples of clear fields. The arcs of the clear fields are denoted by arrows. A Graver coloring is shown in each case. Bold primal edges denote the edges in $E_W \cup E_B$. If fewer than three vertices in $W$ are on a hexagon then it is shaded.

**Lemma 3.1.7.** *[31, Lemma 6] If any particular one of the six paths in Theorem 3.1.6 is viewed as a dual walk from one degree five vertex to its mate, then if the walk has a wide right (left) turn then the next turn cannot be another wide right (left) turn.*

Figure 3.5 shows sample regions that may occur in a Graver coloring of a fullerene. If a hexagon has fewer than three vertices from $W$ then it is shaded. The (primal) edges in $E_W \cup E_B$ are shown in bold. In each of the four subfigures, the edges of the

dual graph that correspond to the bold primal edges induce a path that pairs the pentagons.

Figure 3.5(a) is an example of a claim by Graver that will be shown by Theorem 3.2.3. The path that connects the two pentagons can be thought of as a dual walk from the lower left pentagon to the upper right pentagon such that the walk follows three arcs in the "up" direction and then makes one wide right turn and follows five arcs in the "right" direction. Graver describes a method of swapping the colors of vertices to obtain a new Graver coloring that induces any one of the choices of a path between the two pentagons with the path having three edges in the "up" direction and five in the "right" direction. Each of these path choices falls into the parallelogram-shaped region illustrated by the circuit of arrows. Graver called such a parallelogram-shaped region of hexagonal faces between the two pentagonal faces a *clear field*, although a more rigorous definition is provided in the next section. The term clear field was chosen because for all maximum independent sets of the fullerene and a corresponding Graver coloring, these regions must be free of pentagons other than the two being paired [31, Lemma 7].

Two different colorings of a fullerene are said to be *color-equivalent* if they have the same values for $|W|$, $|B|$, $|E_W|$ and $|E_B|$. The colorings that induce different paths in the clear field are color-equivalent and in particular "any two [such] paths in the clear field between paired pentagons will have the same contribution to $2|E_W| + |E_B|$; hence that contribution is a property of the pairing" [31, p. 861]. A proof of this fact is given later by Theorem 3.2.3. The contribution that a clear field makes to $2|E_W| + |E_B|$ is called the *penalty* of a clear field because in Equation (3.1), the sum of the penalties divided by three gives the difference between $|V|/2$ and the independence number. In the example in Figure 3.5(a), the clear field contains five edges in $E_W$ and three edges in $E_B$ for a penalty of $2(5) + 3 = 13$. Any Graver coloring that results in this clear field with the boundary colored as indicated will have a path in the dual corresponding to the edges of $E_W \cup E_B$ that gives the same penalty. Similarly, in Figure 3.5(b), the clear field contains three edges in $E_W$ and five edges in $E_B$ for a penalty of $5 + 2(3) = 11$.

In the final section of [31], Graver comments that to find a maximum independent set on any fullerene, one could consider the independent sets that correspond to each of the possible pairings of the pentagons resulting in clear fields and select the independent set with the largest order. That is the general approach taken by the algorithm.

## 3.2 Extensions of Graver's Results

This section proves some new results that expand on Graver's work in order to build a theoretical foundation for the algorithm. The first result uses the Graver coloring properties to examine how the colors of two vertices are related by examining primal walks that connect them.

**Theorem 3.2.1.** *Given a fullerene with a Graver coloring, let $P$ be a primal walk of length $k$ from vertex $v_0$ to $v_k$ such that $v_0$ and $v_k$ are not gray. Let $\ell$ be the number of arcs of $P$ that correspond to some edge in $E_W \cup E_B$. If $k \equiv \ell \pmod 2$ then $v_0$ and $v_k$ are the same color, otherwise they are different colors.*

*Proof.* The possible internal vertex colorings are considered by cases.

**Case 1** [$k < 2$]**:** If $k = 0$ then the result holds trivially. If $k = 1$ then the single arc must connect a white and a black vertex and the edge it corresponds to is hence not in $E_W \cup E_B$, so the result holds.

**Case 2** [$k \geq 2$ and all internal vertices of $P$ are gray]**:** There are two subcases.

**Case 2.1** [$k = 2$]**:** The Graver coloring properties illustrated in Figure 3.4 show that $v_0$ and $v_2$ are the same color if and only if neither arc on the path corresponds to an edge in $E_W \cup E_B$ or if both arcs correspond to the same edge in $E_W \cup E_B$. Otherwise, $v_0$ and $v_2$ are different colors.

**Case 2.2** [$k \geq 3$]**:** No gray vertex has more than one gray neighbor, so the situation must be the one shown in Figure 3.4(c). The $k-2$ arcs of $P$ that are not the first or last arc must correspond with the edge that connects the two gray vertices (recall that walks can have repeated arcs). Only the first and last arcs of $P$ can correspond with an edge in $E_W \cup E_B$. If $k$ is odd, then $v_0$ and $v_k$ are the same color if and only if exactly one of the arcs corresponds with an edge in $E_W \cup E_B$. If $k$ is even, then $v_0$ and $v_k$ are the same color if and only if zero or two of the arcs correspond with an edge in $E_W \cup E_B$. Otherwise, $v_0$ and $v_k$ are different colors.

**Case 3** [$k \geq 2$ and not all internal vertices of $P$ are gray]**:** Let $v_i$ be a vertex in $P$ that is not gray. Walk $P$ can be decomposed into two smaller walks $P_1$ with length $k_1$ from $v_0$ to $v_i$ and $P_2$ with length $k_2$ from $v_i$ to $v_k$. Induction shows that if the result holds for $P_1$ and $P_2$, then the result holds for $P$. □

The idea of a clear field was given in the previous section, but here a more rigorous definition is provided.

**Definition 3.2.2** (Clear Field)**.** *A* clear field *with dimensions* $(x, y)$ *is a circuit in the dual graph such that each of the following hold:*

1. *The dimensions* $(x, y)$ *satisfy* $x \geq 1$ *and* $y \geq 0$*.*

2. *The circuit and its internal region (the one on the right side of the circuit) contain exactly two vertices corresponding to pentagons.*

3. *The circuit corresponds to a closed dual walk using* $2x+2y$ *arcs and is constructed in one of the following two ways, depending on the value of* $y$*.*

   **Case 1** [$y = 0$] *Two examples with* $y = 0$ *are shown in Figures 3.5(c) and 3.5(d). The dual walk begins at a degree five vertex and follows an arc. It then continues for* $x - 1$ *more arcs in the straight direction (through vertices of degree six) to reach a degree five vertex. This vertex must be a different vertex than the starting vertex. The walk then reverses its steps back to the initial degree five vertex.*

   **Case 2** [$y > 0$] *Two examples with* $y > 0$ *are shown in Figures 3.5(a) and 3.5(b). The dual walk begins at a degree five vertex and follows an arc to an adjacent degree six vertex. It then continues for* $y - 1$ *more arcs in the straight direction (through vertices of degree six) for a total of* $y$ *arcs to reach a degree six vertex, where a wide right turn is made. The walk continues for* $x - 1$ *more arcs in the straight direction and reaches a degree five vertex after a total of* $x + y$ *arcs. This vertex must be a different vertex than the starting vertex. A sharp right turn is made at the vertex of degree five. The walk continues for* $y - 1$ *more arcs in the straight direction where a wide right turn is made. Finally, the walk continues* $x - 1$ *more arcs in the straight direction and reaches the original degree five vertex to close the walk (if a sharp right turn were made, it would use the first arc of the walk).*

The *clear field subgraph* for a given clear field is the primal subgraph induced by the faces that correspond with either a dual vertex that is part of the clear field or a dual vertex on the inside (right side) of the clear field.

Some clear fields will be named as necessary in the discussions. Given a specific clear field, arbitrarily label its two vertices of degree five $a$ and $b$ and denote the clear

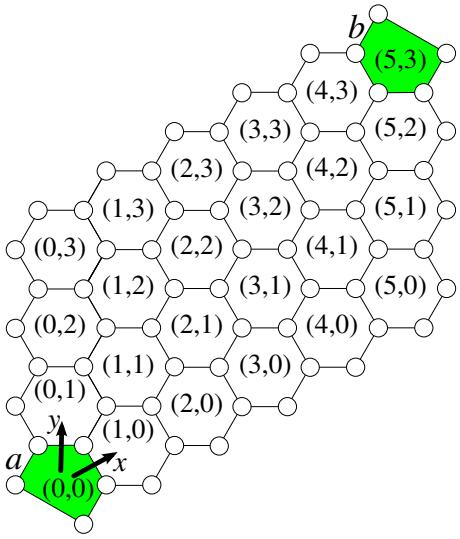



Figure 3.6: The faces in a clear field region may be labeled with coordinates using one of the pentagons ($a$) as the origin $(0, 0)$.

field $\square ab$ having dimensions $(x_{ab}, y_{ab})$. Multiple clear fields may exist between two vertices of degree five and so it is important to note that given two such vertices $a$ and $b$, the label $\square ab$ is not always sufficient to identify a unique clear field. When such notation is used, however, ambiguity will be avoided by identifying the clear field and then naming it. When a clear field is named $\square ab$, let $\overrightarrow{ab}$ denote the subwalk of $\square ab$ from $a$ to $b$ and let $\overrightarrow{ba}$ denote the subwalk of $\square ab$ from $b$ to $a$.

Given a Graver coloring, the six connected components of the dual subgraph induced by the edges of $E_W \cup E_B$ are called the *pairing paths*. A particular clear field $\square ab$ is said to correspond to a pairing path if the path pairs vertices $a$ and $b$, all of the primal edges corresponding to the edges of the path are in the clear field subgraph of $\square ab$, and the length of the path is $x_{ab} + y_{ab}$. A clear field is a *pairing clear field* if it corresponds to a pairing path.

The faces in a clear field subgraph and their corresponding dual vertices may be referred to using a coordinate system. Choose a clear field and label it $\square ab$ with degree five vertices $a$ and $b$ labeled arbitrarily. The two arcs of $\square ab$ that are incident with $a$ correspond with one of the five quintants of $a$ (Figure 3.6). This quintant is said to be the *quintant of a for* $\square ab$. A coordinate system is defined for $\square ab$ such that a straight walk beginning with the $x$-arc of the quintant defines the *$x$-axis* of the quintant and the *$y$-axis* is similarly defined using the $y$-arc. A dual vertex corresponding to a face in the clear field subgraph of $\square ab$ receives the coordinate $(x, y)$ if it may be reached by following the $y$-axis for $y$ arcs and, if $x > 0$, then making

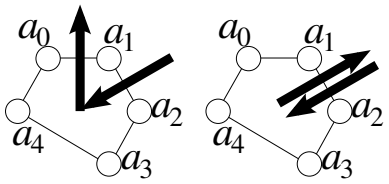



Figure 3.7: A numbering system for the vertices of a pentagon with the clear field arcs shown.

a wide right turn (and following the appropriate arc) then taking $x-1$ more arcs in the straight direction, as in Figure 3.6. Vertex $a$ is the origin at $(0,0)$ and $b$ is located at $(x_{ab}, y_{ab})$.

It is important to note that the faces of the clear field subgraph do not necessarily have unique coordinates, but each coordinate uniquely identifies a face. Figure 3.10 is an example of such a situation. If the face at coordinate $(x_1, y_1)$ is the same face as the one at coordinate $(x_2, y_2)$, then the notation $(x_1, y_1) \equiv (x_2, y_2)$ is used to express this fact.

Given a particular quintant, an *x-direction walk at y* is a straight dual walk with its first arc from the dual vertex at $(0, y)$ to the one at $(1, y)$. Similarly, a *y-direction walk at x* is a straight dual walk with its first arc from the dual vertex at $(x, 0)$ to the one at $(x, 1)$.

For a Graver coloring, each vertex has a color and a pentagon is now defined to have a color of white or black as follows. Let $\angle ab$ be the clear field that pairs degree five vertices $a$ and $b$ given some Graver coloring. As in Figure 3.7, number the primal vertices on the pentagon corresponding to $a$ in clockwise order as $a_0, \ldots, a_4$ such that the dual arc of $\angle ab$ that enters $a$ crosses primal edge $(a_1, a_2)$. Define the *pentagon color* of the pentagon for $a$ to be the color of $a_2$, unless $a_2$ is gray, in which case the pentagon receives the color of $a_1$. In Figures 3.5(a) and 3.5(c), the pentagons are white and in Figures 3.5(b) and 3.5(d), the pentagons are black.

Exactly one of $(a_0, a_1)$ and $(a_1, a_2)$ is in $E_W \cup E_B$ because $\angle ab$ pairs $a$ and $b$. Note that if $a_2$ is gray, then $(a_1, a_2) \in E_W \cup E_B$ so $a_1$ has a different color than the other two neighbors of $a_2$ (see Figure 3.4) and a color-equivalent coloring may be obtained by swapping the colors of $a_1$ and $a_2$. Similarly, if $(a_0, a_1) \in E_W \cup E_B$ then a color-equivalent coloring may be obtained by swapping the colors of $a_0$ and $a_1$. Equivalently, the color of the pentagon for $a$ is white if $(a_1, a_2) \in E_W$ or $(a_0, a_1) \in E_B$ and is black if $(a_1, a_2) \in E_B$ or $(a_0, a_1) \in E_W$.

**Theorem 3.2.3.** *Let $a$ and $b$ be two degree five vertices that are paired by a Graver coloring and let $\angle ab$ be their pairing clear field with dimensions $(x, y)$. Let $F' =$*

*$(V', E')$ be the clear field subgraph of $\angle ab$.*

*(i) There are exactly $2^{x+y}\binom{x+y}{x}$ ways to color the vertices of $F'$ such that a pairing path exists between $a$ and $b$ and the pentagon for $a$ is the same color as in $F$.*

*(ii) If the pentagon for $a$ is white then $|E_W \cap E'| = x$ and $|E_B \cap E'| = y$, otherwise $|E_W \cap E'| = y$ and $|E_B \cap E'| = x$.*

*(iii) The colors of the pentagons for $a$ and $b$ are the same.*

*Proof.* Label the vertices of the pentagon corresponding with $a$ as $a_0$ through $a_4$ as in Figure 3.7. Similarly, label $b_0$ through $b_4$.

(i) The paths in the dual of $F'$ that pair $a$ and $b$ correspond to walks of length $x + y$ from $a$ to $b$ within the clear field region of $\angle ab$ such that no two consecutive wide turns are made in the same direction. Counting these walks is equivalent to the classical problem of counting the number of lattice walks from the origin $(0, 0)$ to $(x, y)$ using steps in the north and east directions only. The north direction corresponds with the $y$-direction from $a$ (taking the dual arc from $a$ that crosses $(a_0, a_1)$ and continuing straight). Likewise, the east direction corresponds with the $x$-direction and $(a_1, a_2)$. The number of such lattice walks is $\binom{x+y}{x}$ [41].

Once a pairing path $P$ has been selected, it is easy to generate a coloring of $F'$ that induces the pairing path. Let $E_P$ be the edges in $P$. For each primal edge corresponding to an edge in $E_P$, select one incident vertex and color it gray. Color $a_1$ and/or $a_2$ (whichever is not gray) black or white as appropriate such that the pentagon for $a$ is the same color in $F'$ as in $F$. The graph $F' - E_P$ is bipartite so there is a unique way that the remaining uncolored vertices of $F'$ can be colored with black and white.

Each of these possible paths pairing $a$ with $b$ contains $x + y$ edges in $E_W \cup E_B$. No two edges in $E_W \cup E_B$ are adjacent so the colors of the vertices on the endpoints of an edge in $E_W \cup E_B$ may be swapped, for a total of $2^{x+y}$ ways to color each pairing path. Each coloring induces exactly one pairing path, so $2^{x+y}\binom{x+y}{x}$ counts each coloring exactly once.

(ii) Let $e_1, e_2, \ldots, e_{x+y}$ be the arcs of the walk from $a$ to $b$ for some pairing path and let $v_0, v_1, \ldots, v_{x+y}$ be the vertices of the walk. Let $e'_1, e'_2, \ldots, e'_{x+y}$ be the edges in $E_W \cup E_B$ such that $e'_i$ is the primal edge crossed by $e_i$. If the walk follows the straight direction at $v_i$ $(0 < i < x + y)$, then $e'_i$ and $e'_{i+1}$ are either both in $E_W$ or both in $E_B$ (Lemma 3.1.4). Conversely, if the walk makes a turn at $v_i$, then one of $e'_i$ and $e'_{i+1}$ is in $E_W$ and the other is in $E_B$ (Lemma 3.1.4). When the walk is viewed as a

walk on the lattice from the origin $(0,0)$ to $(x,y)$, each step in the north direction indicates an edge in one of $E_W$ or $E_B$ and each step in the east direction indicates an edge in the other set. If the pentagon for $a$ is white then either $(a_1,a_2) \in E_W$ or $(a_0,a_1) \in E_B$, so $|E_W \cap E'| = x$ and $|E_B \cap E'| = y$. If the pentagon for $a$ is black then either $(a_1,a_2) \in E_B$ or $(a_0,a_1) \in E_W$, so $|E_B \cap E'| = x$ and $|E_W \cap E'| = y$.

(iii) Continuing the argument from (ii), if the pentagon for $a$ is white and the final step of the walk from $a$ to $b$ is in the east direction, it crosses $(b_1,b_2)$ so $(b_1,b_2) \in E_W$. If the final step is in the north direction, it crosses $(b_0,b_1)$ so $(b_0,b_1) \in E_B$. In either case, the pentagon for $b$ is white. If the pentagon for $a$ is black then either $(b_1,b_2) \in E_B$ or $(b_0,b_1) \in E_W$, so the pentagon for $b$ is black. □

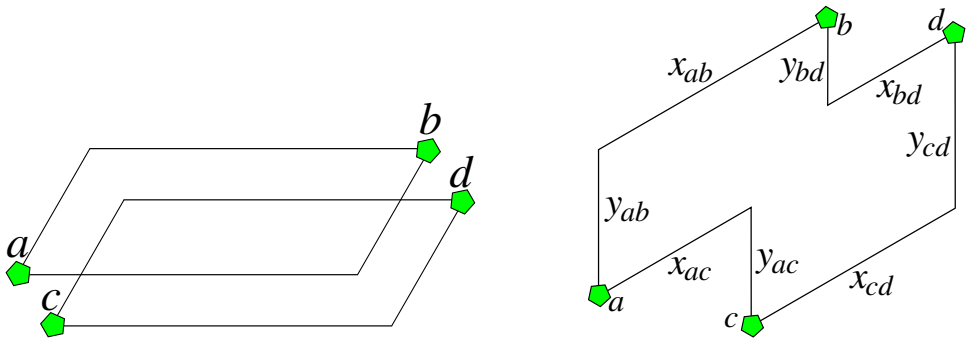
Graver's claim that each clear field will have the same contribution to $2|E_W|+|E_B|$ is backed up by Theorem 3.2.3. Furthermore, it gives the ability to replace one pairing path corresponding to a clear field with any other pairing path corresponding to the same clear field without affecting the independent set order.

The *clear field color* is defined as the color the clear field's corresponding pentagons. Figures 3.5(a) and 3.5(c) show white clear fields and Figures 3.5(b) and 3.5(d) show black clear fields. If a clear field is white, then the primal edges crossed by the edges of the pairing path in the $x$-direction are in $E_W$ and if the clear field is black, these edges are in $E_B$. Because the contribution of a clear field to $2|E_W|+|E_B|$ is the penalty of the clear field, it follows from Theorem 3.2.3(ii) that if a clear field with dimensions $(x,y)$ is white, its penalty is $2x+y$ and if it is black, the penalty is $x+2y$. Equation (3.1) may be rewritten as

$$|W| = \frac{|V(G)|}{2} - \frac{1}{3} \sum_{\text{clear fields}} (\text{clear field penalties}). \tag{3.2}$$

**Theorem 3.2.4.** *Given a Graver coloring of a fullerene, let $a$ and $b$ be two dual vertices of degree five paired by the coloring and let $\square ab$ be their pairing clear field. Similarly, $c$ and $d$ are paired by pairing clear field $\square cd$. The clear field subgraphs of $\square ab$ and $\square cd$ have no faces in common.*

*Proof.* For contradiction, assume the clear field subgraphs of $\square ab$ and $\square cd$ has a face in common, so at least one of $\overrightarrow{ab}$ and $\overrightarrow{ba}$ have a dual vertex in common with at least one of $\overrightarrow{cd}$ and $\overrightarrow{dc}$. By Theorem 3.1.6, the pairing paths of $\square ab$ and $\square cd$ have no dual vertices in common. For such pairing paths to exist, at least one of $\overrightarrow{ab}$ and $\overrightarrow{ba}$ must not share a dual vertex with either $\overrightarrow{cd}$ nor $\overrightarrow{dc}$ and at least one of $\overrightarrow{cd}$ and $\overrightarrow{dc}$ must not share a dual vertex with either $\overrightarrow{ab}$ nor $\overrightarrow{ba}$. Therefore, by interchanging labels $a$




(a) Example of $\overrightarrow{ba}$ sharing dual vertices with $\overrightarrow{cd}$

(b) Distance labels when two clear fields overlap

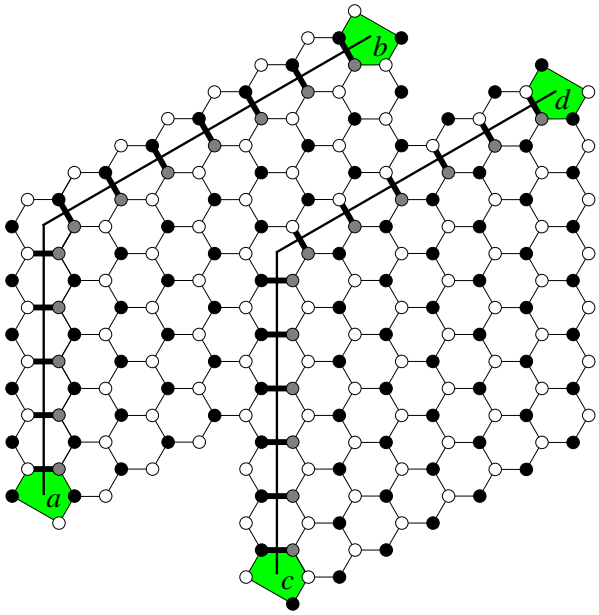
Figure 3.8: Situation when two clear fields overlap in the proof of Theorem 3.2.4

and $b$ and/or $c$ and $d$ if necessary, it can be assumed without loss of generality that $\overrightarrow{ba}$ shares at least one dual vertex with $\overrightarrow{cd}$, as illustrated in Figure 3.8(a). It will be shown that a better coloring exists by pairing $a$, $b$, $c$, and $d$ differently, which implies the coloring is not optimal and provides a contradiction.

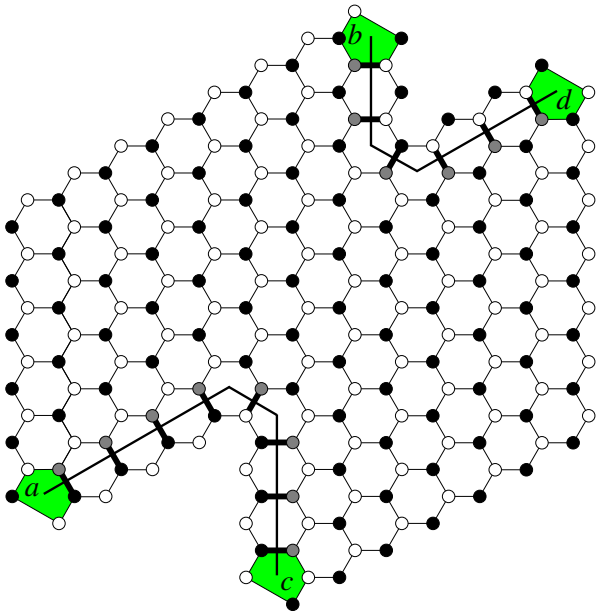
Some distances in the graph are labeled in Figure 3.8(b). The length of the pairing path between $a$ and $b$ is $x_{ab} + y_{ab}$. The distance from $c$ to $a$ using the arcs of the clear fields involves the first $y_{ac}$ arcs of $\overrightarrow{cd}$ followed by the last $x_{ac}$ arcs of $\overrightarrow{ba}$. The other distances are similarly defined. Figure 3.9(a) shows an example with $\square ab$ with dimensions $(7, 5)$, $\square cd$ with dimensions $(6, 6)$, $x_{ac} = 5$, $y_{ac} = 4$, $x_{bd} = 4$, and $y_{bd} = 3$.

It can be assumed that no other pairing path uses any of the dual vertices on the dual walk from $c$ to $a$, nor on the walk from $b$ to $d$. This is because if some other pairing path did cross between $c$ and $a$, it must also cross between $b$ and $d$ since it cannot share a dual vertex with the path pairing $a$ and $b$ nor with the path pairing $c$ and $d$. Such a path would indicate that a third clear field shares dual vertices with $\square ab$ and $\square cd$. Without loss of generality, the third clear field can be renamed to be $\square cd$, repeating as necessary, so as to assume that no pairing path shares a dual vertex with $c$ to $a$ or $b$ to $d$.

Label the vertices of the pentagon at $a$ with $a_0$ through $a_4$ as per Figure 3.7 and similarly label the pentagons at $b$, $c$, and $d$. Consider the primal walk from $c_2$ to $a_2$ around the boundary of the subgraph induced by the clear field subgraphs of $\square ab$ and $\square cd$. This walk includes an odd number of arcs (three if $x_{ac} = y_{ac} = 1$ and $1 + 2(y_{ac} - 1) + 2(x_{ac} - 1)$ otherwise), none of which are in $E_W \cup E_B$ because no pairing path crosses them. By Theorem 3.2.1, $\square ab$ and $\square cd$ have opposite colors. Figure 3.9(a) shows an example in which $\square ab$ is black and $\square cd$ is white. This may be assumed without loss of generality because the other option is obtained by




(a) $|E_W| = 11$ and $|E_B| = 13$




(b) $|E_W| = 6$ and $|E_B| = 8$

Figure 3.9: Colorings of the subgraphs of two clear fields that share faces

interchanging labels $a$ and $d$ as well as $b$ and $c$.

Define a dual walk $W_{ca}$ from $c$ to $a$ as follows. If $y_{ac} > 1$, $W_{ca}$ begins at $c$ and uses the first $y_{ac} - 1$ arcs of $\overrightarrow{cd}$, then it makes a wide left turn to reach a dual vertex on $\square ab$. If $y_{ac} = 1$, $W_{ca}$ begins at $c$ and uses the arc that crosses primal arc $(c_0, c_4)$. In either case, if $x_{ac} > 1$, $W_{ca}$ continues by following the last $x_{ac} - 1$ arcs of $\overrightarrow{ba}$ to reach $a$. The total length of $W_{ca}$ is $x_{ac} + y_{ac} - 1$. Similarly, define $W_{bd}$ from $b$ to $d$ having

length $x_{bd} + y_{bd} - 1$. Figure 3.9(b) shows these walks on an example.

For the original coloring pairing of $a$ with $b$ and $c$ with $d$, the clear field subgraphs of $\square ab$ and $\square cd$ contribute $y_{ab} + x_{cd}$ edges to $E_W$ and $x_{ab} + y_{cd}$ edges to $E_B$. Obtain a new coloring by instead pairing $a$ with $c$ and $b$ with $d$. Assign the primal edges crossed by $W_{ca}$ and $W_{bd}$ to $E_W$ or $E_B$ and color the clear field subgraphs of $\square ab$ and $\square cd$ such that $a$ and $b$ remain white and $c$ and $d$ remain black. In this new coloring, the clear field subgraphs of $\square ab$ and $\square cd$ contribute $x_{bd} + y_{bd} - 1$ edges to $E_W$ and $x_{ac} + y_{ac} - 1$ edges to $E_B$. The new coloring is better (it provides a larger $|W|$) because $x_{ac} \leq x_{ab}$, $y_{ac} \leq y_{cd}$, $x_{bd} \leq x_{cd}$, and $y_{bd} \leq y_{ab}$. Note that if the new pairings make two consecutive left turns (they do when $x_{ac} + y_{ac} > 2$ or $x_{bd} + y_{bd} > 2$), then the new coloring is not optimal (Lemma 3.1.7); however it is better than the original coloring, which is sufficient for a contradiction. □

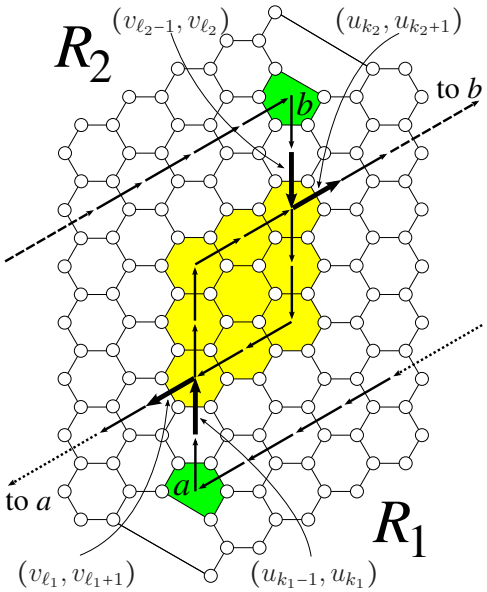
The previous lemma shows that the clear field subgraphs of two clear fields that pair pentagons cannot share a face with each other. The next lemma extends that idea to a single clear field by showing that the clear field subgraph of a pairing clear field cannot share a face with itself.

**Theorem 3.2.5.** *Given a Graver coloring of a fullerene, let $a$ and $b$ be two dual vertices of degree five paired by the coloring and let $\square ab$ be their pairing clear field with dimensions $(x, y)$. If $y = 0$, then the $x + 1$ dual vertices of $\square ab$ from $a$ to $b$ are unique. If $y > 0$, then all the dual vertices of $\square ab$ are unique. That is, the clear field subgraph of $\square ab$ contains $(x + 1)(y + 1)$ unique faces.*

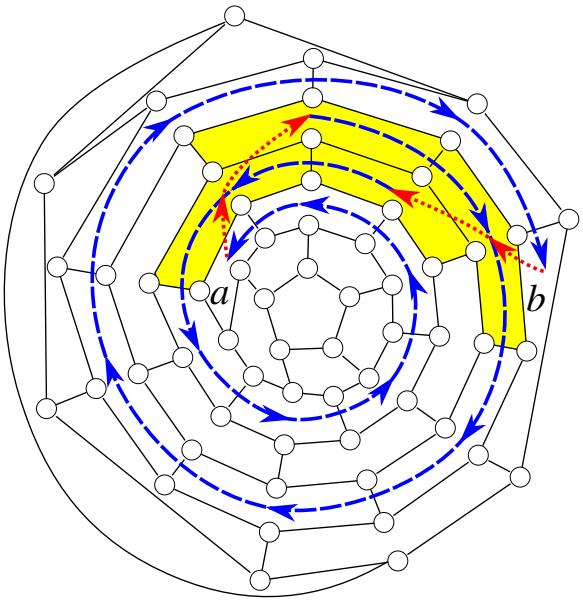
*Proof.* For contradiction, suppose $\square ab$ contains a repeated dual vertex. Walk $\overrightarrow{ab}$ does not contain a repeated dual vertex because if it did, the clear field subgraph of $\square ab$ could be recolored using the primal edges crossed by $\overrightarrow{ab}$ to induce a pairing path that is not a simple path and would violate Theorem 3.1.6. Likewise, $\overrightarrow{ba}$ does not contain a repeated dual vertex. It must be that a repeated dual vertex in $\square ab$ is found once in $\overrightarrow{ab}$ and once in $\overrightarrow{ba}$.

The above gives the result for the case where $y = 0$. Henceforth, assume $y > 0$.

Let $u_0, u_1, \ldots, u_{x+y}$ be the dual vertices visited by $\overrightarrow{ab}$ and let $v_0, v_1, \ldots, v_{x+y}$ be the dual vertices visited by $\overrightarrow{ba}$. Note that $a \equiv u_0 \equiv v_{x+y}$ and $b \equiv v_0 \equiv u_{x+y}$. Walks $\overrightarrow{ab}$ and $\overrightarrow{ba}$ make their wide right turns at $u_y$ and $v_y$, respectively. Let $w_0, w_1, \ldots, w_k$ be the dual vertices of an arbitrary straight dual walk. Because $\overrightarrow{ab}$ makes a wide right turn at $u_y$, it is not possible for the straight walk to "cut the corner" of $\overrightarrow{ab}$. That is, $w_i$ and $w_j$ $(i < j)$ cannot exist such that $w_i$ is a vertex in $\{u_1, \ldots, u_{y-1}\}$, $w_j$ is a vertex in $\{u_{y+1}, \ldots, u_{x+y-1}\}$, and $\{w_i, \ldots, w_j\}$ all correspond to faces in the clear field subgraph




(a) Four dual arcs are labeled. Labels $R_1$ and $R_2$ are for identifying this diagram with Figure 3.11(a). Shaded hexagons do not have unique coordinates.



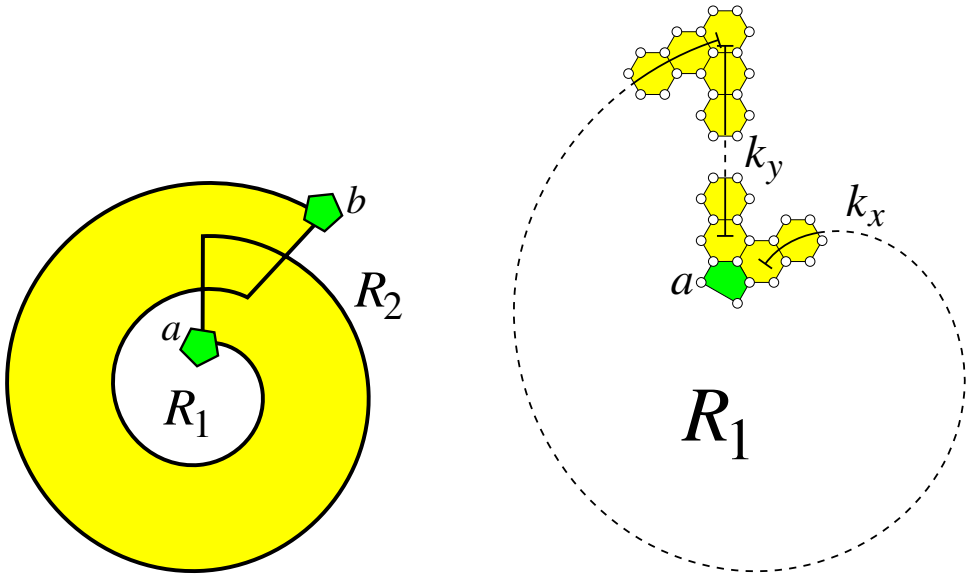

(b) An example of the situation in (a) on a fullerene with $n = 60$ showing a clear field with a $y$-value (dotted) of 2 and an $x$-value (dashed) of 7. Shaded hexagons do not have unique coordinates.

Figure 3.10: A situation used in the proof of Theorem 3.2.5

of $\square ab$. Therefore, if $\overrightarrow{ab}$ and $\overrightarrow{ba}$ share a vertex, it must be the situation pictured in Figure 3.10(a). That is, $\overrightarrow{ba}$ must "enter" $\square ab$ by meeting $\overrightarrow{ab}$ at a vertex $u_y$ through $u_{x+y-1}$ and then make its turn and "exit" $\square ab$ by meeting $\overrightarrow{ab}$ at a vertex $u_1$ through $u_y$. More specifically, there is a choice of $1 \le k_1 \le y$ and $y \le \ell_1 \le x + y - 1$ such that $u_{k_1} \equiv v_{\ell_1}$ and arc $(v_{\ell_1}, v_{\ell_1+1})$ is 1 clockwise position after arc $(u_{k_1-1}, u_{k_1})$ and there is a choice of $y \le k_2 \le x + y - 1$ and $1 \le \ell_2 \le y$ such that $u_{k_2} \equiv v_{\ell_2}$ and arc $(u_{k_2}, u_{k_2+1})$ is 1 clockwise position after $(v_{\ell_2-1}, v_{\ell_2})$. Figure 3.10(b) shows an example of a complete fullerene exhibiting the situation in Figure 3.10(a). Such a situation is common in larger fullerenes.

Figure 3.11(a) shows an abstract drawing of a clear field that exhibits the situation in Figure 3.10(a). Because the three-dimensional surface of the fullerene is being drawn in two dimensions, one part of each of $\overrightarrow{ab}$ and $\overrightarrow{ba}$ is drawn with a curved line, even though they represent the straight direction. The faces not in the clear field subgraph of $\square ab$ induce two fullerene subgraphs $R_1$ and $R_2$, which are also identified in Figure 3.10(a). Figure 3.11(b) shows a more detailed drawing of the faces in the clear field subgraph of $\square ab$ that are adjacent to subgraph $R_1$.

The clear field subgraph of $\square ab$ contains two of the twelve pentagons, leaving ten pentagons to be distributed between subgraphs $R_1$ and $R_2$. Five of these pentagons




(a) An abstract diagram with the clear field region shaded

(b) Boundary of the interior white region in (a)

Figure 3.11: More diagrams of the situation pictured in Figure 3.10

are in $R_1$ due to the following. Refer to Figure 3.11(b) and note that the $x$-axis and $y$-axis of the quintant of $a$ for $\angle ab$ meet at the hexagon that has both coordinates $(0, k_y)$ and $(k_x, 0)$ when the origin is defined at $a$. Subgraph $R_1$ has a large outer face bordered by the faces shown in the figure. This outer face of $R_1$ has two vertices of degree three from $a$, one from each of the first $k_x - 2$ hexagons on the $x$-axis and one from each of the first $k_y - 2$ hexagons on the $y$-axis for a total of $k_x + k_y - 2$ vertices of degree three. The remaining $k_x + k_y - 1$ vertices on the outer face of $R_1$ have degree two. There are $2k_x + 2k_y - 3$ edges on the outer face of $R_1$. Let $p$ be the number of pentagonal faces in $R_1$ and $h$ be the number of hexagonal faces in $R_1$. The total number of faces, edges, and vertices, respectively, are:

$$\begin{aligned} f &= p + h + 1 \\ e &= (5p + 6h + (2k_x + 2k_y - 3))/2 \\ v &= (5p + 6h - \underbrace{(k_x + k_y - 1)}_{\text{outer vert. of deg. 2}} + \underbrace{(k_x + k_y - 2)}_{\text{outer vert. of deg. 3}})/3 + \underbrace{(k_x + k_y - 1)}_{\text{total vert. of deg. 2}} \\ &= (5p + 6h - 1)/3 + k_x + k_y - 1. \end{aligned}$$

Euler's formula ($v - e + f = 2$) [20] gives:

$$((5p + 6h - 1)/3 + k_x + k_y - 1) - ((5p + 6h - 3)/2 + k_x + k_y) + (p + h + 1) = 2$$

that simplifies to $p = 5$.

Five pentagons are in $R_1$, which implies that five are in $R_2$. The dual vertices of degree five corresponding to these ten pentagons are paired up by the Graver coloring using clear fields, which implies that at least one of these dual vertices, $c$, corresponding to a pentagon in $R_1$ must be paired with another dual vertex, $d$, corresponding to a pentagon in $R_2$. This implies that pairing clear field $\square\!\!/cd$ must share a dual vertex with $\square\!\!/ab$, but this cannot happen by Theorem 3.2.4. □

**Definition 3.2.6** (Proper Clear Field)**.** *A clear field with dimensions* $(x, y)$ *is called a* proper clear field *if its clear field subgraph contains* $(x + 1)(y + 1)$ *unique faces.*

The previous theorem shows that all pairing clear fields are proper clear fields. The next two theorems consider clear fields on a fullerene whether or not they pair two vertices of degree five for some coloring. The first shows that there is a simple linear upper bound on the number of clear fields. The second shows that there is a logarithmic upper bound for the number of proper clear fields.

**Theorem 3.2.7.** *The number of clear fields is at most* $3n/2$.

*Proof.* The proof is divided into two parts. An upper bound will be determined for the number of clear fields with a $y$-dimension of zero and for those with a $y$-dimension greater than zero.

To count the number of clear fields with a $y$-dimension of zero, note that walking straight starting at a vertex of degree five eventually reaches exactly one vertex of degree five. (To prove this fact by contradiction, assume no vertex of degree five is reached, so the straight walk has infinite length. Since the number of arcs in the fullerene is finite, some arc of the straight walk must be repeated. However, to repeat the arc, the straight walk must have passed through the originating vertex of degree five where the straight direction is not defined.) Because the straight walk must arrive at a vertex of degree five, it corresponds with a clear field if the final vertex of degree five is not the originating vertex of degree five. There are five choices of a straight walk for each of the twelve pentagons for a total of 60 choices of such a straight walk. Any clear field with a $y$-dimension of zero corresponds to two of these walks, so an upper bound is 30.

The number of clear fields with a $y$-dimension greater than zero are counted by considering their wide right turns. Choose one of the $n/2 - 10$ dual vertices of degree six. For this vertex, choose one of the six pairs of one exiting arc and one entering arc that correspond to a wide right turn. At most one clear field uses this wide right turn

because walking straight from the exiting arc reaches at most one vertex of degree five and there is at most one way to walk straight from a vertex of degree five and reach the entering arc of the turn. Each clear field with a $y$-dimension greater than zero uses two of these $6(n/2-10)$ choices of a wide right turn, so an upper bound is $3(n/2-10)$.

Combining the two bounds gives a maximum number of $3n/2$ clear fields. □

The previous theorem shows that the number of clear fields is in $O(n)$. There is an infinite family of fullerenes each with a linear number of clear fields (with respect to $n$). The first three members of this family are shown in Figure 3.12. The fourth member is shown in Figure 3.10(b). A fullerene exists in this family for each value of $n = 24 + 12k$ for $k \geq 0$ and a face spiral $[25,\ 2.5]$ locates the pentagons at positions:

$$0,\ 1,\ 2,\ 3,\ 4,\ 6,\ (n/2)-5,\ (n/2)-3,\ (n/2)-2,\ (n/2)-1,\ (n/2),\ (n/2)+1.$$

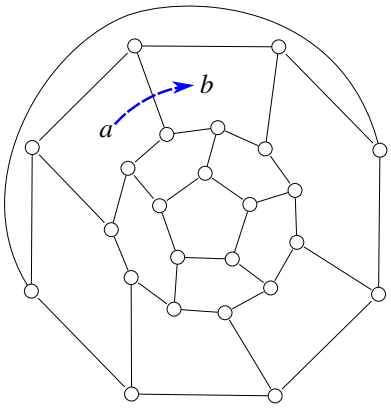
Such a fullerene has $k+1$ clear fields between the same two quintants: one quintant of the pentagon at position 6 (labeled $a$ in Figures 3.12 and 3.10(b)) and one quintant of the pentagon at position $(n/2)-5$ (labeled $b$). The $k+1$ clear fields have dimensions $(6i+1, k-i)$ for $0 \leq i \leq k$. Thus, some fullerenes have a linear number of clear fields. This is why when designing a method for arbitrary fullerenes, restricting the work to proper clear fields is beneficial as shown by the following result.

**Theorem 3.2.8.** *The number of proper clear fields is in $O(\lg n)$.*

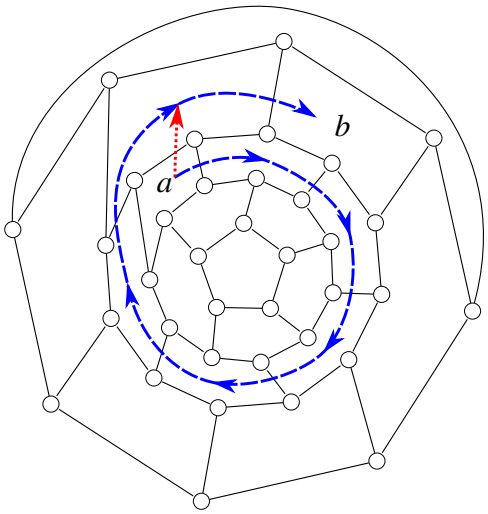
*Proof.* Let $k$ be the number of proper clear fields between a quintant $Q_a$ of a vertex $a$ of degree five and a quintant $Q_b$ of another vertex $b$ of degree five. Denote these $k$ clear fields as $C_1, C_2, \ldots, C_k$ with $C_i$ having dimensions $(x_i, y_i)$. Fixing $a$ as the origin $(0,0)$ and using $Q_a$, assign coordinates to the dual vertices (see Figure 3.6). A counter-intuitive situation emerges. Because $C_1$ pairs $a$ and $b$ using $Q_a$ with dimensions $(x_1, y_1)$, $b$ receives coordinate $(x_1, y_1)$. Similarly, $C_2$ pairs $a$ and $b$ using $Q_a$ with dimensions $(x_2, y_2)$, so $b$ also receives coordinate $(x_2, y_2)$. This also holds for clear fields $C_3, C_4, \ldots C_k$, so $b$ receives coordinates $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$.

The $y$-dimensions of clear fields $C_1, C_2, \ldots C_k$ must be unique because no two clear fields may share a wide right turn, as in the proof of Theorem 3.2.7. Likewise, the $x$-dimensions must be unique. Order the clear fields such that $y_1 < y_2 < \ldots < y_k$.

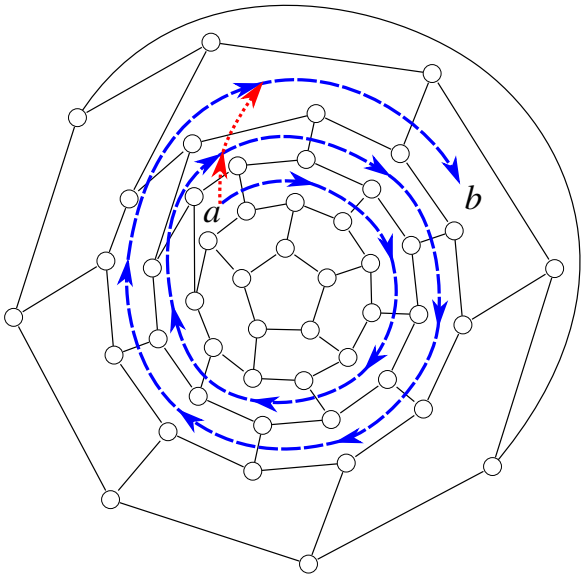
Consider two of these clear fields with $y$-dimensions $y_i$ and $y_j$ such that $y_i < y_j$. This non-intuitive situation is shown in Figure 3.13. Figure 3.13(a) shows clear field $C_i$ that starts at $a$ with coordinate $(0,0)$, goes straight to $(0, y_i)$ where it makes a




(a) For $k = 0$, $n = 24$ with clear field $(1, 0)$




(b) For $k = 1$, $n = 36$ with clear fields $(7, 0)$ and $(1, 1)$




(c) For $k = 2$, $n = 48$ with clear fields $(13, 0)$, $(7, 1)$, and $(1, 2)$

Figure 3.12: The first three fullerenes in an infinite family having $n = 24 + 12k$ and $k + 1$ clear fields pairing pentagons $a$ and $b$. The $x$-axis (dashed) and $y$-axis (dotted) of the quintant at $a$ is shown.

wide right turn and continues straight along the path marked $B$ to $b$ at $(x_i, y_i)$. At $b$, $C_i$ makes a sharp right turn and continues straight to $(x_i, 0)$ where it makes a wide right turn and continues straight along the path marked $A$ to return to $a$ at $(0, 0)$. Figure 3.13(b) shows clear field $C_j$ in a more typical illustration that shows the tight right turns at $a$ $(0, 0)$ and $b$ $(x_j, y_j)$ and the wide right turns at $(0, y_j)$ and

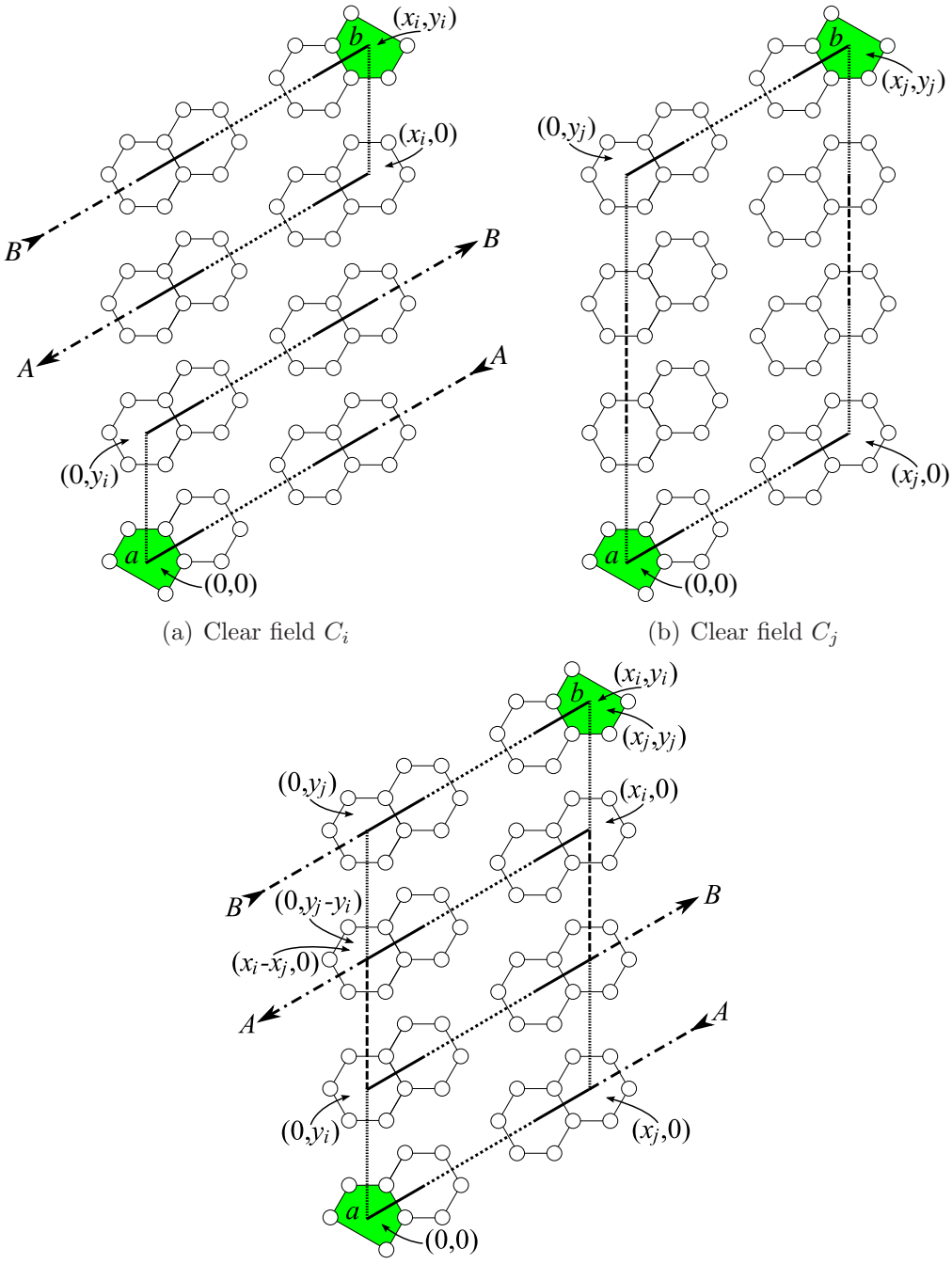
$(x_j, 0)$. Figure 3.13(c) combines these figures to show the full picture, which must be as shown because $C_i$ and $C_j$ both share quintants $Q_a$ and $Q_b$.

To show that $x_i > x_j$, assume for contradiction that $x_i < x_j$. In this case, the dual vertex at $(x_i, y_i)$ is in the interior of $C_j$, which implies the dual vertex at $(x_i, y_i)$ has degree six. Because $C_i$ pairs $a$ and $b$, the dual vertex at $(x_i, y_i)$ is $b$, which has degree five, which provides the contradiction. Therefore, $x_i > x_j$.

The subwalk of $C_i$ from $b$ to $a$ makes its wide right turn at $(x_i, 0)$. Therefore, the $x$-axis of $a$ meets the $y$-axis of $a$ at the dual vertex at coordinate $(x_i - x_j, 0)$, which is also $(0, y_j - y_i)$ because $C_i$ and $C_j$ share $Q_a$ and $Q_b$ (see Figure 3.13(c)). Clear field $C_i$ is a proper clear field, so it contains no repeated dual vertices. Therefore the vertex at $(0, y_j - y_i)$ is not one of the dual vertices at coordinates $(0, 0)$ through $(0, y_i)$, so $y_j - y_i > y_i$, which implies $2y_i + 1 \leq y_j$. This gives $2^{k-1}(y_1 + 1) - 1 \leq y_k$, which simplifies to $2^{k-1} - 1 \leq y_k$ because $y_1 \geq 0$. Clear field $C_k$ is a proper clear field, so $y_k \leq n/2 - 10$, the number of vertices of degree six. Therefore, $2^{k-1} < n/2$, which gives $k < \lg n$.

There is a constant number of quintant pairs so the total number of proper clear fields is in $O(\lg n)$. □

This completes the necessary background to describe an algorithm to find a maximum independent set of a fullerene.




(a) Clear field $C_i$

(b) Clear field $C_j$

(c) Both $C_i$ and $C_j$

Figure 3.13: Diagram for the proof of Theorem 3.2.8. Some coordinates are labeled, using $a$ as the origin. Lines of the same style represent the same distance. Arrows represent the continuation of a straight walk with the matched ends identified by letters $A$ and $B$.

# Chapter 4
# A Maximum Independent Set Algorithm

In his comments, Graver hinted at an algorithm for arbitrary fullerenes that constructs a maximum independent set by pairing the pentagons:

> "For an arbitrary fullerene, one could compute the size of the maximum independent set given by each of the 10395 possible pairings of its pentagonal faces and select the largest. There should be some way to quickly eliminate many pairings from consideration, leading to a reasonable algorithm for computing the independence number of an arbitrary fullerene." [31, p. 862]

The approach presented here follows this remark in that it is concerned with pairing the pentagons, but it overlooks some of the complexities. The count of 10395 ways to select six pairs of pentagons does not take into account the fact that the number of clear fields between two pentagons can be anywhere from zero up to a linear number in terms of $n$.

## 4.1 Algorithm Overview

The results of Chapter 3 show that a Graver coloring yields a pairing of the dual vertices of degree five using six proper clear fields (Theorem 3.2.5) that do not have any dual vertices in common (Theorem 3.2.4) and whose penalties together maximize $|W|$ by minimizing the sum of the penalties (Equation (3.2)). Such a selection of six clear fields and their colors is called an *optimal pairing*. Each of the six clear fields in an optimal pairing is colored white or black and the colors of five of the clear fields can be determined (by Theorem 3.2.1) if the color of the first is known. This is because the vertices not in the clear field subgraph of a pairing clear field must be white or black.

Consider a set of six proper clear fields with no two sharing a dual vertex. The set must pair the twelve vertices of degree five because each clear field pairs two vertices of degree five and no two clear fields have such a vertex in common. There are two ways to color the clear fields in the set. The first of these colorings can be obtained

by choosing one of the six clear fields and coloring it white and determining the colors of the other five clear fields using Theorem 3.2.1. The second of these colorings can be obtained from the first by switching the colors of all of the clear fields.

To find an optimal pairing, every choice of six proper clear fields with no dual vertices in common is considered. Theorem 3.2.8 implies that there are $O(\lg^6 n)$ such sets. For each set, the two possible colorings are determined and the penalty of each coloring is computed. Out of all possible sets and colorings, one with the minimum penalty is remembered because it is an optimal pairing.

A Graver coloring can be constructed from an optimal pairing. First, choose a pairing path for each of the six clear fields. One primal vertex incident to each of the primal edges corresponding to a dual edge on a pairing path is colored gray. The remaining vertices are colored white and black such that no two whites or blacks are adjacent and such that the clear fields receive the colors indicated by the optimal pairing.

The algorithm presented here takes the following procedure, which is broken into four phases:

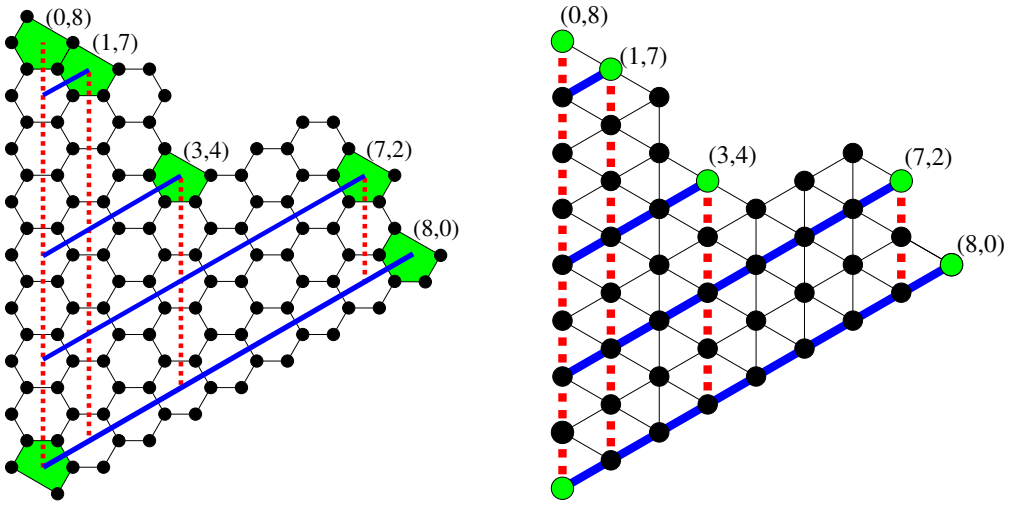
**Phase 1 ( 4.2):** Discover all of the proper clear fields by locating their dimensions and positions in the fullerene.

**Phase 2 ( 4.3):** Determine whether or not each pair of proper clear fields shares a dual vertex.

**Phase 3 ( 4.4):** Find an optimal pairing by considering each set of six proper clear fields such that no two clear fields share a dual vertex. This involves the following steps:

1. For each set of six proper clear fields, determine the color and then penalty for each clear field assuming the clear field with pentagon 0 in its clear field subgraph is white. Compute the total penalty of all six clear fields.
2. Compute the total penalty of all six clear fields obtained by switching the color of all of the clear fields in the previous step.
3. If either of the total penalties computed in the previous two steps is the smallest so far, remember the selection of six clear fields and the coloring.

**Phase 4 ( 4.5):** Construct a maximum independent set using the optimal pairing discovered in the previous phase.




(a) Primal (b) Dual

Figure 4.1: Part of a fullerene corresponding to a quintant of the lower left pentagon. Five pentagons are labeled by the coordinate system for the quintant, shown in both the primal and dual.

The input for the algorithm is a rotation system for the fullerene. It is assumed that the corresponding dual rotation system has been computed and is also available to the algorithm. It is also assumed that the twelve pentagonal faces have been located and have an arbitrary fixed numbering with labels 0 through 11.

## 4.2 Phase 1: Discover All Proper Clear Fields

The first phase of the algorithm is to discover all of the proper clear fields. The algorithm is quite complicated and is best explained in three parts, adding details each time. First, Section 4.2.1 gives an algorithm to locate all of the (not necessarily proper) clear fields in $O(n^2)$ time. Section 4.2.2 adds restrictions to the logic in order to locate only the proper clear fields in $O(n^2)$ time. Finally, Section 4.2.3 gives an algorithm that uses some tricks to reduce the running time to $O(n)$.

### 4.2.1 All Clear Fields in $O(n^2)$ Time

Each clear field corresponds to two quintants, one for each of its two vertices of degree five. To locate the clear fields, each of the 60 quintants is scanned to discover all of the clear fields that use it. In this way, each clear field is discovered twice. Figure 4.1(a) shows an example of a quintant of the lower left pentagon along with the coordinates of five other pentagons and Figure 4.1(b) shows the same on the dual. In order to have two uniquely defined quintants for each clear field (one per vertex of degree five), clear fields were defined to have an $x$-dimension of at least one. The clear field

```
1  foreach Pentagon fromPent ← 0 to 11 do   /* Pents numbered arbitrarily */
2  |  foreach Quintant do                  /* 5 quintants of current pentagon */
3  |  |  ScanQuintantAllCFs ();
```

**Algorithm 4.1**: An $O(n^2)$ algorithm for discovering all clear fields of a fullerene.

```
   ScanQuintantAllCFs () begin
4  |  MaxX ← ∞; MaxY ← 0;
5  |  y ← 1; currentFace ← face at (0, y) ;          /* Discover the y-axis */
6  |  while currentFace is not a pentagon do
7  |  |  MaxY ← MaxY + 1;
8  |  |  y ← y + 1; currentFace ← face at (0, y);
9  |  for y ← 0 to MaxY do                    /* Loop through y-axis faces */
10 |  |  currentFace ← face at (0, y);
11 |  |  ScanYValueAllCFs ();              /* Begin the y-value scan for y */
   end
```

```
   ScanYValueAllCFs () begin
12 |  x ← 0;
13 |  while x ≤ MaxX do  /* Rule 1:  Loop in x-direction, stop at MaxX */
14 |  |  x ← x + 1;
15 |  |  currentFace ← face at (x, y) ;     /* Visit next face in x-direction */
16 |  |  if currentFace is a pentagon then   /* Rule 2:  Detect clear fields */
17 |  |  |  toPent ← currentFace's pentagon number (0 to 11);
18 |  |  |  if fromPent < toPent then Record clear field (x, y);
19 |  |  |  MaxX ← x − 1;
   end
```

with the pentagon at coordinate $(0, 8)$ in Figure 4.1(a) is not in the quintant shown, but would receive dimensions $(8, 0)$ in the counter-clockwise adjacent quintant. The other four pentagons at coordinates $(1, 7)$, $(3, 4)$, $(7, 2)$, and $(8, 0)$ each correspond to a clear field for the given quintant.

The algorithm discovers clear fields for a given quintant in order of increasing $y$-dimension, with pseudo-code given in Algorithm 4.1 that includes Procedures `ScanQuintantAllCFs` and `ScanYValueAllCFs`. For each of the twelve pentagons, the algorithm does a *quintant scan* (Procedure `ScanQuintantAllCFs`) for each of its five quintants. To begin scanning a quintant, values $MaxX$ and $MaxY$ are initialized on Line 4. The value $MaxX$, which represents the maximum $x$-dimension that a clear field can have, is set to infinity. In Lines 5-8, the algorithm locates the $y$-axis of the quintant to record its location and its length, thereby determining $MaxY$. This is done by visiting the faces (using the dual graph) at coordinates $(0, 1)$, $(0, 2)$, ... until

a pentagon is reached. The value $MaxY$ is set to be the largest value such that the faces at coordinates $(0, 1)$ through $(0, MaxY)$ are all hexagons.

The algorithm proceeds by visiting the dual vertices of the $y$-axis in order from $(0, 0)$ through $(0, MaxY)$ (Line 9). A *$y$-value scan* is then performed at each $y$-value so that a quintant scan includes a $y$-value scan from each location on the $y$-axis (Procedure `ScanYValueAllCFs`). The $y$-value scan leaves the $y$-axis and continues along an $x$-direction walk to visit each dual vertex $(x, y)$ for $x = 1, 2, \ldots$ until it is stopped by one of the following two rules:

**Rule 1:** If $x > MaxX$ then do not visit the face at $(x, y)$, but instead increment $y$ and begin the next $y$-value scan.

**Rule 2:** If the face at $(x, y)$ is a pentagon, then:

> If the pentagon at $(x, y)$ is not the same pentagon as at $(0, 0)$, then a clear field with dimensions $(x, y)$ exists with this pentagon and should be recorded if it is not recorded already (recall that each clear field is discovered twice).
>
> Set $MaxX$ to $x - 1$.

These two rules work together to ensure that $MaxX$ always stores the largest allowable $x$-dimension for the current $y$-value and that this limit is never exceeded. When Rule 2 lowers the value of $MaxX$, Rule 1 is triggered when $x$ is incremented to prepare a visit to the next face for the current $y$-value.

Here is an example of the algorithm's execution when scanning the quintant shown in Figure 4.1. It begins with some initialization that sets $MaxX$ to $\infty$ and locates the $y$-axis as faces $(0, 0)$ through $(0, 7)$ because $(0, 8)$ is a dual vertex of degree five. This sets $MaxY$ to 7. Beginning the the $y$-value scan at $y = 0$, the algorithm visits the coordinates in order from $(0, 0)$ through $(8, 0)$, where it finds a vertex of degree five (Rule 2) and records a clear field having dimensions $(8, 0)$. The algorithm sets $MaxX = 7$ because no clear fields can have a $y$-dimension greater than 0 and an $x$-dimension greater than 7 because the pentagon at $(8, 0)$ would be located in the clear field subgraph of such a clear field. Next, the $y$-value scan at $y = 1$ begins and visits $(0, 1)$ through $(7, 1)$, at which point $x = MaxX$ and it stops (Rule 1). The $y$-value scan at $y = 2$ locates the degree five vertex at $(7, 2)$ (Rule 2), records the clear field, and sets $MaxX = 6$. The $y$-value scan at $y = 3$ ends due to Rule 1. At $y = 4$, the $y$-value scan locates the clear field with dimensions $(3, 4)$ and sets $MaxX = 2$. The $y$-value scans at $y = 5$ and $y = 6$ both end due to Rule 1 and at $y = 7$, the clear field

$(1,7)$ is recorded and $MaxX$ is set to 0 (Rule 2). The quintant scan is now complete because a $y$-value scan occurred at each location on the $y$-axis.

Rule 1 is implemented by Line 13 and Rule 2 is implemented by Lines 16-19. To ensure that each clear field is only recorded once, the clear field is only recorded if $fromPent$, the pentagon number (0 to 11) of the source pentagon at $(0,0)$, is smaller than $toPent$, the number of the discovered pentagon at $(x,y)$. This also ensures that the pentagons at $(0,0)$ and $(x,y)$ are different pentagons, as required for a clear field. In the example, if the pentagon at $(0,0)$ had be scanned after any or all of the pentagons at $(8,0)$, $(7,2)$, $(3,4)$ or $(1,7)$, then the corresponding clear fields would have already been recorded when such scans located the pentagon at $(0,0)$ and Rule 2 would cause them not to be recorded a second time here.

There is at most one clear field per $y$-value and if a pentagon is found, at $(x,y)$ then any clear field at a larger $y$-value must have a smaller $x$-value because the pentagon at $(x,y)$ cannot be in the clear field subgraph of another clear field. Thus, Rules 1 and 2 guarantee that every clear field subgraph contains only two pentagons because when a pentagon is found, $MaxX$ is lowered. This ensures that every pentagon that is found corresponds with a clear field. Furthermore, the algorithm scans every $y$-value for each quintant so that the algorithm discovers every clear field of the fullerene.

In order to show that the time complexity of Algorithm 4.1 is in $O(n^2)$, a lemma is given regarding the length of the $x$-axis.

**Lemma 4.2.1.** *Given a quintant, the length of the $x$-axis visited by Algorithm 4.1 is in $O(n)$.*

*Proof.* Recall that the $x$-axis begins at the pentagon at $(0,0)$ and continues in the straight direction, visiting the hexagons at $(1,0)$, $(2,0)$, $(3,0)$, ... until a pentagon is reached at some location $(x,0)$. The faces at $(0,0)$ and $(x,0)$ are the only two pentagons (possibly the same) on the $x$-axis.

Let $a_i$ be the arc that is followed from the dual vertex at $(i,0)$ to $(i+1,0)$ for $0 \leq i < x$. The claim is that $a_i$ is contained in the $x$-axis at most once. For contradiction, suppose that some arc $a_i$ is contained in the $x$-axis twice. Let $a_j$ be the second such occurrence $(i < j)$. Because the $x$-axis is straight, $a_{i-k}$ must be the same as $a_{j-k}$ for any $0 \leq k \leq i$, so $a_0$ must be the same as $a_{j-i}$. The face at $(0,0)$ is a pentagon, which implies that the face at $(j-i,0)$ is the same pentagon. This is a contradiction because $0 < j-i < x$ so the face at $(j-i,0)$ must be a hexagon.

Therefore, the arcs of the $x$-axis must be unique and because the number of dual arcs of a fullerene is in $O(n)$, the length of the $x$-axis must be in $O(n)$. □

The following lemma gives the same result as the previous lemma, but for the $y$-axis. The proof uses the same logic as in the previous lemma and is therefore omitted.

**Lemma 4.2.2.** *Given a quintant, the length of the y-axis visited by Algorithm 4.1 is in* $O(n)$.

These two lemmas give the following complexity result.

**Theorem 4.2.3.** *The time complexity of Algorithm 4.1 is in* $O(n^2)$.

*Proof.* There are 60 quintants and for each quintant scan it follows from Lemma 4.2.2 that the number of $y$-value scans is in $O(n)$. Each quintant scan begins with $MaxX$ set to $\infty$. It follows from Lemma 4.2.1 that after the $y$-value scan at $y = 0$ is complete, the value of $MaxX$ is in $O(n)$ and $MaxX + 1$ faces were visited during the $y$-value scan. Rule 1 ensures that no more than $MaxX$ faces are visited in each subsequent $y$-value scan. Thus, the number of faces visited in each of the $y$-value scans is in $O(n)$. Therefore, the time complexity of each quintant scan is in $O(n^2)$. A constant number of quintant scans gives the result. □

### 4.2.2 All Proper Clear Fields in $O(n^2)$ Time

Algorithm 4.1 gave a procedure to find all of the clear fields in a fullerene. This section adds two additional rules in order to limit the clear fields to only proper clear fields. The pseudo-code for discovering all of the proper clear fields in $O(n^2)$ time is given in Algorithm 4.2 after some new theoretical results are presented.

Let $a$ be the pentagon being scanned during a quintant scan and suppose a clear field to pentagon $b$ is found. This clear field $\angle ab$ is a proper clear field if and only if it contains no repeated dual vertices. That is, $\angle ab$ is a proper clear field if and only if all of the following conditions are true:

**Condition 1:** Walk $\overrightarrow{ab}$ contains no repeated dual vertices.

**Condition 2:** Walk $\overrightarrow{ab}$ shares no dual vertices with walk $\overrightarrow{ba}$ other than $a$ and $b$.

**Condition 3:** Walk $\overrightarrow{ba}$ contains no repeated dual vertices.

The following two theorems apply to any clear field in a fullerene, not just those that are in an optimal solution. Theorem 4.2.4 shows that Condition 1 implies Condition 3, rendering it unnecessary to independently verify the latter when determining if a clear field is proper or not. Theorem 4.2.5 shows that given that Condition 1 is true, Condition 2 can be relaxed to something easier for the algorithm to verify.

**Theorem 4.2.4.** *Let $\square ab$ be a clear field in a fullerene with dimensions $(x, y)$ such that walk $\overrightarrow{ab}$ contains no repeated dual vertices. Then walk $\overrightarrow{ba}$ contains no repeated dual vertices.*

*Proof.* If $y = 0$ then the proof is trivial because $\overrightarrow{ab}$ and $\overrightarrow{ba}$ contain the same set of dual vertices. Henceforth, assume $y > 0$.

Define the coordinate system for $\square ab$ to be $(0, 0)$ at $a$. Then $b$ is at coordinate $(x, y)$, $\overrightarrow{ab}$ makes its wide right turn at $(0, y)$ and $\overrightarrow{ba}$ makes its wide right turn at $(x, 0)$.

Let walk $\overrightarrow{ab_1}$ be the subwalk of $\overrightarrow{ab}$ that contains the first $y + 1$ dual vertices. Let walk $\overrightarrow{ab_2}$ be the subwalk of $\overrightarrow{ab}$ that contains the last $x+1$ dual vertices. The only dual vertex in common with $\overrightarrow{ab_1}$ and $\overrightarrow{ab_2}$ is the one at coordinate $(0, y)$, which is the last dual vertex of $\overrightarrow{ab_1}$ and the first of $\overrightarrow{ab_2}$. Let $u_0, u_1, \ldots, u_{x+y}$ be the dual vertices visited by $\overrightarrow{ab}$ such that $\overrightarrow{ab_1}$ contains $u_0$ through $u_y$ and $\overrightarrow{ab_2}$ contains $u_y$ through $u_{x+y}$.

Similarly, define walks $\overrightarrow{ba_1}$ and $\overrightarrow{ba_2}$ to contain the first $y + 1$ and last $x + 1$ dual vertices of $\overrightarrow{ba}$, respectively. Let $v_0, v_1, \ldots, v_{x+y}$ be the dual vertices visited by $\overrightarrow{ba}$ such that $\overrightarrow{ba_1}$ contains $v_0$ through $v_y$ and $\overrightarrow{ba_2}$ contains $v_y$ through $v_{x+y}$.

This proof is broken into three parts: (i) it is shown that the dual vertices of $\overrightarrow{ba_1}$ are unique, (ii) it is shown that the dual vertices of $\overrightarrow{ba_2}$ are unique, and (iii) it is shown that the only dual vertex in common between $\overrightarrow{ba_1}$ and $\overrightarrow{ba_2}$ is $v_y$ at coordinate $(x, 0)$ where $\overrightarrow{ba}$ makes its wide-right turn.

**(i)** It will now be shown that the dual vertices of $\overrightarrow{ba_1}$ are unique. For contradiction, suppose $\overrightarrow{ba_1}$ contains the the same dual vertex twice. This assumption will give the contradiction that the vertices of $\overrightarrow{ab_1}$ are not unique.

Choose $i$ and $j$ such that $0 < i < j \leq y$ and $i$ is the smallest value such that dual vertices $v_i$ and $v_j$ are two copies of a repeated vertex in $\overrightarrow{ba_1}$. That is, $v_i \equiv v_j$.

There are six cases to consider, based on how many clockwise positions arc $(v_{j-1}, v_j)$ is after arc $(v_{i-1}, v_i)$. First, the cases for zero and three clockwise positions are ruled out.

**Case 0 clockwise positions:** Here, arc $(v_{j-1}, v_j)$ is the same as arc $(v_{i-1}, v_i)$. In this case, $v_{i-1} \equiv v_{j-1}$, which contradicts the assumption that $i$ and $j$ are the smallest values such that $v_i \equiv v_j$.

**Case 3 clockwise positions:** Here, arc $(v_{j-1}, v_j)$ is three clockwise positions after arc $(v_{i-1}, v_i)$. Let $k = \lfloor (j - i)/2 \rfloor$. In this case, $v_{i+1} \equiv v_{j-1}$, $v_{i+2} \equiv v_{j-2}$, ..., $v_{i+k} \equiv v_{j-k}$. This is an impossible situation because $\overrightarrow{ba_1}$ is a straight walk.

Four cases remain to be considered. First, note that walk $\overrightarrow{ab_1}$ begins at $a \equiv u_0$ and goes in the straight direction to $u_y$, running in a sense parallel to $\overrightarrow{ba_1}$ but in the

opposite direction. Dual vertex $u_{y-i}$ is the dual vertex of $\overrightarrow{ab_1}$ that corresponds with $v_i$ in that they both receive the same $y$-coordinate. Similarly, $u_{y-j}$ corresponds with $v_j$.

The remaining four cases are pictured in Figure 4.2 with dual vertices $a$, $b$, $u_y$, $u_{y-i}$, $u_{y-j}$, and $v_i \equiv v_j$ labeled as well as some other case-specific dual vertices. Each case contains a shaded parallelogram-shaped region that arises from the clear field overlapping itself and has $x+1$ dual vertices and $x$ arcs on each side.

**Case** 1 **clockwise position [Figure 4.2(a)]:** Here, arc $(v_{j-1}, v_j)$ is one clockwise position after arc $(v_{i-1}, v_i)$. In this case, vertex $u_{y-i+x} \equiv u_{y-j}$. Because $i < j$, $y - i + x > y - j$, so the vertices of $\overrightarrow{ab_1}$ are not unique, which provides the contradiction.

**Case** 2 **clockwise positions [Figure 4.2(b)]:** Here, arc $(v_{j-1}, v_j)$ is two clockwise positions after arc $(v_{i-1}, v_i)$. In this case, vertex $u_{y-i+2x} \equiv u_{y-j-x}$. Because $i < j$, $y - i + 2x > y - j - x$, so the vertices of $\overrightarrow{ab_1}$ are not unique, which provides the contradiction.

**Case** 4 **clockwise positions [Figure 4.2(b)]:** Here, arc $(v_{j-1}, v_j)$ is four clockwise positions after arc $(v_{i-1}, v_i)$. In this case, vertex $u_{y-i-x} \equiv u_{y-j+2x}$. It must be that $y-i-x > y-j+2x$ because the dashed loop in the figure cannot have zero length. Thus, the vertices of $\overrightarrow{ab_1}$ are not unique, which provides the contradiction.

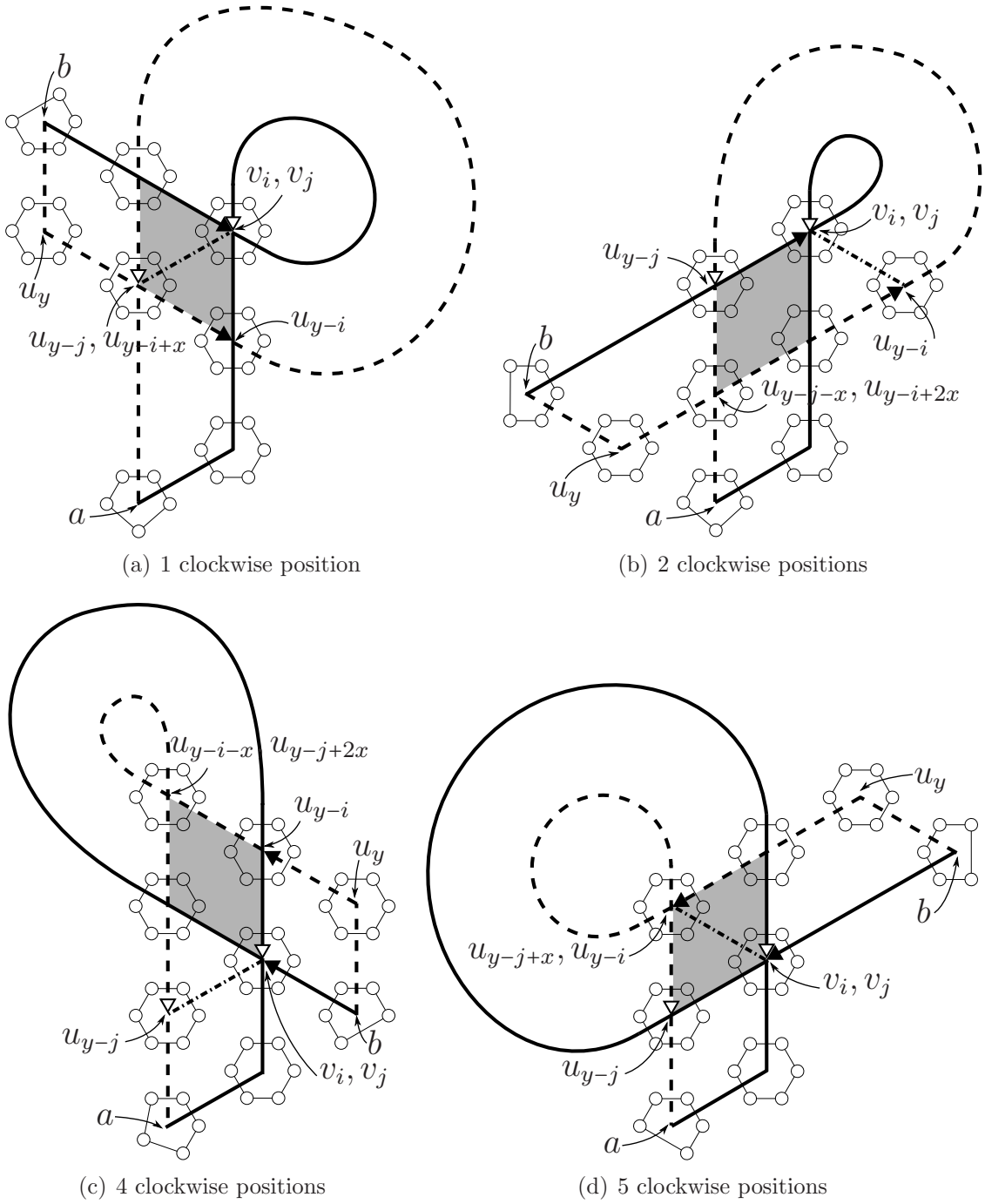
**Case** 5 **clockwise positions [Figure 4.2(b)]:** Here, arc $(v_{j-1}, v_j)$ is five clockwise positions after arc $(v_{i-1}, v_i)$. In this case, vertex $u_{y-i} \equiv u_{y-j+x}$. It must be that $y - i > y - j + x$ because the dashed loop in the figure cannot have zero length. Thus, the vertices of $\overrightarrow{ab_1}$ are not unique, which provides the contradiction.

This concludes the proof of part (i).

**(ii)** An argument very similar to that in part (i) can be used to show that the dual vertices of $\overrightarrow{ba_2}$ are unique.

**(iii)** To show that walks $\overrightarrow{ba_1}$ and $\overrightarrow{ba_2}$ have only vertex $v_y$ in common, the opposite is assumed and a contradiction is reached. That is, suppose dual vertices $v_i$ and $v_j$ are the same for $0 < i < y < j < x + y$. Six cases are considered based on the number of clockwise positions arc $(v_{j-1}, v_j)$ is after arc $(v_{i-1}, v_i)$.

**Case** 1 **clockwise position [Figure 4.3(a)]:** Here, arc $(v_{j-1}, v_j)$ is one clockwise position after arc $(v_{i-1}, v_i)$. Vertex $v_i$ is at coordinate $(x, y - i)$ and $v_j$ is at




(a) 1 clockwise position

(b) 2 clockwise positions

(c) 4 clockwise positions

(d) 5 clockwise positions

Figure 4.2: Diagrams for the proof of Theorem 4.2.4 part (i) labeled with the number of clockwise positions arc $(v_{j-1}, v_j)$ is after arc $(v_{i-1}, v_i)$. Some dual vertices are labeled. The dashed and solid lines represent $\overrightarrow{ab}$ and $\overrightarrow{ba}$, respectively. Black arrows on the dashed (solid) line indicate the dual vertex of $\overrightarrow{ab_1}$ ($\overrightarrow{ba_1}$) that is distance $i$ from $u_y$ ($b$). Similarly, the white arrows represent distance $j$. For clarity, dash-dot lines connect $v_i$ with $u_{y-i}$ or $v_j$ with $u_{y-j}$.

coordinate $(x + y - j, 0)$. The number of clockwise positions implies that $b$ is inside the clear field (at coordinate $(x + y - j, y - i)$), for a contradiction.

**Case 2 clockwise positions [Figure 4.3(b)]:** Here, arc $(v_{j-1}, v_j)$ is two clockwise positions after arc $(v_{i-1}, v_i)$. Vertex $v_i$ is at coordinate $(x, y - i)$ and $v_j$ is at coordinate $(x + y - j, 0)$. The number of clockwise positions implies that $\overrightarrow{ba}$ crosses $\overrightarrow{ab}$ at $v_{j+i}$, which is at coordinate $(x + y - j - i, 0)$. Thus, $b$ is inside the clear field (at coordinate $(x + y - j - i, y - i)$), for a contradiction.

**Case 3 clockwise positions [Figure 4.3(c)]:** Here, arc $(v_{j-1}, v_j)$ is three clockwise positions after arc $(v_{i-1}, v_i)$. In this case, $\overrightarrow{ba}$ continues in the straight direction from $v_j$ and must therefore reach $b$ at $v_{j+i}$, which cannot result in a clear field and is a contradiction.

**Case 4 clockwise positions [Figure 4.3(d)]:** Here, arc $(v_{j-1}, v_j)$ is four clockwise positions after arc $(v_{i-1}, v_i)$. In this case, Euler's formula shows that there must be three pentagons in the shaded region, but that region is inside the clear field, which is a contradiction.

**Case 5 clockwise positions [Figure 4.3(e)]:** Here, arc $(v_{j-1}, v_j)$ is five clockwise positions after arc $(v_{i-1}, v_i)$. In this case, Euler's formula shows that there must be four pentagons in the shaded region, but that region is inside the clear field, which is a contradiction.

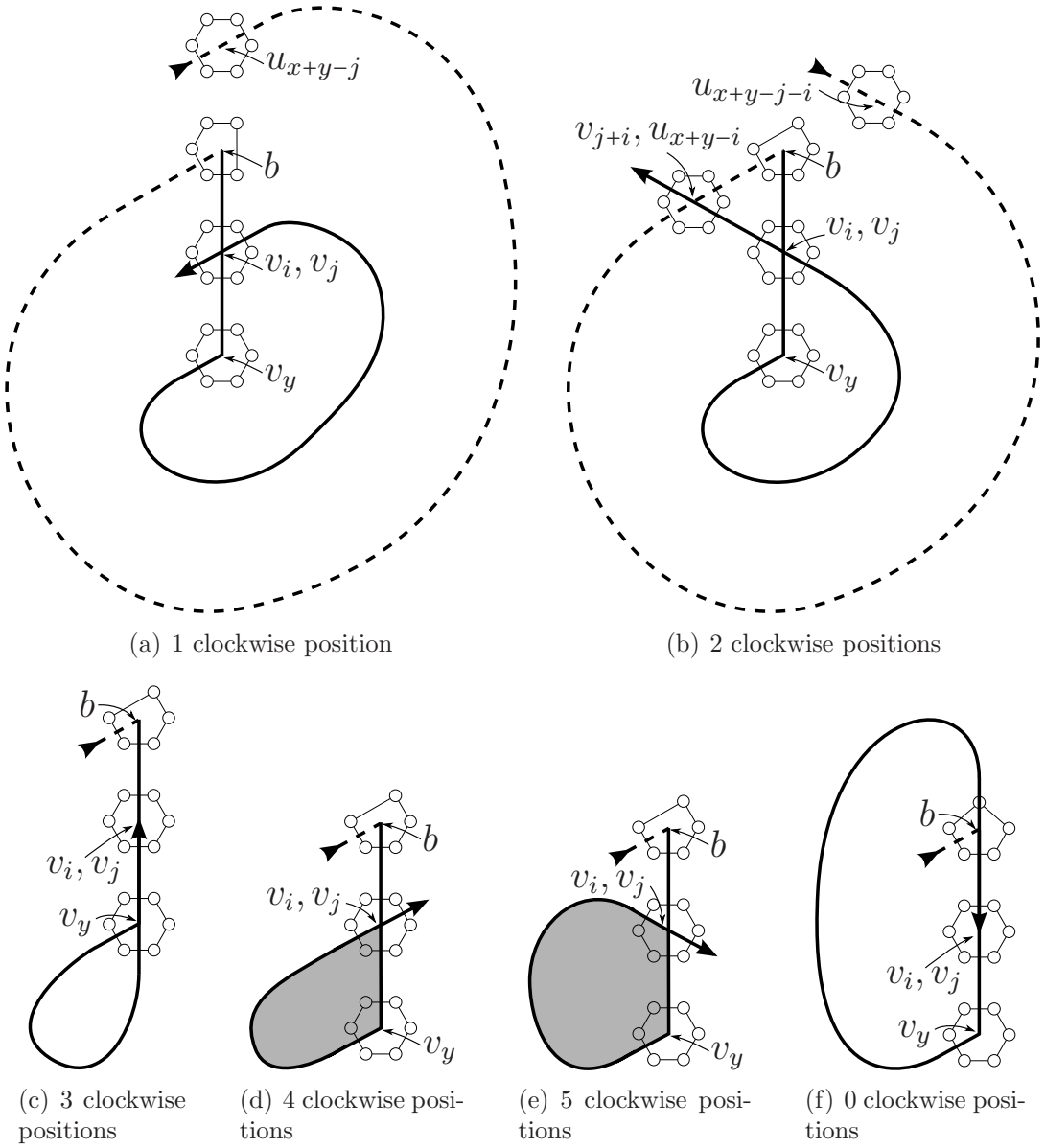
**Case 0 clockwise positions [Figure 4.3(f)]:** Here, arc $(v_{j-1}, v_j)$ is zero clockwise positions after arc $(v_{i-1}, v_i)$. This situation is impossible, because $b$ must be the same as $v_{j-i}$ and the straight direction is not defined through a hexagon.

A contradiction was reached in each case, so the proof is complete. □

**Theorem 4.2.5.** *Let $\square ab$ be a clear field in a fullerene with dimensions $(x, y)$ such that $\square ab$ contains a repeated vertex and walk $\overrightarrow{ab}$ contains no repeated dual vertices. Let $u_0, u_1, \ldots, u_{x+y}$ be the dual vertices visited by $\overrightarrow{ab}$ and let $v_0, v_1, \ldots, v_{x+y}$ be the dual vertices visited by $\overrightarrow{ba}$. Then the following hold:*

*(i) There is a choice of $1 \leq k_1 \leq y$ and $y \leq \ell_1 \leq x + y - 1$ such that $u_{k_1} \equiv v_{\ell_1}$ and arc $(v_{\ell_1}, v_{\ell_1+1})$ is 1 clockwise position after arc $(u_{k_1-1}, u_{k_1})$.*

*(ii) There is a choice of $y \leq k_2 \leq x + y - 1$ and $1 \leq \ell_2 \leq y$ such that $u_{k_2} \equiv v_{\ell_2}$ and arc $(u_{k_2}, u_{k_2+1})$ is 1 clockwise position after $(v_{\ell_2-1}, v_{\ell_2})$.*




(a) 1 clockwise position

(b) 2 clockwise positions

(c) 3 clockwise positions

(d) 4 clockwise positions

(e) 5 clockwise positions

(f) 0 clockwise positions

Figure 4.3: Diagrams for the proof of Theorem 4.2.4 part (iii) labeled with the number of clockwise positions arc $(v_{j-1}, v_j)$ is after arc $(v_{i-1}, v_i)$. Some dual vertices are labeled. Solid lines represent $\overrightarrow{ba}$ and dashed lines $\overrightarrow{ab}$ with arrows indicating $\square ab$'s clockwise direction.

*That is, $\square ab$ must have the situation pictured in Figures 3.10 and 3.11.*

*Proof.* From Theorem 4.2.4, walk $\overrightarrow{ba}$ contains no repeated dual vertices. The proof of Theorem 3.2.5 considers the situation in which $\square ab$ contains a repeated dual vertex, yet $\overrightarrow{ab}$ contains no repeated dual vertices and $\overrightarrow{ba}$ contains no repeated dual vertices. There the conclusion is as stated and is not reproduced here. □

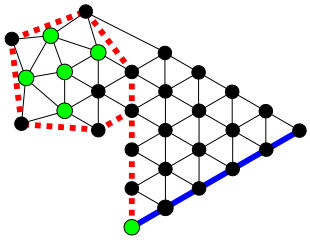


Figure 4.4: A $y$-axis sharing a dual vertex with itself

The strategy to eliminate non-proper clear fields is to create two additional rules. Rule 3 will be created to ensure Condition 1: walk $\overrightarrow{ab}$ contains no repeated dual vertices. Rule 4 will be created to ensure Condition 2 given that Condition 1 holds: walks $\overrightarrow{ab}$ and $\overrightarrow{ba}$ have only $a$ and $b$ in common given that walk $\overrightarrow{ab}$ contains no repeated dual vertices.

First, Rule 3 is created to ensure that walk $\overrightarrow{ab}$ contains no repeated dual vertices. The dual vertices in $\overrightarrow{ab}$ are those along the $y$-axis from $(0, 0)$ to $(0, y_{ab})$ then along the $y$-value scan from $(0, y_{ab})$ to $(x_{ab}, y_{ab})$. To ensure that none of these vertices are repeated, the algorithm does the following three things:

1. Ensure that no face is included in the $y$-axis more than once. Figure 4.4 shows an example of the $y$-axis containing a repeated dual vertex.

2. Ensure that the $y$-value scan that began at $(0, y)$ does not visit a face that was previously visited during the same $y$-value scan.

3. Ensure that the $y$-value scan that began at $(0, y)$ does not visit a face on the $y$-axis with a $y$-value smaller than $y$.

These three items become Rule 3 as follows.

**Rule 3:** This rule is composed of three parts:

**Rule 3a:** Set $MaxY$ to be the largest value such that the faces at $(0, 1)$, $(0, 2)$, ..., $(0, MaxY)$ are each *unique* hexagons.

**Rule 3b:** During a $y$-value scan, if $x$ is incremented such that the face at $(x, y)$ was already visited during the $y$-value scan, then set $MaxX$ to $x - 1$. Note that Rule 1 will cause the $y$-value scan to end here.

**Rule 3c:** During a $y$-value scan, if $x$ is incremented such that the face at $(x, y)$ is a location $(0, y_x)$ on the $y$-axis of the quintant such that $0 < y_x < y$, then set $MaxX$ to $x - 1$. Note that Rule 1 will cause the $y$-value scan to end here.

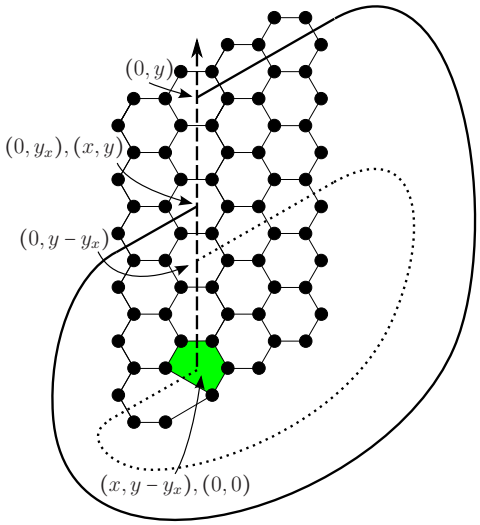
It turns out that Rule 3c is unnecessary due to Rules 1, 2, 3a and 3b because in the cases in which it applies, the other rules will be sufficient for guaranteeing that no clear field is detected with an $x$ value larger than where the $y$-axis was met. The following proof formalizes the argument by considering all possible cases.

**Theorem 4.2.6.** *The implementation of Rules 1, 2, 3a, and 3b render Rule 3c unnecessary; no non-proper clear fields will be detected at a result of omitting Rule 3c.*

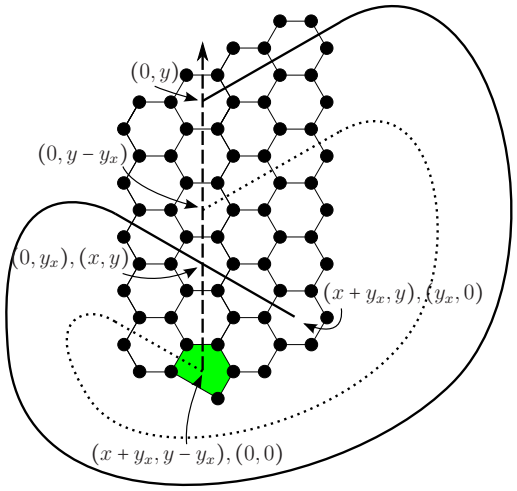
*Proof.* To show that Rule 3c is unnecessary, consider the state of the algorithm at which time Rule 3c would be in effect. The algorithm must have just incremented $x$ such that the face at $(x, y)$ is a face at $(0, y_x)$ on the $y$-axis of the quintant such that $0 < y_x < y$. The face $(x, y) \equiv (0, y_x)$ was visited immediately after the visit to $(x-1, y)$. There are six cases to considered that correspond to the number of clockwise positions that the arc from $(x-1, y)$ to $(x, y)$ is after the arc from $(0, y_x - 1)$ to $(0, y_x)$. In each case, it is assumed that the faces located at $(i, j)$ for $0 \leq i \leq x$ and $0 \leq j \leq y$ are hexagons, otherwise Rule 2 would have set $MaxX < x$ and $(x, y)$ would not be visited by the algorithm. Knowledge that these faces are hexagons provides necessary information about the fullerene geometry.

**Case** 1 **clockwise position [Figure 4.5(a)]:** The arc from $(x-1, y)$ to $(x, y)$ is 1 clockwise position after the arc from $(0, y_x - 1)$ to $(0, y_x)$ so the fullerene geometry implies $(x, y-i) \equiv (0, y_x-i)$ for all $0 \leq i \leq y_x$. In particular, $(x, y-y_x) \equiv (0, 0)$, which is the dual vertex $a$ that corresponds to a pentagon. The fact that the algorithm visited $(x, y)$ implies that the face at $(x, y - y_x) \equiv (0, 0)$ is a hexagon, otherwise Rules 1 and 2 would have prevented the visit to $(x, y)$. This is a contradiction so this situation can never occur.

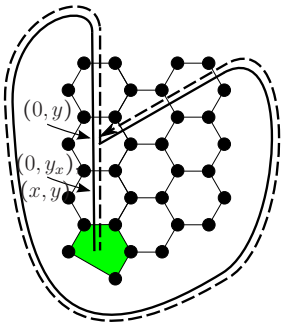
**Case** 2 **clockwise positions [Figure 4.5(b)]:** The arc from $(x-1, y)$ to $(x, y)$ is 2 clockwise positions after the arc from $(0, y_x - 1)$ to $(0, y_x)$ so the fullerene geometry implies $(x + i, y - i) \equiv (0, y_x - i)$ for all $0 \leq i \leq y_x$. In particular, $(x+y_x, y-y_x) \equiv (0, 0)$, which is the dual vertex $a$ that corresponds to a pentagon. This means $MaxX < x + y_x$ due to Rule 2 from the $y$-value scan that began at $(0, y - y_x)$. Also, $(x + j, y) \equiv (j, y_x - j)$ for all $0 \leq j \leq y_x$ and the faces at these coordinates must be hexagons because they were previously visited by the algorithm. Therefore, no clear field can exist with dimensions $(x + j, y)$ for $1 \leq j \leq MaxX$ so ignoring Rule 3c's detection of $(x, y) \equiv (0, y_x)$ does not lead to the discovery of a clear field during the execution of the algorithm for the given $y$-value. It also does not cause problems when the algorithm later uses a



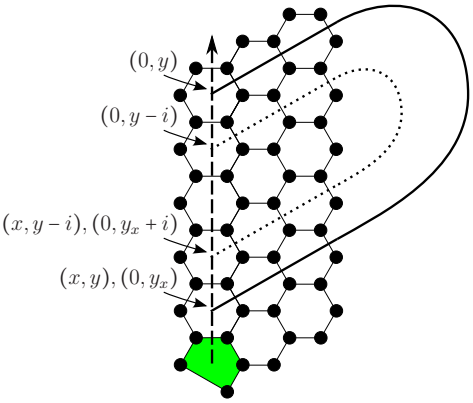

(a) 1 clockwise position



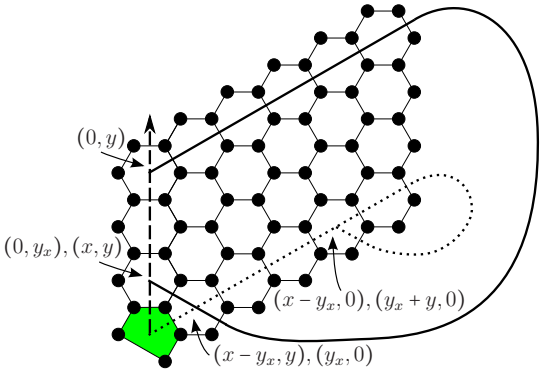

(b) 2 clockwise positions



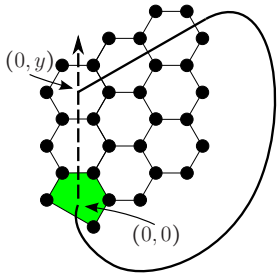

(c) 3 clockwise positions




(d) 4 clockwise positions




(e) 5 clockwise positions




(f) 0 clockwise positions

Figure 4.5: Illustrations for the proof of Theorem 4.2.6. The six situations in which a $y$-value scan from $(0, y)$ (solid) of length $x$ can share a face at $(0, y_x)$ such that $y_x \leq y$. Situation are labeled with the clockwise positions that the arc from $(x-1, y)$ to $(x, y)$ is after the arc from $(0, y_x - 1)$ to $(0, y_x)$. Coordinates of selected faces are labeled. The $y$-axis is dashed. Straight walks relevant to the proof (dotted) are drawn.

$y$-value of $y+\ell$ for $0 > \ell$ because $(x+k, y+\ell) \equiv (\ell, y_x - k)$ for all $0 \le k < y_x$ (all of which correspond to hexagons) and because $MaxX < x + y_x$.

**Case 3 clockwise positions [Figure 4.5(c)]:** It is impossible for this case to occur because having the arc from $(x-1, y)$ to $(x, y)$ be 3 clockwise positions after the arc from $(0, y_x - 1)$ to $(0, y_x)$ would require the face at $(x - (y - y_x), y)$ to also be at $(0, y)$, which would have caused $MaxX = x - (y - y_x) - 1 < x$.

**Case 4 clockwise positions [Figure 4.5(d)]:** For brevity, let $z = \lfloor (y - y_x - 1)/2 \rfloor$. The arc from $(x-1, y)$ to $(x, y)$ is 4 clockwise positions after the arc from $(0, y_x - 1)$ to $(0, y_x)$ so the fullerene geometry implies $(x, y-i) \equiv (0, y_x + i)$ for all $0 \le i \le z$. For any choice of $i$ in this range the faces at coordinates $(0, y-i)$ to $(x, y-i)$ must all be hexagons because $x \le MaxX$. Also for any choice of $i$, Euler's formula [20] shows (using a method similar to that in the proof of Theorem 3.2.5) that there are three pentagons in the fullerene subgraph bounded by the faces in the straight direction from $(0, y-i)$ to $(x, y-i) \equiv (0, y_x + i)$ then back up the faces of the $y$-axis to $(0, y-i)$. Because the faces at $(0, y-z)$ to $(x, y-z)$ are all hexagons, it must be that $y - y_x$ is even so that the subgraph bounded by the faces at $(0, y-z)$ to $(x, y-z) \equiv (0, y_x + z)$ $\equiv (0, y-z-2)$ then back up two faces of the $y$-axis to $(0, y-z)$ contains three pentagons. Since $x \le MaxX$, the faces from $(0, y-z-1)$ to $(x, y-z-1)$ must all be hexagons, but this is not possible because the face at $(j, y-z-1)$ is adjacent to both $(j, y-z)$ and $(x-j, y-z)$ for all $0 \le j \le \lfloor x/2 \rfloor$, which leaves no possible location for three pentagons. This is a contradiction so this situation can never occur.

**Case 5 clockwise positions [Figure 4.5(e)]:** The arc from $(x-1, y)$ to $(x, y)$ is 5 clockwise positions after the arc from $(0, y_x - 1)$ to $(0, y_x)$ so the fullerene geometry implies $(x - y_x, y) \equiv (y_x, 0)$, a dual vertex on the $x$-axis and $(x - y_x, y - i) \equiv (y_x + i, 0)$ for all $0 \le i \le y$. In particular, $(x - y_x, 0) \equiv (y_x + y, 0)$ so when the algorithm visited $(x - y_x, 0)$, it would have detected the previous visit to the hexagon at $(y_x + y, 0)$, triggering Rule 3b. This implies $MaxX < x - y_x < x$, which is a contradiction so this situation can never occur.

**Case 0 clockwise positions [Figure 4.5(f)]:** It is impossible for this case to occur because having the arc from $(x-1, y)$ to $(x, y)$ be the same as the arc from $(0, y_x - 1)$ to $(0, y_x)$ would require the face at $(x - y_x, y)$ to also be at $(0, 0)$, which is a pentagon so $MaxX$ would have been set to at most $x - y_x$.

```
1  Create arrays FaceVisits and YAxis of size n/2 + 2: hold info about face visits and
   y-axis;
2  foreach Pentagon fromPent ← 0 to 11 do   /* Pents numbered arbitrarily */
3  |  foreach Quintant do                    /* 5 quintants of current pentagon */
4  |  |  ScanQuintantProperCFs ();
```

**Algorithm 4.2**: An $O(n^2)$ algorithm for discovering all proper clear fields of a fullerene.

```
   ScanQuintantProperCFs () begin
5  |  foreach Face f ← 0 to n/2+2 do          /* Initialize array values */
6  |  |  FaceVisits[f].x ← null; FaceVisits[f].y ← null;
7  |  |  YAxis[f].y ← null; YAxis[f].MaxXFromXAxis ← ∞;
8  |  MaxX ← ∞; MaxY ← 0;
9  |  y ← 1; currentFace ← face at (0, y) ;          /* Discover the y-axis */
10 |  while currentFace is not a pentagon AND YAxis[currentFace].y ≠ null do
   |  /* Rule 3a:  Unique hexagons only */
11 |  |  MaxY ← MaxY + 1;
12 |  |  YAxis[currentFace].y ← y;
13 |  |  y ← y + 1; currentFace ← face at (0, y);
14 |  for y ← 0 to MaxY do                    /* Loop through y-axis faces */
15 |  |  currentFace ← face at (0, y);
16 |  |  FaceVisits[currentFace].x ← 0; FaceVisits[currentFace].y ← y;
17 |  |  MaxX ← min(MaxX, YAxis[currentFace].MaxXFromXAxis);
18 |  |  ScanYValueProperCFs ();          /* Begin the y-value scan for y */
   |  end
```

In summary, the cases of 1, 3, 4, 5, and 0 clockwise positions cannot occur and the case of 2 clockwise positions does not cause the algorithm to change its output. □

Now that Rule 3 has been specified and justified, the implementation is presented. To implement Rule 3a, the algorithm records more information about the location of the $y$-axis through the use of an array $YAxis$ created at the beginning of the algorithm (Line 1) and initialized at the beginning of the quintant scan (Line 7). When the $y$-axis is pre-walked at the beginning of the quintant scan, $YAxis$ stores the $y$-value of each face on the $y$-axis (Line 12). Also during the pre-walking to set $MaxY$, this array is checked (Line 10) to ensure that the faces at $(0,1)$, $(0,2)$, ..., $(0, MaxY)$ are each unique hexagons to satisfy Rule 3a.

To implement Rule 3b, the algorithm records more information as it visits each face in order to later detect repeat visits. An array $FaceVisits$ was added (Line 1) and initialized (Line 6) for this purpose. The array stores the $x$- and $y$-values of the most recent visit to the face (Lines 16 and 33). This information can be used

```
   ScanYValueProperCFs () begin
19   x ← 0;
20   while x ≤ MaxX do                          /* Loop through x-direction */
21     x ← x + 1;
22     currentFace ← face at (x, y) ;          /* Visit next face in x-direction */
23     if currentFace is a pentagon then       /* Detect & record clear fields */
24       toPent ← currentFace's pentagon number (0 to 11);
25       if fromPent < toPent then Record clear field (x, y);
26       MaxX ← x − 1;
27     else
         /* Rule 4: the y-value scan indicates a wrap-around here      */
28       if wrap-around at currentFace then
29         if y = 0 then                        /* x-axis indicates a wrap-around */
30           YAxis[currentFace].MaxXFromXAxis ← x − 1;
         /* Rule 3b: currentface was prev visited by this y-value scan  */
31       if FaceVisits[currentFace].y = y then
32         MaxX ← x − 1;
       /* Record that (x, y) was visited                                 */
33     FaceVisits[currentFace].x ← x; FaceVisits[currentFace].y ← y;
   end
```

to determine if the currently visited face at $(x, y)$ has been visited previously during this $y$-value scan by checking if the face has already been marked as visited with the same $y$-value (Line 31).

To finish this section, Rule 4 is explained. Rule 4 enforces the requirement that walk $\overrightarrow{ab}$ shares no dual vertices with walk $\overrightarrow{ba}$ given that Rule 3 filters out the clear fields in which walk $\overrightarrow{ab}$ contains a repeated dual vertex. From Theorem 4.2.5, if $\overrightarrow{ab}$ contains no repeated dual vertices but shares dual vertices with $\overrightarrow{ba}$ other than $a$ and $b$, then it must be the situation pictured in Figures 3.10 and 3.11: a vertex of $\overrightarrow{ba}$ at or after the wide right turn in $\overrightarrow{ba}$ must be the same as a vertex of $\overrightarrow{ab}$ at or before the wide right turn in $\overrightarrow{ab}$. In other words, after $\hat{x}$ steps, the $x$-axis must meet the $y$-axis such that a face receives both coordinates $(\hat{x}, 0)$ and $(0, y_{\hat{x}})$ for some value $y_{\hat{x}} \leq y_{ab}$. Furthermore, the arc from $(\hat{x} - 1, 0)$ to $(\hat{x}, 0)$ must be 1 clockwise position after the arc from $(0, y_{\hat{x}} - 1)$ to $(0, y_{\hat{x}})$.

One may say that in this case, the $x$-axis "wraps around" to meet the $y$-axis. The following definition is given to formalize this concept for any $y$-value scan, not just the $x$-axis. This provides a framework for Rule 4 that will also be used heavily in Section 4.2.3.

**Definition 4.2.7** (Wrap-around)**.** *A* wrap-around *occurs at* $(x, y)$ *if, for some* $y_x$*,* $(x, y) \equiv (0, y_x)$ *and the arc from* $(x - 1, y)$ *to* $(x, y)$ *is* 1 *clockwise position after the*

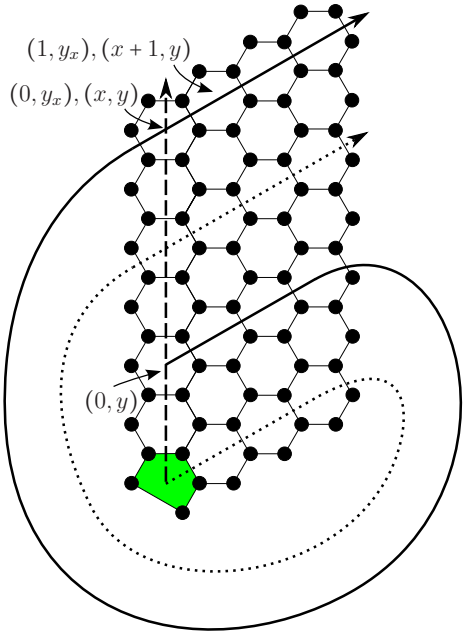


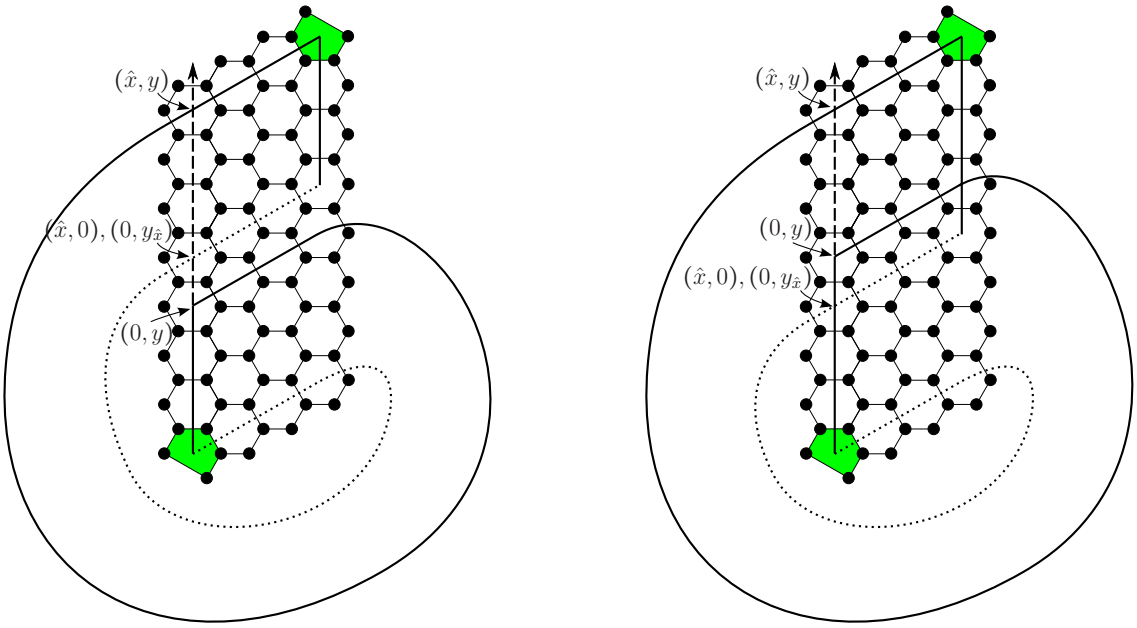

Figure 4.6: An illustration of a wrap-around at $(x, y) \equiv (0, y_x)$ during the $y$-value scan from $(0, y)$. The $x$-axis (dotted) and $y$-axis (dashed) are shown for reference.

*arc from* $(0, y_x - 1)$ *to* $(0, y_x)$.

In other words, the $x$-direction walk that left from $(0, y)$ "wrapped around" the fullerene's surface to meet the $y$-axis at some location $(0, y_x)$ such that if the $x$-direction walk continued (in the straight direction) it would follow the same arcs as the $x$-direction walk that leaves from $(0, y_x)$. Figure 4.6 illustrates a wrap-around.

Rule 4 is concerned only with when a wrap-around occurs such that the $x$-axis meets the $y$-axis at $(\hat{x}, 0) \equiv (0, y_{\hat{x}})$. Figure 4.7 shows examples in which the $x$-axis wraps around to meet the $y$-axis. In Figure 4.7(a), an $x$-direction walk from a $y$-value $y < y_{\hat{x}}$ discovers a proper clear field. However, in Figure 4.7(b), an $x$-direction walk from a $y$-value $y \geq y_{\hat{x}}$ discovers a non-proper clear field. In general, if a wrap-around occurs for the $x$-axis, then for a clear field with $y$-dimension greater than or equal to $y_{\hat{x}}$ to be a proper clear field, it can have an $x$-dimension of at most $\hat{x} - 1$. Otherwise (Figure 4.7(b)), the clear field would not be proper because the hexagon at $(\hat{x}, 0) \equiv (0, y_{\hat{x}})$ would be in both walks $\overrightarrow{ab}$ and $\overrightarrow{ba}$. This logic yields the following rule, which is checked during the $y$-value scan at $y = 0$ (when the $x$-axis is being walked):

**Rule 4:** If a wrap-around occurs at $(x, 0) \equiv (0, y_x)$, then when the algorithm begins a $y$-value scan at a $y$-value greater than or equal to $y_x$, the algorithm will ensure that $MaxX$ is at most $x - 1$.




(a) Proper clear field ($y < y_{\hat{x}}$)

(b) Non-proper clear field ($y \geq y_{\hat{x}}$)

Figure 4.7: Examples for Rule 4 where the $x$-axis (dotted) wraps around to meet the $y$-axis (dashed) and a later $x$-direction walk (solid) locates a pentagon. Solid vertical lines complete the clear field.

To implement Rule 4, the algorithm adds a field $MaxXFromXAxis$ to each array element of the $YAxis$ array and these fields are initialized to $\infty$ for each face (Line 7). When the algorithm visits a face at $(x, y)$, it checks if the location indicates a wrap-around by checking if the current face is on the $y$-axis, at say $(0, y_x)$, and if the the arc from $(x-1, y)$ to $(x, y)$ is 1 clockwise position after the arc from $(0, y_x - 1)$ to $(0, y_x)$ (Line 28). If the wrap-around occurs for the $x$-axis, when $y = 0$ (Line 29), then the value $MaxXFromXAxis$ is set to $x - 1$ for the $y$-axis location that corresponds with the current face (Line 30). Then when the algorithm begins the $y$-value scan at $(0, y_x)$, $MaxX$ is lowered to the $MaxXFromXAxis$ value that corresponds with $y_x$, if it is not already lower (Line 17).

The complexity of Algorithm 4.2 is the same as that of Algorithm 4.1. Algorithm 4.2 is just a restricted version of Algorithm 4.1. The additional checks take only constant time and the initialization of the arrays takes linear time per quintant. Therefore, the running time of Algorithm 4.2 is also in $O(n^2)$.

### 4.2.3 All Proper Clear Fields in $O(n)$ Time

Algorithm 4.2 finds all proper clear fields in $O(n^2)$ time and this section discusses modifications that improve the run-time to $O(n)$. The pseudo-code for the modified

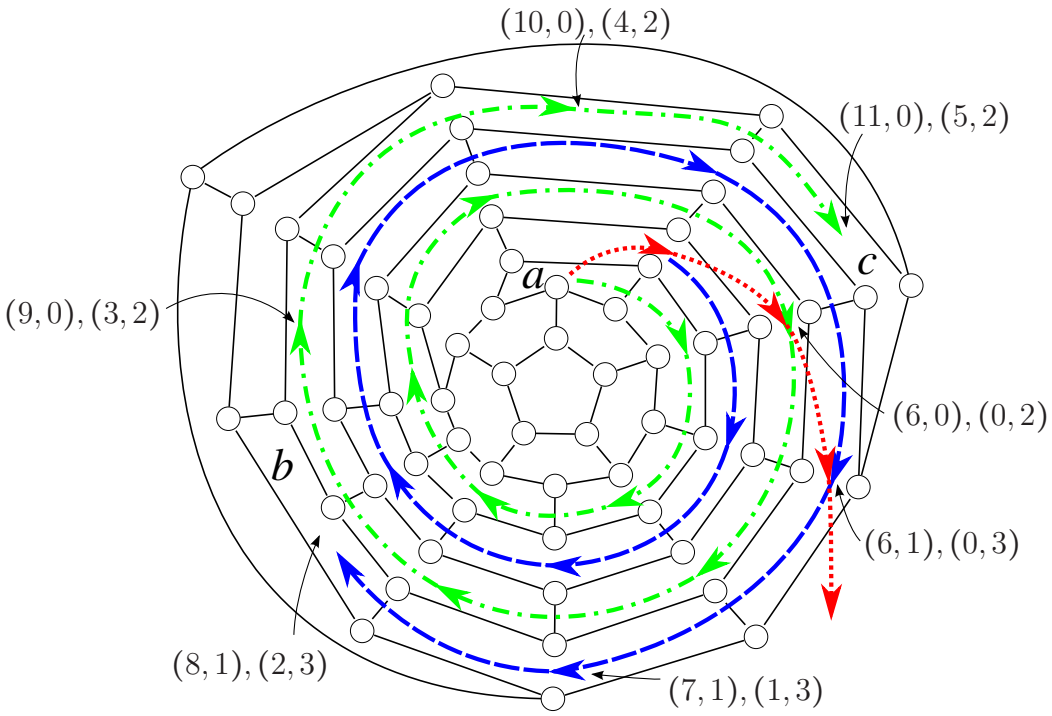



Figure 4.8: Fullerene $C_{60}$:3 in which the $y$-value scans from a quintant of $a$ at $y = 0$ (dash-dot) and $y = 1$ (dashed) indicate wrap-arounds at $(6, 0) \equiv (0, 2)$ and $(6, 1) \equiv (0, 3)$, respectively. The $y$-axis (dotted) is shown.

algorithm is given later in Algorithm 4.3.

To motivate the modifications, an example is given for which Algorithm 4.2 runs poorly. Figure 4.8 gives a small example, which is representative of a situation commonly found in larger fullerenes. Algorithm 4.2 would do the following when scanning the indicated quintant of $a$. At $y = 0$, the algorithm would visit the dual vertices from $(0, 0)$ to $(11, 0)$ (the dash-dotted walk), the last of which is labeled $c$ and corresponds to a pentagon, so a clear field with dimensions $(11, 0)$ would be recorded and $MaxX$ is set to 10. At $y = 1$, the algorithm visits $(0, 1)$ to $(8, 1)$ (the dashed walk) and discovers a proper clear field with $b$ with dimensions $(8, 1)$ and sets $MaxX = 7$. At $y = 2$, the algorithm visits $(0, 2)$ to $(5, 2)$ (re-walks the end of the dash-dotted walk) and discovers a proper clear field with $c$ with dimensions $(5, 2)$ and sets $MaxX = 4$. At $y = 3$, the algorithm visits $(0, 3)$ to $(2, 3)$ (re-walks the end of the dashed walk) and discovers a proper clear field with $b$ with dimensions $(2, 3)$ and sets $MaxX = 1$. Because the outer face $(0, 4)$ is a pentagon, the algorithm has finished scanning the current quintant.

In this example, the $y$-value scan that begins at $(0, 0)$ follows an arc from $(5, 0)$ to $(6, 0)$ to indicate a wrap-around at $(0, 2)$. This caused the six dual vertices at $(6, 0) \equiv$

$(0,2)$, $(7,0) \equiv (1,2)$, $(8,0) \equiv (2,2)$, $(9,0) \equiv (3,2)$, $(10,0) \equiv (4,2)$, and $(11,0) \equiv (5,2)$ to be visited twice, first when $y = 0$ and again when $y = 2$. Similarly, the $y$-value scan that begins at $(0,1)$ follows an arc from $(5,1)$ to $(6,1)$ that indicates a wrap-around at $(0,3)$. This caused the three dual vertices at $(6,1) \equiv (0,3)$, $(7,1) \equiv (1,3)$, and $(8,1) \equiv (2,3)$ to be visited twice, first when $y = 1$ and again when $y = 3$. This issue arises because the two clear fields between $a$ and $c$ that were discussed share the same quintant at $a$ and the same quintant at $c$. Likewise for the two clear fields between $a$ and $b$ that were discussed.

Recall that Theorem 3.2.8 gave an upper bound of $O(\lg n)$ on the number of proper clear fields that share a quintant of one pentagon and a quintant of another pentagon. Thus, in the general case, a dual vertex may be visited by Algorithm 4.2 $O(\lg n)$ times. The goal of this section is to provide modifications that reduce the number of times that any one dual vertex is visited to $O(1)$.

Before discussing the modifications in general, they are presented in the context of the example in Figure 4.8. The modifications result in an algorithm that behaves as follows. At $y = 0$, the algorithm visits $(0,0)$ to $(11,0)$ and detects and records the wrap-around at $(6,0) \equiv (0,2)$. Then when $(11,0)$ is reached, a clear field with dimensions $(11,0)$ is recorded and $MaxX$ is set to 10, as before. Additionally, because of the wrap-around at $(6,0) \equiv (0,2)$, the algorithm records that the $y$-value scan from $y = 2$ will result in the discovery of pentagon $c$ at $(5,2)$ (since 11−6 = 5). The algorithm increments $y$ to 1 and visits $(0,1)$ to $(8,1)$ and detects and records the wrap-around at $(6,1) \equiv (0,3)$. Then the clear field with dimensions $(8,1)$ is recorded and $MaxX = 7$. Additionally, the algorithm records that the $y$-value scan from $y = 3$ will result in the discovery of pentagon $b$ at $(2,3)$ (since $8 - 6 = 2$). The algorithm increments $y$ to 2 and recalls that the result of the $y$-value scan is already known: $(5,2)$ corresponds to pentagon $c$. Because $5 < MaxX$, the algorithm records the clear field $(5,2)$ and then sets $MaxX = 4$, all without actually visiting $(0,2)$ through $(5,2)$. At $y = 3$, the algorithm recalls that $(2,3)$ corresponds with pentagon $b$ and records the clear field because $2 < MaxX$ and then sets $MaxX = 1$. The output of the algorithm with these modifications is exactly the same, however each hexagon in this example was visited at most once instead of at most twice as before.

In general, modifications are introduced to detect wrap-arounds during $y$-value scans, remember the result of such walks, and use the results to avoid re-walking later $y$-value scans that would start at the wrap-around locations. To discuss the details of the modifications in general, some values are defined as shown in Figure 4.6. Suppose $(x, y)$ is the coordinate of the dual vertex that the algorithm is currently

```
34 Create arrays FaceVisits and YAxis of size n/2 + 2: hold info about face visits and
   y-axis;
35 foreach Pentagon fromPent ← 0 to 11 do   /* Pents numbered arbitrarily */
36 |  foreach Quintant do                    /* 5 quintants of current pentagon */
37 |  |  ScanQuintant ();
```

**Algorithm 4.3**: An $O(n)$ algorithm for discovering all proper clear fields of a fullerene.

```
   ScanQuintant () begin
38 |  foreach Face f ← 0 to n/2+2 do              /* Initialize array values */
39 |  |  FaceVisits[f].x ← null; FaceVisits[f].y ← null;
40 |  |  YAxis[f].y ← null; YAxis[f].MaxXFromXAxis ← ∞;
   |  |  YAxis[f].scanBegun ← false; YAxis[f].MaxXFromWA ← null;
   |  |  YAxis[f].toPentFromWA ← null;
41 |  MaxX ← ∞; MaxY ← 0;
42 |  y ← 1; currentFace ← face at (0, y) ;          /* Discover the y-axis */
43 |  while currentFace is not a pentagon AND YAxis[currentFace].y ≠ null do
44 |  |  MaxY ← MaxY + 1;
45 |  |  YAxis[currentFace].y ← y;
46 |  |  y ← y + 1; currentFace ← face at (0, y);
47 |  for y ← 0 to MaxY do                       /* Loop through y-axis faces */
48 |  |  currentFace ← face at (0, y);
49 |  |  FaceVisits[currentFace].x ← 0; FaceVisits[currentFace].y ← y;
50 |  |  MaxX ← min(MaxX, YAxis[currentFace].MaxXFromXAxis);
51 |  |  if YAxis[currentFace].scanBegun then
   |  |  |  /* result of the y-value scan is already known                  */
52 |  |  |  if YAxis[currentFace].toPentFromWA ≠ null and
   |  |  |  YAxis[currentFace].MaxXFromWA < MaxX then
   |  |  |  |  /* a pentagon is reached at this y-value                    */
53 |  |  |  |  Record clear field from fromPent to pentagon
   |  |  |  |  YAxis[currentFace].toPentFromWA with dimensions
   |  |  |  |  (YAxis[currentFace].MaxXFromWA + 1, y);
54 |  |  |  MaxX ← min(MaxX, YAxis[currentFace].MaxXFromWA);
55 |  |  else ScanYValue () ;                  /* Begin the y-value scan for y */
   end
```

visiting and a wrap-around occurs at $(x, y) \equiv (0, y_x)$. Then $(x + i, y) \equiv (i, y_x)$ for $i > 0$ where the faces at $(x, y)$ through $(x + i - 1, y)$ are hexagons.

The modifications include adding a boolean-value field *scanBegun* to each $YAxis$ array location and these are initially all *false* (Line 40). This value indicates whether or not the straight walk for the corresponding $y$-value scan was begun, either through the standard way (incrementing $y$) or due to the detection of a wrap-around at the corresponding location. Thus, when the algorithm gets a new $y$-value, it sets *scanBegun*

```
   ScanYValue () begin
56   Create queue WAQueue: records info for wrap-arounds of this y-value scan;
57   YAxis[currentFace].scanBegun ← true;
58   x ← 0;
59   while x ≤ MaxX or WAQueue not empty do /* Loop through x-direction */
60     x ← x + 1;
61     currentFace ← face at (x, y) ;        /* Visit next face in x-direction */
62     if currentFace is a pentagon then     /* Detect & record clear fields */
63       toPent ← currentFace's pentagon number (0 to 11);
64       if x ≤ MaxX then
65         if fromPent < toPent then Record clear field (x, y);
66         MaxX ← x − 1;
         /* Record the pentagon for each recorded wrap-around            */
67       while WAQueue is not empty do
68         Pop the pair (WAFace, WAx) from WAQueue;
69         YAxis[WAFace].MaxXFromWA ← x − WAx − 1;
70         if fromPent < toPent then YAxis[WAFace].toPentFromWA ← toPent;
71     else
         /* Detect if the y-value scan indicates a wrap-around here      */
72       if wrap-around at currentFace and ¬YAxis[currentFace].scanBegun then
73         if y = 0 then                     /* x-axis indicates a wrap-around */
74           YAxis[currentFace].MaxXFromXAxis ← x − 1;
75         Push the pair (currentFace, x) onto WAQueue;
76         YAxis[currentFace].scanBegun ← true;
         /* Detect if currentface was prev visited by this y-value scan  */
77       if FaceVisits[currentFace].y = y then
78         if x ≤ MaxX then MaxX ← x − 1;
           /* Remove and mark wrap-arounds that have a previous visit    */
79         while WAQueue is not empty do
80           Peek the pair (WAFace, WAx) from WAQueue;
81           if FaceVisits[currentFace].x − WAx ≥ 0 then  /* previous visit */
82             YAxis[WAFace].MaxXFromWA ← x − WAx − 1;
83             Pop the pair from WAQueue;
84           else break out of while loop;
         /* Remove oldest wrap-around if it reached MaxX                 */
85       if WAQueue is not empty then
86         Peek the pair (WAFace, WAx) from WAQueue;
87         if x − WAx = MaxX then
88           YAxis[WAFace].MaxXFromWA ← MaxX;
89           Pop the pair from WAQueue;
       /* Record that (x, y) was visited                                 */
90     FaceVisits[currentFace].x ← x; FaceVisits[currentFace].y ← y;
   end
```

to *true* (Line 57) at the start of the $y$-value scan. And, if during the $y$-value scan the algorithm detects a wrap-around at $(x, y) \equiv (0, y_x)$, then it sets the *scanBegun* value for $(0, y_x)$ to *true* (Line 76).

When the algorithm increments $y$ in preparation for the next $y$-value scan, it checks if the corresponding *scanBegun* value is *true* (Line 51). If it is *true*, then a wrap-around occurred at $(0, y)$ during a previous $y$-value scan and the $y$-value scan can be avoided (Lines 51-54), otherwise the $y$-value scan is executed (Line 55).

In order to avoid a $y$-value scan due to a wrap-around, the algorithm not only records that the wrap-around occurred (using *scanBegun*), but also the result of the straight walk. Two additional fields are added to each $YAxis$ array value to store this information: $MaxXFromWA$ and $toPentFromWA$. (The $WA$ in the variable names indicate that they record information about wrap-arounds.) Each of these values is initialized to *null* at the beginning of the quintant scan (Line 40). The meaning of the values of these two fields is apparent from their usage, which is described next.

The field $MaxXFromWA$ is used as follows. A non-*null* $MaxXFromWA$ value for some $y$-axis vertex $(0, y_x)$ indicates that a wrap-around occurred at $(0, y_x)$ during a $y$-value scan at a $y$-value less than $y_x$. In such a case, it is unnecessary to execute the $y$-value scan at $y_x$. However, had the $y$-value scan at $y_x$ been executed, it would have resulted in $Max$ being set to the smaller of $MaxX$ and $MaxXFromWA$ (Line 54) due to one of the four rules. The details of how $MaxXFromWA$ is determined are given later in this section.

The field $toPentFromWA$ is used as follows. If the *scanBegun* value at $(0, y_x)$ is *true*, then a wrap-around occurred at $(0, y_x)$ during a $y$-value scan at a $y$-value less than $y_x$ and that wrap-around reached pentagon number $toPentFromWA$ (0 to 11) at an $x$-distance of $MaxXFromWA + 1$. Therefore, if the algorithm is avoiding the $y$-value scan at $y_x$ (Line 51) it can check if $MaxXFromWA < MaxX$ (Line 52) and if *true*, then a clear field is recorded with pentagon $toPentFromWA$ with dimensions $(MaxXFromWA + 1, y_x)$.

In order to determine the values of $MaxXFromWA$ and $toPentFromWA$, a queue called $WAQueue$ is used to store information about each of the wrap-around locations because multiple wrap-arounds may occur during a $y$-value scan. The queue is initially empty (Line 56) at the beginning of the $y$-value scan. When the algorithm detects a wrap-around at $(x, y)$, a pair of values is added to $WAQueue$ (Line 75). This *value-pair* consists of $x$ and the unique face number (0 to $n/2 + 1$) of the face at $(x, y)$. Line 72 ensures that each $y$-axis location is represented in the queue at most

once by checking the *scanBegun* value for the location $(x, y)$. Because $WAQueue$ is a queue and the wrap-arounds are added as they are discovered, the value-pairs are added in order of increasing $x$-value, which is an important fact for some of the logic that uses the queue.

The implementation of Rules 1-4 change with the introduction of queue $WAQueue$ and values *scanBegun*, $MaxXFromWA$, and $toPentFromWA$. The modifications for each rule are described here and included are the details of how $MaxXFromWA$ and $toPentFromWA$ are determined.

**Rule 1** [$MaxX$ **limit**]**:** There are now cases in which the algorithm visits $(x, y)$ and $x > MaxX$. To see why, suppose a wrap-around was recorded during the current $y$-value scan and let $WAx$ be the $x$-value that was recorded by an arbitrarily chosen value-pair in $WAQueue$ such that $(0, y_{WAx}) \equiv (WAx, y)$ is the face represented by the value-pair. Therefore, the currently visited dual vertex, $(x, y)$, also has coordinate $(x - WAx, y_{WAx})$. If $x - WAx \leq MaxX$, then the algorithm still visits $(x, y)$, regardless of whether or not $x > MaxX$. This is done in order to determine the result of the $y$-value scan from $y_{WAx}$ that will later be skipped.

The code at Lines 85-89 implement this logic. After the visit to a face $(x, y)$, the algorithm checks the value-pair at the front of $WAQueue$ to see if $x - WAx = MaxX$. If this is the case, then the value-pair is popped from the queue and it is recorded that $MaxXFromWA = MaxX$ for the $y$-axis face at $(0, y_{WAx})$. At most one value-pair in $WAQueue$ can satisfy $x - WAx = MaxX$ and such a pair must be at the front of the queue because the value-pairs are added to $WAQueue$ in order of strictly increasing $WAx$ value.

This block of code ensures that if $WAQueue$ is not empty, then at least one value pair in $WAQueue$ satisfies $x - WAx \leq MaxX$. The $x$-loop condition on Line 59 allows the algorithm to have $x > MaxX$ if and only if $WAQueue$ is not empty.

**Rule 2** [**Pentagon found**]**:** Additional code is added in the case where the face at $(x, y)$ is a pentagon. A clear field with dimensions $(x, y)$ is only recorded if $x \leq MaxX$ (Line 64).

When the face at coordinate $(x, y)$ is a pentagon, the $MaxXFromWA$ and $toPentFromWA$ values must be set for each wrap-around that is represented by a value-pair in $WAQueue$. For an arbitrary value-pair in $WAQueue$ with its corresponding $WAx$ value, let $y_{WAx}$ be the value such that $(0, y_{WAx}) \equiv (WAx, y)$

is the vertex of the wrap-around represented by the value-pair. Because a pentagon exists at $(x, y)$, it also has coordinate $(x - WAx, y_{WAx})$. Thus, in Lines 67-70, each value pair in $WAQueue$ is popped from the queue, the $MaxXFromWA$ value for location $(0, y_{WAx})$ is set to $x - WAx - 1$, and the $toPentFromWA$ value for location $(0, y_{WAx})$ is set to the unique pentagon number ($toPent$ from 0 to 11) of the discovered pentagon if it has not yet been scanned (if $fromPent < toPent$, just as in the original Rule 2).

Note that the existence of a pentagon at $(x - WAx, y_{WAx})$ does not imply the existence of a proper clear field, so no clear field is recorded here. For a clear field to exist, the algorithm must have a $MaxX$ value of at least $x - WAx$ when the algorithm reaches the $y$-value of $y_{WAx}$.

**Rule 3 [Ensure $\overrightarrow{ab}$ does not contain repeated vertices]:**

**Rule 3a [Unique $y$-axis faces]:** No changes are made because wrap-arounds are not relevant.

**Rule 3b [Unique $y$-value scan faces]:** Additional code is added in the case where the face at $(x, y)$ was previously visited at some location $(\hat{x}, y)$ during the same $y$-value scan (so $\hat{x} < x$). Previously, such a situation would cause $MaxX$ to be changed to $x - 1$, but because it is now possible for $x > MaxX$ after a wrap-around occurred, the value $MaxX$ is only set to $x - 1$ if $x \leq MaxX$ (Line 78).

To understand what $(x, y) \equiv (\hat{x}, y)$ means for a wrap-around, consider an arbitrary value-pair in $WAQueue$ with its corresponding $WAx$ value. Let $y_{WAx}$ be the value such that $(0, y_{WAx}) \equiv (WAx, y)$ is the vertex of the wrap-around represented by the value-pair. Because $(x, y) \equiv (\hat{x}, y)$, the $y$-value scan from $y_{WAx}$ (that will be skipped due to the wrap-around at $(0, y_{WAx}) \equiv (WAx, y)$) would revisit the face at $(x - WAx, y_{WAx})$ after previously visiting it at $(\hat{x} - WAx, y_{WAx})$ if $\hat{x} - WAx \geq 0$.

The algorithm therefore determines which value-pairs in $WAQueue$ meet the condition of $\hat{x} - WAx \geq 0$. The $y$-axis vertices for such value-pairs will have their $MaxXFromWA$ value set to $x - WAx - 1$ and the value-pairs will be removed from the queue. To determine which of the value-pairs in $WAQueue$ satisfy the condition $\hat{x} - WAx \geq 0$, the algorithm exploits the fact that the value-pairs were added to $WAQueue$ in order of increasing $WAx$ value. The code block on Lines 79-84 handles this. The loop repeatedly looks at the value-pair at the front of the queue and if $\hat{x} - WAx \geq 0$

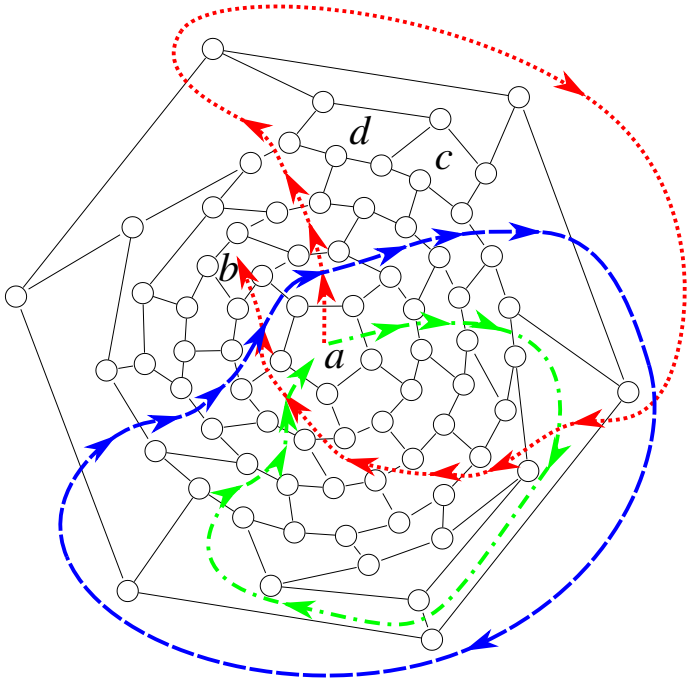



Figure 4.9: An example situation in which $x$ obtains its maximum value of $2 \cdot MaxX$, shown on a fullerene having $n = 76$ with the $x$-axis (dash-dot) and $y$-axis (dotted) of a quintant of $a$ along with the $y$-value scan from $y = 1$ (dashed)

then $MaxXFromWA$ is set to $x - WAx - 1$ and the value-pair is popped, otherwise the loop exits.

**Rule 4** [**$x$-axis wrap-around**]**:** The implementation of this rule is in two parts: first detecting where the $x$-axis has wrap-arounds and second to decrease $MaxX$ before a $y$-value scan begins. No changes are necessary to the first part since the $y$-value scan at $y = 0$ is always executed. For the second part, the code that implements this rule at each $y$-value is found on Line 50, which is executed regardless of whether or not a $y$-value scan is executed or skipped due to a wrap-around.

A simple version of how the algorithm handles a complex situation is illustrated in Figure 4.9. The scanning of the indicated quintant of $a$ begins by identifying the $y$-axis (here $MaxY = 11$ because pentagon $b$ is found at $(0, 12)$) and initializing the values $MaxXFromWA$ and $toPentFromWA$ to null and the value $scanBegun$ to *false* for each $YAxis$ array element. The value $MaxX$ is initialized to $\infty$.

The $y$-value scans begin with $y = 0$ and so $scanBegun$ is set to *true* for the dual vertex $(0, 0)$. $WAQueue$ is initially empty. For the $y$-value scan, the algorithm visits the dual vertices on the $x$-axis $(0, 0)$ through $(8, 0)$, the last of which indicates a wrap-

around at $(0, 10)$. This causes a value-pair describing $(8, 0) \equiv (0, 10)$ to be added to $WAQueue$ and the *scanBegun* value is set to *true* for the $y$-axis vertex $(0, 10)$. The algorithm continues the $y$-value scan and visits $(9, 0)$, which is pentagon $a$ and causes $MaxX$ to be set to 8. No clear field is recorded because $a$ cannot pair with itself to form a clear field. Furthermore, the value-pair is popped from $WAQueue$ and $MaxXFromWA$ is set to 1 (since $9 - WAx = 1$) for the corresponding $y$-axis location $(0, 10)$. The value $toPentFromWA$ is not set because $a$ cannot pair with itself to form a clear field.

When $y = 1$, the algorithm sets *scanBegun* to *true* for the dual vertex $(0, 1)$. The algorithm then begins the $y$-value scan with an empty $WAQueue$. It visits $(0, 1)$ through $(8, 1)$, the last of which indicates a wrap-around at $(0, 11)$. This causes a value-pair describing $(8, 1) \equiv (0, 11)$ to be added to $WAQueue$ and the *scanBegun* value is set to *true* for the $y$-axis vertex $(0, 11)$. Because $WAQueue$ is not empty, the algorithm continues allowing $x > MaxX$ and visits $(9, 1)$, which indicates a wrap-around at $(0, 1)$. This wrap-around is not recorded in $WAQueue$ because *scanBegun* is *true* for $(0, 1)$. The algorithm continues visiting $(10, 1) \equiv (1, 1)$ through $(16, 1) \equiv (7, 1)$, at which point it stops because the $WAx$ value in the value-pair in $WAQueue$ is 8 and $x - WAx = MaxX$. This causes the value-pair to be popped from $WAQueue$ and the $MaxXFromWA$ is set to 8 ($MaxX$) for the corresponding $y$-axis location $(0, 11)$.

The remainder of the scanning of the quintant of $a$ is summarized as follows. At $y = 2$, a clear field with $c$ is discovered and recorded with dimensions $(2, 2)$ and $MaxX$ is set to 1. At $y = 3$, a clear field with $d$ is discovered and recorded with dimensions $(1, 3)$ and $MaxX$ is set to 0. Nothing interesting happens for $y = 4$ through $y = 9$. When the algorithm reaches $y = 10$, the $y$-value scan is skipped because *scanBegun* is *true* and $MaxX$ remains 0 because it is already smaller than the corresponding $MaxXFromWA$ value. Likewise, at $y = 11$, the $y$-value scan is skipped and $MaxX$ remains the same.

There are six ways that a straight walk may pass through a vertex of degree six. An important thing to notice in this example is that during the $y$-value scan when $y = 1$ the dual vertices at $(0, 1)$ through $(7, 1)$ were visited a second time in the same direction. The following Lemma proves that this is the highest possible number of visits.

**Lemma 4.2.8.** *Each dual arc is traversed at most two times during the scanning of a quintant in Algorithm 4.3.*

*Proof.* The set of dual arcs is partitioned into three classes: those that are never

traversed by the algorithm during the scanning of a quintant, those that are traversed as part of the $y$-axis, and those that are traversed as part of a $y$-value scan. The arcs traversed as part of the $y$-axis are those that are followed as the algorithm visits the vertices at coordinates $(0,0)$ to $(0, MaxY+1)$, but not the arcs in the opposite direction. No arc traversed as part of the $y$-axis can be traversed during a $y$-value scan. This is because the straight direction of each $y$-value scan is uniquely defined, begins at a unique vertex of degree six, and does not pass through a vertex of degree five.

If an arc is traversed as part of the $y$-axis, then it is traversed at most twice: once when the $y$-axis is pre-walked to determine $MaxY$ and initialize the $YAxis$ array, and once when $y$ is incremented to prepare for each $y$-value scan.

Let $(u,v)$ be an arc that is traversed as part of a $y$-value scan. The algorithm's detection of wrap-arounds and the use of $WAQueue$ and *scanBegun* ensure that $(u,v)$ is traversed by at most one $y$-value scan. Therefore, if $(u,v)$ is traversed more than once during the quintant scan, then these traversals each correspond with the same $y$-value scan. Let $y$ be the $y$-value of the algorithm during these traversals. Furthermore, these multiple traversals occur due to a wrap-around at $(0,y)$, as in the situation illustrated by Figure 4.9.

For contradiction, suppose $(u,v)$ is traversed at least three times and let $0 < x_1 < x_2 < x_3$ be the $x$-values of the algorithm when it makes the first three visits to $v$. Thus, $(u,v)$ is the arc from coordinate $(x_1-1,y)$ to $(x_1,y)$ that is also the arc from $(x_2-1,y)$ to $(x_2,y)$ and from $(x_3-1,y)$ to $(x_3,y)$. Because vertices $(x_1,y) \equiv (x_2,y)$, vertices $(x_2-x_1,y) \equiv (0,y)$. When the algorithm visited vertex $(x_2-x_1,y)$, it visited a vertex that was previously visited with the same $y$-value but a smaller $x$-value. Rule 3 ensures that after this visit $MaxX$ is at most $x_2-x_1-1$. Despite a wrap-around at $(x_2-x_1,y) \equiv (0,y)$, no value pair is added to $WAQueue$ because *scanBegun* is *true* for the location $(0,y)$.

Any arc that is traversed to arrive at a vertex at coordinate $(\hat{x},y)$ for $\hat{x} > x_2-x_1$ must have been previously traversed to reach the vertex at $(\hat{x}-x_1,y)$. No value-pair can be added to $WAQueue$ due to re-traversing an arc because the first traversal set *scanBegun* to *true* if a wrap-around was detected (Line 76), which causes a value-pair to not be added at future times when the wrap-around is detected (Line 72). This means the largest $WAx$-value in a value-pair in $WAQueue$ is $x_2-x_1-1$, which will be removed from the queue due to $MaxX$ when $x-(x_2-x_1-1) = MaxX$ (Rule 1, Line 87). Because $MaxX \le x_2-x_1-1$, this implies $x \le 2(x_2-x_1-1)$. That is, the largest $x$-value that can occur during a $y$-value scan is $2(x_2-x_1-1)$. This contradicts

| Proper Clear Fields per | Min | Max |
|---|---|---|
| Fullerene | 28 | 90 |
| Pentagon | 3 | 20 |
| Quintant | 0 | 9 |
| Pentagon-Pentagon Pair | 0 | 6 |
| Quintant-Pentagon Pair | 0 | 4 |
| Quintant-Quintant Pair | 0 | 3 |

Table 4.1: Number of proper clear fields for all of the fullerenes with 120 or fewer vertices.

the fact that $(u, v)$ was traversed three times because the vertex equivalence $(x_1, y) \equiv (x_2, y) \equiv (x_3, y)$ implies $x_3 - x_2 = x_2 - x_1$, so $x_3 = 2x_2 - x_1 > 2(x_2 - x_1 - 1)$. □

**Theorem 4.2.9.** *The time complexity of Algorithm 4.3 is in* $O(n)$.

*Proof.* As before, the time complexity proof analyzes the time complexity of scanning any one of the $O(1)$ quintants. Lemma 4.2.8 showed that during the scanning of a quintant, each of the $O(n)$ dual arcs is traversed $O(1)$ times. There are six incoming arcs for each dual vertex, so the number of times each dual vertex is visited is $O(1)$.

The amount of work done at a visit to a dual vertex is not constant, however the only non-constant operations involve popping value-pairs from *WAQueue*. Each value-pair corresponds with exactly one of the $O(n)$ $y$-values and may be pushed onto *WAQueue* at most once during the scanning of each quintant due to the use of *scanBegun*. Therefore, over all the visits to dual vertices for a particular quintant, at most $O(n)$ values are popped, with $O(1)$ time for each pop.

Initialization before scanning a quintant takes $O(n)$ time and before a $y$-value scan takes $O(1)$ time. This gives $O(n)$ overall. □

Algorithm 4.3 detects all proper clear fields, the number of which is $O(\lg n)$ from Theorem 3.2.8. It was run on all of the fullerenes with 120 or fewer vertices. Some of the findings are summarized in Table 4.1.

## 4.3 Phase 2: Detect Pairs of Proper Clear Fields that Share Dual Vertices

As described in Section 4.1, the second phase of the algorithm determines which pairs of proper clear fields have a dual vertex in common. Such clear fields cannot be together in an optimal pairing.

A straightforward way to detect which pairs of clear fields have dual vertices in common is to loop through the list of clear fields and for each clear field visit its

dual vertices, leaving a token at each dual vertex. Before leaving a token at a dual vertex, the list of tokens previously left at the vertex is read and the clear fields they represent share the current dual vertex with the current clear field. The desired output is generated by storing a record, initialized to false, for each pair of clear fields and marking it as true when a common vertex is detected. Such an algorithm on $C$ clear fields would visit $O(n)$ dual vertices for each of the $C$ clear fields and would take $O(C)$ time at each dual vertex inspecting the previous tokens. This gives $O(nC^2)$ time overall, which is $O(n \lg^2 n)$ because $C \in O(\lg n)$ by Theorem 3.2.8. An asymptotically faster algorithm is preferred so that the entire independent set algorithm does not take $O(n \lg^2 n)$ time due to this computation.

A simple observation enables a faster algorithm. There are sixty *axes* on the fullerene that correspond with the $x$- and $y$-axes of the sixty quintants. Each axis is a straight dual walk that begins at one of the sixty arcs for the twelve vertices of degree five, and continues as far as possible such that each vertex of the axis is unique and the only vertex of degree five in an axis is the start vertex. Observe that each dual vertex of a clear field is a member of at least one of the sixty axes because these sixty axes form the $x$- and $y$-axes of the quintants. With a few tricks, it is possible to first create a mapping between the clear fields and the axes they use and then visit the dual vertices on the axes, analyze and leave tokens, and determine where two axes share a dual vertex and thereby determine which pairs of clear fields share a dual vertex. This approach will be asymptotically faster because there are $O(1)$ axes, $O(n)$ dual vertices on each, and $O(1)$ tokens to analyze at each vertex. This will give $O(n)$ time overall.

The details of such an approach are discussed here and pseudo-code is given in Algorithm 4.4. The sixty axes are numbered $A_0$ through $A_{59}$ arbitrarily. For each axis $A_i$, an array $CFs_i$ is created (Line 4) that references the clear fields that use $A_i$ as either the $x$- or $y$-axis of either of the two quintants used by the clear field. In this manner, the clear fields in $CFs_i$ are all of the clear fields that use either the quintant for which $A_i$ acts as the $x$-axis or the clockwise neighboring quintant for which $A_i$ acts as the $y$-axis. The clear fields are listed in array $CFs_i$ in increasing order by their *sort-value*, which is their $x$ or $y$-dimension, depending on if they use $A_i$ as an $x$- or $y$-axis, respectively.

The algorithm loops through the sixty axes $A_0$ to $A_{59}$ one at a time (Line 3), visiting the dual vertices on the axis in order beginning at the start vertex of degree five, for a maximum distance equal to the largest sort-value in $CFs_i$ (Line 8). At each vertex distance $d$ along the axis, an associated value $k$ is maintained (Lines 7

```
1  Create a C × C array initialized to false to hold the output;
2  Create empty token list at each dual vertex;
3  foreach Axis A_i (i ← 0 to 59) do
4      Create persistent array CFs_i of clear fields that use A_i as an x or y-axis, sorted by
       increasing x or y-value (sortValue) as appropriate;
5      Create persistent array Intersections_i of size length(CFs_i) with values initialized
       to ∞;
6      if length(CFs_i) = 0 then skip to the next axis;
7      k ← 0 ; /* k is the smallest index of a CF in CFs_i with sort-value ≥ d
       */
       /* Walk along A_i as far as the largest associated clear field      */
8      for d ← 0 to CFs_i[length(CFs_i) − 1].sortValue do
9          Move to the dual vertex distance d along axis A_i;
10         while CFs_i[k].sortValue < d do k ← k + 1 ;   /* Maintain the condition on
           k */
11         foreach token t in the token list at current dual vertex do
               /* A_i's CFs with index ≥ k and t.i's CFs with index ≥ t.k share the
                  current dual vertex, so remember the smallest k seen for t.k */
12             if Intersections_{t.i}[t.k] > k then Intersections_{t.i}[t.k] ← k;
13         Add a token t to list at current dual vertex with t.i ← i and t.k ← k;
14     Intersections_i[0] ← 0 ; /* All CFs in CFs_i share a dual vertex with each
       other */
15     for j ← 0 to i do            /* Consider each previously-walked axis A_j */
16         MinIndex ← ∞;
17         for g ← 0 to length(CFs_j) − 1 do    /* Consider each clear field CFs_j[g]
           */
18             MinIndex ← min(MinIndex, Intersections_j[g]);
               /* Each CF in CFs_i with index ≥ MinIndex shares a dual vertex
                  with each CF in CFs_j with index ≥ g.  In particular, CF
                  CFs_j[g] shares a dual vertex with each CFs_i[h] such that
                  h ≥ MinIndex                                              */
19             for h ← MinIndex to length(CFs_i) − 1 do
20                 Mark CFs_j[g] and CFs_i[h] as sharing a dual vertex (set true in output);
21             Intersections_j[g] ← ∞ ;                /* Reset for next i iteration */
```

**Algorithm 4.4**: An $O(n)$ algorithm for detecting the pairs of clear fields that have one or more dual vertices in common.

and 10), which is the smallest index of a clear field in $CFs_i$ that has a sort-value of at least $d$. Thus, only the clear fields that have index $k$ or larger use all of the first $d$ vertices of the axis. A token $t$ is left at the vertex $d$ steps along the axis. The token records the values $i$ and $k$ as $t.i$ and $t.k$ (Line 13), which indicates that the clear fields in $CFs_{t.i}$ with index $t.k$ or greater all include the vertex.

When the algorithm visits a dual vertex but before a token is added (Line 13), it analyzes the tokens that were left at the vertex by previous axes (Line 11). Let $t$ be a token that was left. Token $t$ indicates that this vertex is on axis $A_{t.i}$ and that the clear fields of $CFs_{t.i}$ with index $t.k$ or greater all contain the current vertex. Therefore, every clear field in $CFs_i$ with index $k$ or greater shares the current vertex with every clear field in $CFs_{t.i}$ with index $t.k$ or greater. A series of arrays is used to record this information. From Line 5, an array $Intersections_j$ exists for every value $0 \leq j \leq i$ such that $Intersections_j$ has the same length as $CFs_j$ and each value in $Intersections_j$ is set to $\infty$ before any vertices of $A_i$ were visited (this property is also maintained by Line 21). These arrays are used to store information such that $Intersections_j[g] = k$ (for $0 \leq g < length(CFs_j)$) means that every clear field in $CFs_i$ ($A_i$ being the current axis) with index $k$ or greater shares a dual vertex with every clear field in $CFs_j$ with index $g$ or greater. In Line 12, the algorithm records the information indicated by token $t$ by setting $Intersections_{t.i}[t.k]$ to $k$ if $Intersections_{t.i}[t.k] > k$ because if $Intersections_{t.i}[t.k] \leq k$ then it is already implied that these clear fields share a different dual vertex. Finally, after all vertices of $A_i$ have been visited (Line 14), the algorithm sets $Intersections_i[0]$ to 0 to indicate that every clear field in $CFs_i$ shares a dual vertex with every clear field in $CFs_i$.

The output of Algorithm 4.4 is an $C \times C$ array (or some other suitable structure) of boolean values, one for each pair of clear fields. These values are initialized to *false* (Line 1) and changed to *true* if the pair of clear fields shares a dual vertex. After the algorithm has finished visiting the dual vertices of $A_i$, it records in the output which pairs of a clear field in $CFs_i$ and a clear field in $CFs_j$ for $0 \leq j \leq i$ share a dual vertex of $A_i$. The algorithm loops through each choice of $j$ from 0 to $i$ (Line 15). To determine which pairs of a clear field in $CFs_i$ and a clear field in $CFs_j$ share a vertex, the algorithm loops through each choice of $g$ from 0 to $length(CFs_j) - 1$ (Line 17). As it loops through the $g$ values, it sets a variable $MinIndex$ to be the smallest of $\{Intersections_j[0], Intersections_j[1], \ldots, Intersections_j[g]\}$ (Line 18). Thus, each clear field in $CFs_i$ with index $MinIndex$ or greater shares a dual vertex with each clear field in $CFs_j$ with index $g$ or greater. In particular, $CFs_j[g]$ shares a dual vertex with each clear field in $CFs_i$ with index $MinIndex$ or greater. To record

this information, the algorithm loops through each choice of $h$ from $MinIndex$ to $length(CFs_i) - 1$ (Line 19) and records in the output array that clear fields $CFs_j[g]$ and $CFs_i[h]$ share a vertex by marking the appropriate value(s) to *true*. Finally, the algorithm resets $Intersections_j[g]$ to $\infty$ for each choice of $j$ and $g$ in preparation for processing axis $A_{i+1}$ (Line 21).

The time complexity analysis of Algorithm 4.4 is straightforward. The $t$ loop at Line 11 looks at at most 59 tokens each time it runs, with each iteration taking $O(1)$ time. Thus, each iteration of the $d$ loop at Line 8 takes $O(1)$ time. The $d$ loop makes $O(n)$ iterations each time it is encountered because an axis contains each dual vertex at most once. The $h$ loop at Line 19 takes $O(1)$ time per iteration, with $O(C)$ iterations each time the loop is encountered, where $C$ is the number of proper clear fields. This means each iteration of the $g$ loop at Line 17 takes $O(C)$ time, with $O(C)$ iterations each time it is encountered. Thus, each iteration of the $j$ loop at Line 15 takes $O(C^2)$ time, with $O(1)$ iterations each time it is encountered. Creating each $CFs_i$ array can be done in $O(C \lg C)$ time to locate and sort the clear fields for $A_i$. Each $Intersections_i$ array can be created and initialized in $O(C)$ time. Therefore, the time complexity of each iteration of the $i$ loop at Line 3 is $O(n + C^2)$. Because $C \in O(\lg n)$ and there are $O(1)$ iterations of the $i$ loop, the $i$ loop runs in time $O(n)$ overall. Creating the output array takes $O(C^2)$ time and creating the empty token lists takes $O(n)$ time, for $O(n)$ overall.

## 4.4 Phase 3: Find an Optimal Pairing

The first two phases have provided the information necessary for this phase to consider each selection of six proper clear fields that have no dual vertices in common as outlined in Section 4.1. Recall that such a selection must pair the twelve vertices of degree five. All possible sets of six proper clear fields with no dual vertex in common can be enumerated easily by using the output from Phases 1 and 2.

When a set of six clear fields is selected, the two possible ways to color the clear fields are determined as outlined in Section 4.1. For each of these two colorings, the penalty is computed. A set with an applied coloring having the minimum penalty over all choices of a set and a coloring is remembered as the optimal pairing, which is the output of this phase.

During a search for an optimal pairing, a practical speed increase may be obtained using the following method to reduce the search space. Recall that the penalty of a clear field $\square ab$ is either $2x_{ab} + y_{ab}$ or $x_{ab} + 2y_{ab}$. Thus, the minimum possible penalty for $\square ab$ is $2\min\{x_{ab}, y_{ab}\} + \max\{x_{ab}, y_{ab}\}$. As the six clear fields are being selected

one at a time, a sum of the minimum possible penalty may be quickly computed before the colors of the clear fields are determined. If this sum is ever greater than or equal to the penalty of the best (minimum penalty) complete solution seen so far, then the current subset of clear fields cannot be together in an optimal pairing.

There are several parts to Phase 3 of the algorithm that together find an optimal pairing. Section 4.4.1 gives a method to determine the two colorings for a selection of six clear fields and the penalties associated with them. To improve the running time, this part relies on some information output by an algorithm described by Section 4.4.2. The methods are discussed in a sequence to aid understanding and not the order of execution. Therefore, Section 4.4.3 ties the parts together by describing the order of execution.

### 4.4.1 Determining the Two Colorings of a Set of Six Clear Fields

For each selection of six clear fields that pair the pentagons, the two possible colorings are determined in order to compute the penalties. Assume that the twelve pentagons/dual vertices of degree five are uniquely and arbitrarily labeled with numbers 0 through 11 with $num(a)$ representing the number of vertex $a$. The approach is to first assume that pentagon 0 (chosen arbitrarily) is white, which implies that the clear field that contains the dual vertex corresponding to pentagon 0 is white. Using this information, the colors of the remaining five clear fields are determined. The other coloring of the set of six clear fields is obtained by switching the color of all of the clear fields.

To begin, a *coloring tree* is constructed, which is a connected directed graph with twelve vertices and eleven arcs. The vertices represent the pentagons and the arcs represent clear fields. These eleven arcs form a spanning tree of the twelve vertices such that a walk exists from the vertex representing pentagon 0 to every other vertex. The coloring tree is constructed by choosing eleven clear fields (from those found in Phase 1) using a breadth-first search from the dual vertex representing pentagon 0. It is important to note that the choice of a coloring tree is independent of any selection of six clear fields for which the color is being determined.

The coloring tree can be used to determine the colors of the twelve pentagons and thus color the six selected clear fields. The details follow, but the general approach is to use the known color of pentagon 0 (white) to determine the color of each of its neighbors in the coloring tree, then repeatedly determine the color of each of their neighbors until the color of each pentagon is known. Given a selection of six clear fields and a coloring tree, the procedure presented here determines the colors of the

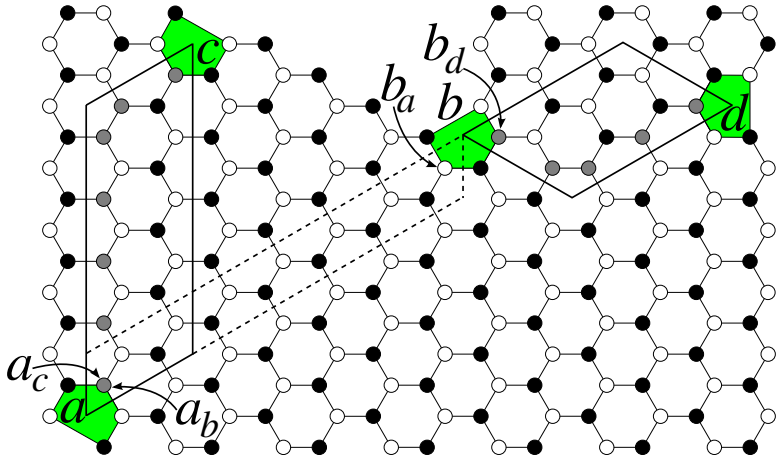



Figure 4.10: Illustration of how the color of clear field $\angle bd$ is determined from the color of $\angle ac$ by using $\angle ab$.

clear fields in constant time.

Figure 4.10 illustrates how the color of one pentagon may be determined using the known color of another pentagon and an arc of the coloring tree. It is assumed that the color of the pentagon at dual vertex $a$ is known, the color of the pentagon at $b$ is to be determined, and an arc from $a$ to $b$ exists in the coloring tree that corresponds to clear field $\angle ab$. Furthermore, let $\angle ac$ and $\angle bd$ be two of the six clear fields in the selection. This means the pentagon at $a$ is paired with the pentagon at $c$ and $b$ is paired with $d$. It may be the case (not pictured) that $c \equiv b$, in which case $a \equiv d$ and hence $\angle ac \equiv \angle ab \equiv \angle bd$, meaning the coloring tree clear field $\angle ab$ is one of the six clear fields selected.

In the last case, it is very simple to determine the color of the pentagon at $b$ from the color of the pentagon at $a$:

**Special Case** [$\angle ac \equiv \angle ab \equiv \angle bd$]**:** When $\angle ab$ is one of the six selected clear fields, the color of the pentagon at $b$ is the same as the color of the pentagon at $a$ due to Theorem 3.2.3(iii).

For the remainder of the discussion, it can be assumed that $\angle ac \not\equiv \angle ab \not\equiv \angle bd$. In other words, assume $c \not\equiv b$ and $d \not\equiv a$.

To aid the discussion, four primal vertices are labeled $a_b$, $b_a$, $a_c$, and $b_d$ such that vertex $a_b$ is the vertex on the pentagon at $a$ that is between the $x$- and $y$-arcs that define the quintant of $a$ used by $\angle ab$ and $b_a$, $a_c$, and $b_d$ are defined similarly. These vertices are labeled in Figure 4.10. Note that these four vertices are not necessarily unique. Two additional vertices are also labeled: $a'_b$ and $b'_a$ such that $a'_b$ is the vertex

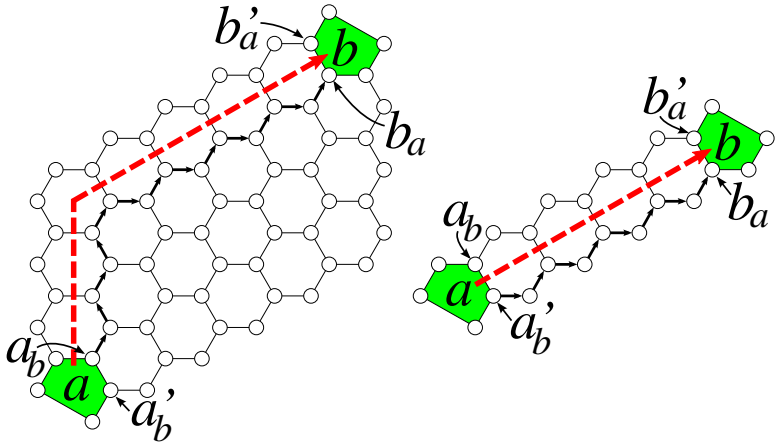



Figure 4.11: The inside walk (**bold** primal edges) of $\square ab$ from pentagon $a$ to pentagon $b$ when $y_{ab} > 0$ (left) and when $y_{ab} = 0$ (right). Its relationship to $\overrightarrow{ab}$ (**bold**, dashed) is also shown.

after $a_b$ in clockwise order around the pentagon at $a$ and $b'_a$ is similarly defined using $b_a$. These vertices are labeled in Figure 4.11.

In order to construct a coloring of the primal vertices, a pairing path is determined for each of the six selected clear fields. From Theorem 3.2.3(i), there can be many choices of such a pairing path for a clear field. For consistency, the rule followed by this algorithm is that for a selected clear field $\square ac$, the pairing path will use the dual edges of walk $\overrightarrow{ac}$ if $num(a) < num(c)$ and walk $\overrightarrow{ca}$ if $num(c) < num(a)$. The fullerene is colored as follows. For each of the six pairing paths, one primal vertex incident with each primal edge crossed by a pairing path edge is colored gray. The uncolored primal vertices create a bipartite graph. Color the four uncolored vertices on pentagon 0 such that the pentagon is white. The remaining uncolored primal vertices are then colored either white or black such that white and black are independent sets.

Such a coloring of the primal vertices is not actually constructed in this phase of the algorithm because it would take $\Omega(n)$ time for each choice of six clear fields. Instead the coloring is constructed only for the optimal pairing in Phase 4. However, important properties of the coloring can be determined without actually coloring the fullerene and these properties are used by this phase of the algorithm. The first important property is the choice of the pairing path for each clear field. The second important property is that the choice of the gray primal vertex incident with an edge crossed by the pairing path is chosen arbitrarily. Thus, the vertices can be assumed to be colored such that $a_c$ and $b_d$ are gray.

The *inside walk* $I$ is a primal walk shown in Figure 4.11 that starts at a vertex

on the pentagon at $a$ and ends at a vertex on the pentagon at $b$ and only uses primal arcs that are on the right side of $\overrightarrow{ab}$. If $y_{ab} > 0$, then $I$ starts at $a_b$ and ends at $b_a$ and has length $2(y_{ab} - 1) + 2(x_{ab} - 1) + 1$ (always odd). If $y_{ab} = 0$, then $I$ starts at $a'_b$ and ends at $b_a$ and has length $2(x_{ab} - 1)$ (always even).

The overall strategy is to apply Theorem 3.2.1 to a primal walk $P$ of length $k$ that starts at a non-gray vertex on the pentagon at $a$ and ends at a non-gray vertex on the pentagon at $b$. Eight cases are considered depending on the dimensions of $\square ab$, the relationship between the quintant of $a$ used by $\square ab$ and the quintant of $a$ used by $\square ac$, and the relationship between the quintant of $b$ used by $\square ab$ and the quintant of $b$ used by $\square bd$. Specifically, the cases are broken down based on whether or not $y_{ab} > 0$, whether or not $a_b \equiv a_c$, and whether or not $b_a \equiv b_d$. Examples of these cases are shown in Figures 4.12 and 4.13. In each figure, the edges of $P$ are bold.

Walk $P$ is constructed by using $I$ and, if necessary for the particular case, the arc between $a_b$ and $a'_b$ and/or the arc between $b_a$ and $b'_a$ are added to the beginning and/or end, respectively. Because $a_c$ is assumed to be gray, the start vertex of $P$ is $a_b$ unless $a_b \equiv a_c$ in which case the start vertex of $P$ is $a'_b$. Because $b_d$ is assumed to be gray, the end vertex of $P$ is $b_a$ unless $b_a \equiv b_d$ in which case the end vertex of $P$ is $b'_a$. The length $k$ of $P$ is at least the length of the inside walk and at most the length of the inside walk plus two. The details of the construction of $P$ are given when each of the eight cases are described in detail below.

The start vertex $P_0$ (either $a_b$ or $a'_b$) and the end vertex $P_k$ (either $b_a$ or $b'_a$) are colored either black or white. The color of $P_0$ can be easily determined from the color of the pentagon at $a$, which is known, by using the number of clockwise positions from $a_c$ to $P_0$. If $P_0$ is 1 or 3 clockwise position(s) after $a_c$, then $P_0$ is the same color as the pentagon at $a$, otherwise $P_0$ is the opposite color of the pentagon at $a$. Once the color of $P_k$ is known, the color of the pentagon at $b$ can be found by using the number of clockwise positions from $b_d$ to $P_k$. If $P_k$ is 1 or 3 clockwise position(s) after $b_d$, then the pentagon at $b$ is the same color as $P_k$, otherwise the pentagon at $b$ is the opposite color of $P_k$.

As in the statement of Theorem 3.2.1, let $\ell$ be the number of edges of $P$ that are white or black, which is the number of times an edge of $P$ is crossed by a dual edge of one of the six pairing paths. Let $Odd_P$ be a boolean value that is *true* if $\ell \equiv 1 \pmod 2$ and *false* otherwise. The color of $P_k$ can be determined by applying Theorem 3.2.1 to the knowledge of the color of $P_0$ and the values $k$, $Odd_P$. That is, if both $k$ is even and $Odd_P$ is *false* or both $k$ is odd and $Odd_P$ is *true* then $P_0$ and $P_k$ have the same color, otherwise they have different colors.

Similarly, let $Odd_I$ be *true* if the number of times the inside walk is crossed by the six pairing paths is odd and *false* otherwise. At this time, it is assumed that an oracle exists to provide the value $Odd_I$, but an algorithm to compute $Odd_I$ is given in Section 4.4.2. The value $Odd_P$ may only differ from $Odd_I$ if $P$ contains the edge between $a_b$ and $a'_b$ and/or $P$ contains the edge between $b_a$ and $b'_a$. The details of obtaining $Odd_P$ from $Odd_I$ are given when each of the eight cases are described in detail below because $P$ is constructed differently in each case.

The analysis of the eight cases is now given. Examples of the cases are shown in Figures 4.12 and 4.13.

**Case 1** $[y_{ab} > 0]$**:** Recall that the inside walk begins at $a_b$, ends at $b_a$, and has odd length.

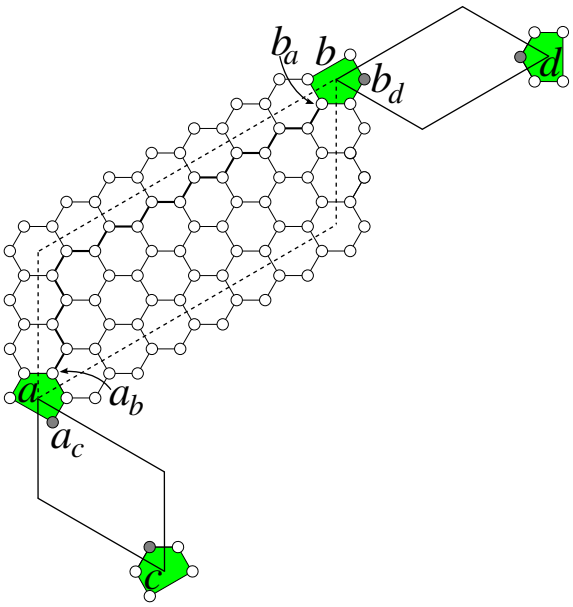
**Case 1.1** $[a_b \not\equiv a_c]$**:** Walk $P$ begins at $a_b$.

**Case 1.1.1** $[b_a \not\equiv b_d]$ **(Figure 4.12(a)):** Let $P$ be $I$. Thus, $P$ has odd length and $Odd_P = Odd_I$. From the application of Theorem 3.2.1 to $P$, $b_a$ has the same color as $a_b$ if $Odd_P$ is *true* and the opposite color otherwise.

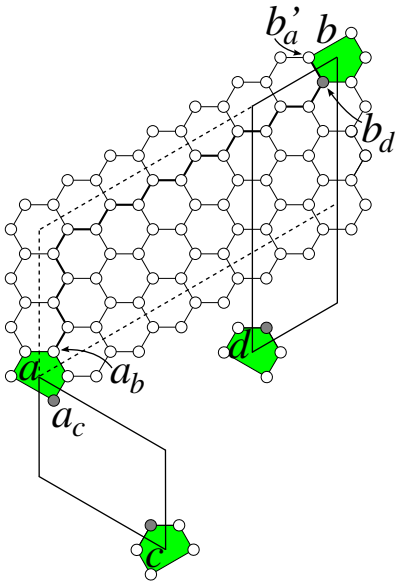
**Case 1.1.2** $[b_a \equiv b_d]$ **(Figure 4.12(b)):** Let $P$ be $I$, $(b_a, b'_a)$, $b'_a$. Thus, $P$ has even length. To know $Odd_P$, it is determined whether or not any of the six pairing paths cross arc $(b_a, b'_a)$. Because this arc is on the pentagon at $b$, only the pairing path that pairs $b$ and $d$ can cross the arc. Recall that the pairing path from $b$ to $d$ follows $\overrightarrow{bd}$ if $num(b) < num(d)$ and follows $\overrightarrow{db}$ if $num(d) < num(b)$. If $\overrightarrow{bd}$ is used, then it crosses $(b_a, b'_a)$ if and only if $y_{bd} = 0$ and if $\overrightarrow{db}$ is used, then it must cross $(b_a, b'_a)$. Therefore, if $num(d) < num(b)$ or both $num(b) < num(d)$ and $y_{bd} = 0$, then $Odd_P = \neg Odd_I$, otherwise $Odd_P = Odd_I$. From the application of Theorem 3.2.1 to $P$, $b'_a$ has the opposite color of $a_b$ if $Odd_P$ is *true* and the same color otherwise.

**Case 1.2** $[a_b \equiv a_c]$**:** Walk $P$ begins at $a'_b$ and first uses arc $(a'_b, a_b)$. Similar to the logic in Case 1.1.2, the only pairing path that can cross arc $(a'_b, a_b)$ is the one that pairs $a$ with $c$. If $num(c) < num(a)$ or both $num(a) < num(c)$ and $y_{ac} = 0$, then $(a'_b, a_b)$ is crossed by a pairing path. This information will be used to find $Odd_P$ in the following two sub-cases.

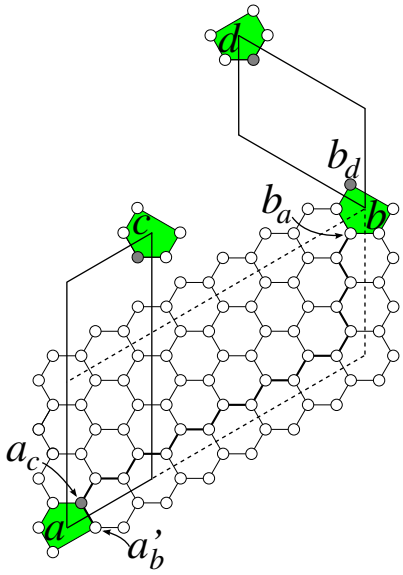
**Case 1.2.1** $[b_a \not\equiv b_d]$ **(Figure 4.12(c)):** Let $P$ be $a'_b$, $(a'_b, a_b)$, $I$. Thus, $P$ has even length. If $num(c) < num(a)$ or both $num(a) < num(c)$



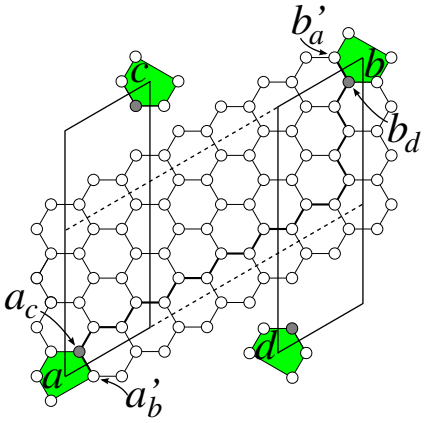

(a) Case 1.1.1: $y_{ab} > 0$, $a_b \not\equiv a_c$, $b_a \not\equiv b_d$




(b) Case 1.1.2: $y_{ab} > 0$, $a_b \not\equiv a_c$, $b_a \equiv b_d$




(c) Case 1.2.1: $y_{ab} > 0$, $a_b \equiv a_c$, $b_a \not\equiv b_d$

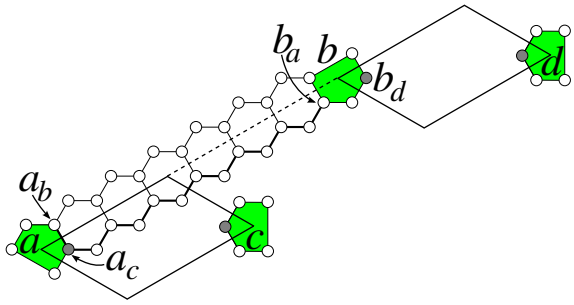



(d) Case 1.2.2: $y_{ab} > 0$, $a_b \equiv a_c$, $b_a \equiv b_d$

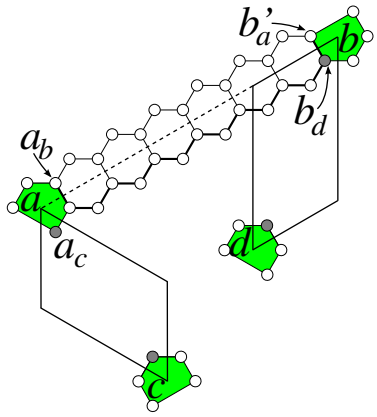
Figure 4.12: Determining the color of the pentagon at $b$ from the color of the pentagon at $a$ using $\square ab$ with $y_{ab} > 0$

and $y_{ac} = 0$, then $Odd_P = \neg Odd_I$, otherwise $Odd_P = Odd_I$. From the application of Theorem 3.2.1 to $P$, $b_a$ has the opposite color of $a'_b$ if $Odd_P$ is *true* and the same color otherwise.

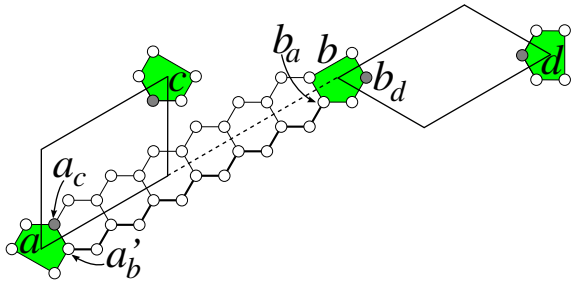
**Case 1.2.2** [$b_a \equiv b_d$] **(Figure 4.12(d)):** Let $P$ be $a'_b$, $(a'_b, a_b)$, $I$, $(b_a, b'_a)$,



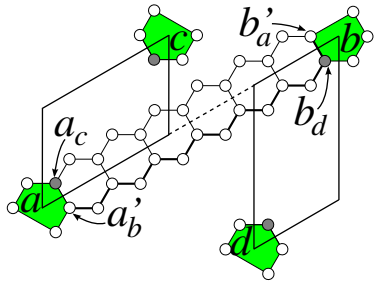

(a) Case 2.1.1: $y_{ab} = 0$, $a_b \not\equiv a_c$, $b_a \not\equiv b_d$




(b) Case 2.1.2: $y_{ab} = 0$, $a_b \not\equiv a_c$, $b_a \equiv b_d$




(c) Case 2.2.1: $y_{ab} = 0$, $a_b \equiv a_c$, $b_a \not\equiv b_d$




(d) Case 2.2.2: $y_{ab} = 0$, $a_b \equiv a_c$, $b_a \equiv b_d$

Figure 4.13: Determining the color of the pentagon at $b$ from the color of the pentagon at $a$ using $\square ab$ with $y_{ab} = 0$

$b'_a$. Thus, $P$ has odd length. As in Case 1.1.2, if $num(d) < num(b)$ or both $num(b) < num(d)$ and $y_{bd} = 0$, then arc $(b_a, b'_a)$ is crossed by a pairing path. Then $Odd_P$ is determined from $Odd_I$ in the straightforward way by negating $Odd_I$ for each of the arcs $(a'_b, a_b)$ and $(b_a, b'_a)$ that are crossed by a pairing path. From the application of Theorem 3.2.1 to $P$, $b'_a$ has the same color as $a'_b$ if $Odd_P$ is *true* and the opposite color otherwise.

**Case 2** $[y_{ab} = 0]$**:** Recall that the inside walk begins at $a'_b$, ends at $b_a$, and has even length.

**Case 2.1** $[a_b \not\equiv a_c]$**:** Walk $P$ begins at $a_b$ and first uses arc $(a_b, a'_b)$. The only pairing path that can cross this arc is the one that pairs $a$ with $c$ and because $a_b \not\equiv a_c$, this can only occur if $a'_b \equiv a_c$, $y_{ac} > 0$ and $num(a) <$

$num(c)$, which is when $\overrightarrow{ac}$ crosses $(a_b, a'_b)$. This information will be used to find $Odd_P$ in the following two sub-cases.

**Case 2.1.1** $[b_a \not\equiv b_d]$ **(Figure 4.13(a)):** Let $P$ be $a_b$, $(a_b, a'_b)$, $I$. Thus, $P$ has odd length. If $a'_b \equiv a_c$, $y_{ac} > 0$ and $num(a) < num(c)$, then $Odd_P = \neg Odd_I$, otherwise $Odd_P = Odd_I$. From the application of Theorem 3.2.1 to $P$, $b_a$ has the same color as $a_b$ if $Odd_P$ is *true* and the opposite color otherwise.

**Case 2.1.2** $[b_a \equiv b_d]$ **(Figure 4.13(b)):** Let $P$ be $a_b$, $(a_b, a'_b)$, $I$, $(b_a, b'_a)$, $b'_a$. Thus, $P$ has even length. As in Case 1.1.2, if $num(d) < num(b)$ or both $num(b) < num(d)$ and $y_{bd} = 0$, then arc $(b_a, b'_a)$ is crossed by a pairing path. Then $Odd_P$ is determined from $Odd_I$ in the straightforward way by negating $Odd_I$ for each of the arcs $(a_b, a'_b)$ and $(b_a, b'_a)$ that are crossed by a pairing path. From the application of Theorem 3.2.1 to $P$, $b'_a$ has the opposite color of $a_b$ if $Odd_P$ is *true* and the same color otherwise.

**Case 2.2** $[a_b \equiv a_c]$**:** Walk $P$ begins at $a'_b$.

**Case 2.2.1** $[b_a \not\equiv b_d]$ **(Figure 4.13(c)):** Let $P$ be $I$. Thus, $P$ has even length and $Odd_P = Odd_I$. From the application of Theorem 3.2.1 to $P$, $b_a$ has the opposite color of $a'_b$ if $Odd_P$ is *true* and the same color otherwise.

**Case 2.2.2** $[b_a \equiv b_d]$ **(Figure 4.13(d)):** Let $P$ be $I$, $(b_a, b'_a)$, $b'_a$. Thus, $P$ has odd length. As in Case 1.1.2, if $num(d) < num(b)$ or both $num(b) < num(d)$ and $y_{bd} = 0$, then arc $(b_a, b'_a)$ is crossed by a pairing path. From the application of Theorem 3.2.1 to $P$, $b'_a$ has the same color as $a'_b$ if $Odd_P$ is *true* and the opposite color otherwise.

By applying this logic to each of the eleven clear fields that compose the coloring tree, the color of each pentagon can be determined when pentagon 0 is colored white. This gives the color of each of the six selected clear fields when the clear field that pairs pentagon 0 is colored white. As stated previously, the other coloring is obtained by swapping the colors of each of the six clear fields. For each of these colorings, the penalty is given by Equation (3.2).

The pseudo-code in Algorithm 4.5 summarizes the logic in each of the cases and shows how the two penalties are computed. The input to the algorithm is a coloring tree and a selection of six proper clear fields with no dual vertices in common. The

```
1  Color pentagon 0 white;
2  foreach Coloring Tree Clear Field ⬜/ab do /* color of a known, b unknown */
3  |  Locate c, d, ⬜/ac, ⬜/bd, a_b, a'_b, a_c, b_a, b'_a, and b_d;
4  |  if c ≡ b then
5  |  |  Color the pentagon at b the same color as the pentagon at a;
6  |  else
7  |  |  Determine color of a_b and a'_b from color of pentagon at a and clockwise positions
   |  |  from a_c;
8  |  |  Odd_P ← Odd_I; /* Find Odd_I from Algorithm 4.6 as in Section 4.4.2 */
9  |  |  switch based on y_ab, a_b, a_c, b_a, and b_d do
10 |  |  |  case y_ab > 0, a_b ≢ a_c, b_a ≢ b_d                          /* Case 1.1.1 */
11 |  |  |  |  if Odd_P then Color b_a the same as a_b else Color b_a the opposite of a_b;
12 |  |  |  case y_ab > 0, a_b ≢ a_c, b_a ≡ b_d                          /* Case 1.1.2 */
13 |  |  |  |  if (num(d) < num(b)) or (num(b) < num(d) and y_bd = 0) then neg(Odd_P);
14 |  |  |  |  if Odd_P then Color b'_a the opposite of a_b else Color b'_a the same as a_b;
15 |  |  |  case y_ab > 0, a_b ≡ a_c, b_a ≢ b_d                          /* Case 1.2.1 */
16 |  |  |  |  if (num(c) < num(a)) or (num(a) < num(c) and y_ac = 0) then neg(Odd_P);
17 |  |  |  |  if Odd_P then Color b_a the opposite of a'_b else Color b_a the same as a'_b;
18 |  |  |  case y_ab > 0, a_b ≡ a_c, b_a ≡ b_d                          /* Case 1.2.2 */
19 |  |  |  |  if (num(c) < num(a)) or (num(a) < num(c) and y_ac = 0) then neg(Odd_P);
20 |  |  |  |  if (num(d) < num(b)) or (num(b) < num(d) and y_bd = 0) then neg(Odd_P);
21 |  |  |  |  if Odd_P then Color b'_a the same as a'_b else Color b'_a the opposite of a'_b;
22 |  |  |  case y_ab = 0, a_b ≢ a_c, b_a ≢ b_d                          /* Case 2.1.1 */
23 |  |  |  |  if a'_b ≡ a_c and y_ac > 0 and num(a) < num(c) then neg(Odd_P);
24 |  |  |  |  if Odd_P then Color b_a the same as a_b else Color b_a the opposite of a_b;
25 |  |  |  case y_ab = 0, a_b ≢ a_c, b_a ≡ b_d                          /* Case 2.1.2 */
26 |  |  |  |  if a'_b ≡ a_c and y_ac > 0 and num(a) < num(c) then neg(Odd_P);
27 |  |  |  |  if (num(d) < num(b)) or (num(b) < num(d) and y_bd = 0) then neg(Odd_P);
28 |  |  |  |  if Odd_P then Color b'_a the opposite of a_b else Color b'_a the same as a_b;
29 |  |  |  case y_ab = 0, a_b ≡ a_c, b_a ≢ b_d                          /* Case 2.2.1 */
30 |  |  |  |  if Odd_P then Color b_a the opposite of a'_b else Color b_a the same as a'_b;
31 |  |  |  case y_ab = 0, a_b ≡ a_c, b_a ≡ b_d                          /* Case 2.2.2 */
32 |  |  |  |  if (num(d) < num(b)) or (num(b) < num(d) and y_bd = 0) then neg(Odd_P);
33 |  |  |  |  if Odd_P then Color b'_a the same as a'_b else Color b'_a the opposite of a'_b;
34 |  |  |  Determine color of the pentagon at b from b_a or b'_a and clockwise positions from
   |  |  |  b_d;
35 whitePen ← 0; blackPen ← 0; /* Penalty if pentagon 0 is white or black */
36 foreach Selected Clear Field ⬜/ac do /* Calc penalties of 6 selected CFs */
37 |  if pentagon at a is white then
38 |  |  whitePen ← whitePen + 2x_ac + y_ac; blackPen ← blackPen + 2y_ac + x_ac;
39 |  else
40 |  |  whitePen ← whitePen + 2y_ac + x_ac; blackPen ← blackPen + 2x_ac + y_ac;
41 return whitePenalty, blackPenalty;
```

**Algorithm 4.5**: An $O(1)$ algorithm that, given a coloring tree, computes the penalties of the two colorings of six proper clear fields that have no dual vertices in common. The function $neg(x)$ sets $x \leftarrow \neg x$.

output is a pair of values $whitePen$ and $blackPen$ which represent the penalties when pentagon 0 is colored white or black, respectively.

The logic presented is just a series of simple comparisons and so it can be executed quickly. For each of the eleven clear fields of the coloring tree, the appropriate one of the nine cases (including the special case) can be determined in constant time. Then, the color can be determined in constant time, assuming a query to the oracle that provides $Odd_I$ takes constant time (Section 4.4.2 provides such an oracle). The coloring tree is independent of any selection of six clear fields and constructing it takes $O(n)$ time and occurs once for the entire algorithm.

### 4.4.2 Detect Inside-Walk Crossings

Section 4.4 deals with evaluating each selection of six proper clear fields with no dual vertices in common and applying each of the two possible pentagon colorings to determine the configuration with the minimum penalty. Section 4.4.1 gave a method to compute the two penalties, one for each pentagon coloring for each selection of six clear fields. This method relied on an oracle to provide the value $Odd_I$ for a specific inside walk $I$ (Figure 4.11), which is true if the number of edges of $I$ crossed by the six pairing paths (one per selected clear field) is odd and otherwise is false. This section provides such an oracle that takes $O(n)$ time for initialization that occurs once and then $O(1)$ time to answer a query for $Odd_I$.

Throughout this section, let $\angle ab$ be a proper clear field and label $a$ and $b$ such that $num(a) < num(b)$. Let $i$ be the zero-based index of $\angle ab$ in the list of all of the $C$ proper clear fields. Let $\angle cd$ be a coloring tree clear field and label $c$ and $d$ such that $num(c) < num(d)$. Let $j$ be the zero-based index of $\angle cd$ in the list of the eleven coloring tree clear fields. Let $I_{cd}$ be the inside walk of $\angle cd$.

The task is to answer queries for $Odd_{I_{cd}}$ in $O(1)$ time. The output of the algorithm in this section is a collection of $OddCrossings$ arrays that are created such that $OddCrossings_i[j]$ is *true* if the number of times $\overrightarrow{ab}$ crosses $I_{cd}$ is odd and *false* otherwise. Given a selection of six proper clear fields with no dual vertices in common from Section 4.4.1, let their unique indexes be $i_0, i_1, \ldots, i_5$. Then $Odd_{I_{cd}}$ is computed as

$$\begin{aligned} Odd_{I_{cd}} \;=\; & (((((OddCrossings_{i_0}[j] \oplus OddCrossings_{i_1}[j]) \oplus OddCrossings_{i_2}[j]) \\ & \oplus OddCrossings_{i_3}[j]) \oplus OddCrossings_{i_4}[j]) \oplus OddCrossings_{i_5}[j]) \end{aligned}$$

where $\oplus$ is the exclusive-or operation.

An overview of the approach is as follows.

1. Initialize each $OddCrossings_i[j]$ value to *false*.

2. For each choice of a coloring tree clear field $\square\!\!/cd$, visit the dual vertices on the walk $\overrightarrow{cd}$ and leave a token at each of the $x_{cd} + y_{cd} - 1$ vertices of degree six. Tokens are left at these dual vertices because the primal edges that compose $I_{cd}$ are those on the right side of $\overrightarrow{cd}$. Each token records:

   the value $j$, which is a unique number from 0 to 10 that identifies $\square\!\!/cd$

   the dual vertex number of the previous vertex in $\overrightarrow{cd}$, and

   whether or not the wide right turn occurs at this dual vertex (which occurs once if $y_{cd} > 0$).

3. Loops through each choice of a proper clear field $\square\!\!/ab$. Recall that if $\square\!\!/ab$ were to be one of the six selected clear fields in Section 4.4.1, then the pairing path would use the edges of $\overrightarrow{ab}$. Visit the dual vertices on the walk $\overrightarrow{ab}$ and looks for tokens left at each face. By using the cases shown in Figure 4.14, each token is analyzed to see if it indicates a crossing with the inside walk represented by the token.

By first leaving the tokens for each of the eleven coloring tree clear fields and then walking $\overrightarrow{ab}$ for every clear field, the crossings are detected for every pair of $\square\!\!/ab$ and $\square\!\!/cd$.

Pseudo-code for an algorithm to perform such a calculation is given in Algorithm 4.6. The initialization consists of creating the $OddCrossings$ arrays and initializing the values to *false* (Line 2) as well as creating an empty token list at each dual vertex of degree six (Line 3). For each coloring tree clear field $\square\!\!/cd$, the algorithm walks along $\overrightarrow{cd}$ and leaves a token at each dual vertex of degree six (Line 5).

To detect the crossings in $O(n)$ time, a method is followed similar to that used in Algorithm 4.4, which walked the sixty axes instead of each clear field separately. There are some notable differences from Algorithm 4.4, particularly with regard to which clear fields are represented by an axis. The algorithm processes each of the sixty axes one-by-one (Line 6). For a particular axis $A_\ell$ ($0 \le \ell \le 59$), the algorithm constructs an array $CFs$ of associated clear fields (Line 7) as follows. Clear field $\square\!\!/ab$ is included in $CFs$ if and only if walk $\overrightarrow{ab}$ uses the axis. That is, $\square\!\!/ab$ is in $CFs$ if $A_\ell$ is the $y$-axis of the quintant of $\square\!\!/ab$ at $a$ or if $A_\ell$ is the $x$-axis of the quintant of $\square\!\!/ab$ at $b$. Along with each clear field $\square\!\!/ab$ in $CFs$, a *sort-value* is stored, which is

```
1  foreach Clear Field CF_i (i ← 0 to C − 1) do
2      Create array OddCrossings_i of size 11 to store in OddCrossings_i[j] whether the
       number of crossings of CF_i with the j-th coloring tree inside walk is odd; initialize
       to false;
3  Create empty token list at each hexagon;
4  foreach coloring tree clear field CTCF_j (j ← 0 to 10) do
5      Label CTCF_j as ⟌cd with num(c) < num(d) and walk along cd⃗ leaving a token
       at each of the x_cd + y_cd − 1 dual vertices of degree six;
6  foreach Axis A_ℓ (ℓ ← 0 to 59) do
7      Create CFs, an array of clear fields that use A_ℓ as an x- or y-axis, sorted by
       increasing sortValue (x- or y-value as appropriate);
8      Create OddCrossingsByAxis, an array of size 11 initialized to false;
9      if length(CFs) = 0 then skip to the next axis;
       /* OddCrossingsByAxis[j] complements each time A_ℓ crosses CTCF_j   */
10     k ← 0 ;  /* k is the smallest index of a CF in CFs with sort-value ≥ d
       */
       /* Walk along A_ℓ as far as the largest associated clear field      */
11     for d ← 1 to CFs[length(CFs) − 1].sortValue do
12         Move to the hexagon distance d along axis A_ℓ;
13         while CFs[k].sortValue < d do                /* Increment k as needed */
14             for j ← 0 to 11 do     /* Record crossings CFs[k] had with CTCF_j */
15                 OddCrossings_CFs[k][j] ←
                   OddCrossings_CFs[k][j] ⊕ OddCrossingsByAxis[j];
16             k ← k + 1;
17         foreach token t in the token list at current hexagon do
18             if Case 4.14(a) or 4.14(b) or 4.14(c) or 4.14(d) or 4.14(e) or 4.14(f) then
                   /* Record crossing between token's CTCF t.CF and all CFs using
                      A_ℓ with sort-value ≥ d                                  */
19                 OddCrossingsByAxis[t.CF] ← ¬OddCrossingsByAxis[t.CF];
20             if Case 4.14(d) or 4.14(e) or 4.14(f) then
                   /* Line 19 recorded crossing for CFs with sort-value ≥ d, but
                      should only be for sort-value > d, so undo the over-count if
                      sort-value = d (at most 2 CFs).                          */
21                 if CFs[k].sortValue = d then
22                     OddCrossings_CFs[k][t.CF] ← ¬OddCrossings_CFs[k][t.CF];
23                     if k < length(CFs) − 1 and CFs[k + 1].sortValue = d then
24                         OddCrossings_CFs[k+1][t.CF] ← ¬OddCrossings_CFs[k+1][t.CF];
25         while k < length(CFs) do     /* Loop through last 1 or 2 CFs in CFs */
26             for j ← 0 to 10 do       /* Record crossings CFs[k] had with CTCF_j */
27                 OddCrossings_CFs[k][j] ← OddCrossings_CFs[k][j] ⊕ OddCrossingsByAxis[j];
28             k ← k + 1;
```

**Algorithm 4.6**: A $O(n)$ algorithm for determining whether the number of times each clear field's pairing path crosses each coloring tree's inside walk is even or odd.

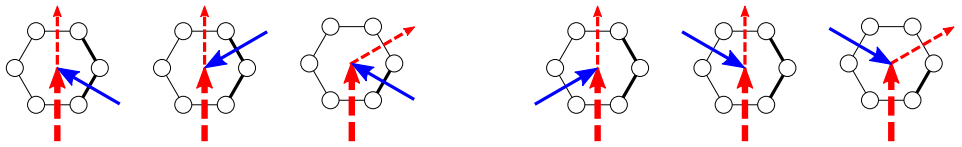



Figure 4.14: The six cases in which arc $(v_{d-1}, v_d)$ of an axis (thin, solid) meets a token for the walk $\overrightarrow{cd}$ of a coloring tree clear field $\square cd$ (dashed) to detect a recent or future crossing of the axis and $I_{cd}$ (**bold** primal edges). Arc $(u, v_d)$ is shown as **bold**, dashed.

$y_{ab}$ if $A_\ell$ is acting as the $y$-axis and $x_{ab}$ if $A_\ell$ is acting as the $x$-axis. The clear fields in $CFs$ are sorted by this sort-value in increasing order. The goal is to visit the dual vertices of each axis and detect when the axis crosses over each of the eleven inside walks marked by tokens and then mark the appropriate clear fields in $CFs$ as having crossed the inside walks.

Given a particular axis, the algorithm visits its dual vertices with the distance walked along the axis recorded as the value $d$ (Line 12). At each dual vertex $d$ steps along the axis, let $v_d$ be the dual vertex at distance $d$ along the axis, then arc $(v_{d-1}, v_d)$ was followed to arrive at $v_d$ and arc $(v_d, v_{d+1})$ will be followed next.

The list of tokens at $v_d$ is then analyzed (Line 17). Let $t$ be a token in the list. Token $t$ contains information about the previously visited vertex along the walk $\overrightarrow{cd}$ that it represents. Let $u$ be the dual vertex previously visited by $t$. Thus, arc $(u, v_d)$ is in $\overrightarrow{cd}$. As shown in Figure 4.14, one or two of the primal edges of the hexagon at $v_d$ are in $I_{cd}$, depending on whether or not $t$ indicates that $\overrightarrow{cd}$ made a wide right turn at $v_d$. If a wide right turn is made at $v_d$ (Figures 4.14(c) and 4.14(f)), then the dual edge 5 clockwise positions around $v_d$ after dual edge $(u, v_d)$ crosses the only primal edge of the hexagon at $v_d$ that is in $I_{cd}$. If no turn is made at $v_d$ (Figures 4.14(a), 4.14(b), 4.14(d), and 4.14(e)), then the two dual edges 4 clockwise positions and 5 clockwise positions around $v_d$ after $(u, v_d)$ each cross a primal edge in $I_{cd}$.

Six mutually exclusive cases are considered for when arc $(v_{d-1}, v_d)$ has just crossed an edge of $I_{cd}$ (Figures 4.14(a), 4.14(b), and 4.14(c)) and when arc $(v_d, v_{d+1})$ will cross an edge of $I_{cd}$ (Figures 4.14(d), 4.14(e), and 4.14(f)). In the cases where the axis has just crossed the inside path, every clear field in $CFs$ with sort-value greater than or equal to $d$ indicates a crossing. In the cases where the axis is about to cross the inside path, every clear field in $CFs$ with sort-value greater than $d$ indicates a crossing.

To aid the bookkeeping of the crossings, an array $OddCrossingsByAxis$ of eleven boolean values is created. Before any dual vertices of an axis are walked, these eleven

values are initialized to *false* (Line 8). As the dual vertices of an axis are visited, the value $OddCrossingsByAxis[j]$ records whether or not the number of times the axis has crossed the inside path of coloring tree clear field number $j$ is odd (the details of maintaining $OddCrossingsByAxis$ are provided later). Recall that the sort-value of a clear field in $CFs$ is the number of dual vertices of degree six on the current axis that are used by the clear field. As in Algorithm 4.4, the value $k$ is maintained (Lines 10 and 16) to be the smallest index such that the sort-value of $CFs[k]$ is at least $d$. When $d$ is incremented, the algorithm checks to see if $CFs[k].sortValue < d$ (Line 13) and if it is, $k$ is incremented (Line 16). If this is the case, then before $k$ is incremented, the algorithm sets

$$OddCrossings_{CFs[k]}[j] \leftarrow OddCrossings_{CFs[k]}[j] \oplus OddCrossingsByAxis[j]$$

for each $0 \leq j \leq 10$ (Line 15). This updates each global odd/even value to include the odd/even value of the number of times the axis has crossed each inside walk, because clear field $CFs[k]$ uses only the dual vertices $v_0$ through $v_{d-1}$ of the axis. Such an update is also done at the end of the axis, when $d$ reaches the value of the largest sort-value in $CFs$ (Line 27).

What remains to be discussed is how to adjust the values in $OddCrossingsByAxis$ as the algorithm increments $d$ and visits $v_d$. The value $OddCrossingsByAxis[j]$ represents the odd/even value of the number of times the clear fields in $CFs$ with index $k$ or greater (those with a sort-value of at least $d$) cross coloring tree clear field number $j$. The cases in which a token indicates the axis has just crossed an inside walk (4.14(a), 4.14(b), and 4.14(c)) and when it indicates the axis is about to cross an inside walk (4.14(d), 4.14(e), and 4.14(f)) are handled differently. In the former three cases, every clear field in $CFs$ with a sort-value greater than or equal to $d$ crosses the inside walk. In the latter three cases, every clear field in $CFs$ with a sort-value greater than $d$ crosses the inside walk. Instead of recording a crossing for each clear field with sort-value greater than or equal to $d$ for the former cases and with sort-value greater than $d$ for the latter cases, the algorithm does two things. First, it records a crossing for all clear fields with a sort-value greater than or equal to $d$ for all six cases by negating the appropriate $OddCrossingsByAxis[j]$ value (Line 19). Second, it effectively subtracts a crossing for those with sort-value equal to $d$ in the latter cases by negating the appropriate $OddCrossings$ value (Lines 21-24).

The running time analysis of Algorithm 4.6 is straightforward. The initial loop through the clear fields takes $O(C)$ time because the initialization of each of the

*OddCrossings* arrays takes constant time. To create an empty token list at each hexagon takes $O(n)$ time for the $O(n)$ hexagons. For each coloring tree clear field $\angle\!\!\!/cd$, walking along $\overrightarrow{cd}$ takes $O(n)$ time and there are $O(1)$ coloring tree clear fields. The analysis of walking along the axes is more involved. Note that the two while loops together take $O(C)$ time for each axis. There may be at most eleven tokens at a given hexagon and it is possible to determine which case of Figure 4.14 applies in constant time. Therefore, visiting the $O(n)$ dual vertices of degree six of an axis takes $O(n)$ time. For each axis, initialization of the array $CFs$ can be done in $O(C \lg C)$ time by scanning the $O(C)$ clear fields to find the subset that use the axis and then naively sorting the array values. Because $C \in O(\lg n)$, walking the sixty axes takes $O(n)$ time, for $O(n)$ time overall.

### 4.4.3 Summary

Phase 3 consists of several parts that fit together to find an optimal selection of six proper clear fields that have no dual vertices in common and a coloring of the clear fields such that the minimum penalty is achieved. The following procedure is used:

1. As in Section 4.4.1, use a breadth-first search to find eleven proper clear fields to construct the coloring tree. This takes $O(\lg n)$ time.

2. For each pair of a proper clear field $\angle\!\!\!/ab$ (labeled such that $num(a) < num(b)$) and a coloring tree clear field $\angle\!\!\!/cd$, determine whether the pairing path following $\overrightarrow{ab}$ crosses the inside path $I_{cd}$ an even or odd number of times (Algorithm 4.6, Section 4.4.2). This takes $O(n)$ time.

3. Find an optimal pairing by considering each selection of six proper clear field that have no dual vertices in common (and thus pair the degree five vertices). There are $O(\lg^6 n)$ such selections. For each selection:

   (a) Use the coloring tree and the output of Algorithm 4.6 to determine the color of the six selected clear fields assuming pentagon 0 is white, then compute the penalty of this configuration (Algorithm 4.5, Section 4.4.1). This takes $O(1)$ time.

   (b) Invert the colors of the clear fields found in the previous step to find the other coloring choice, then compute the penalty of this configuration (Algorithm 4.5, Section 4.4.1). This takes $O(1)$ time.

   (c) Remember the selection of six clear fields and an associated color that achieves the minimum penalty of all such configurations considered so far. This takes $O(1)$ time.

Altogether, this step takes $O(\lg^6 n)$ time.

Overall, these parts take $O(n)$ time and so the running time of Phase 3 is $O(n)$.

## 4.5 Phase 4: Construct a Maximum Independent Set

The method of obtaining a maximum independent set from an optimal pairing was discussed briefly in Section 4.1. First, choose a pairing path for each of the six proper clear fields that compose the optimal pairing. For each primal edge that corresponds to a dual edge on a pairing path, one endpoint is colored gray. The graph induced by the uncolored vertices is bipartite. The optimal pairing indicates a color for each clear field and thus each pentagon. Color the uncolored vertices of pentagon 0 as specified by the color for the optimal pairing. A simple breadth-first search on the uncolored vertices can color each such that no two whites or blacks are adjacent. The maximum independent set is given by the vertices that are colored white in this manner.

Identifying the pairing paths and coloring the gray vertices takes $O(n)$ time. Then coloring the rest of the vertices using a simple breadth-first search takes $O(n)$ time. Hence the whole graph can be colored in $O(n)$ time.

## 4.6 Conclusion

An algorithm to find a maximum independent set of vertices of a fullerene was presented by breaking the problem into four phases that run sequentially. Each phase was discussed and the runtime was found to be $O(n)$ in all four cases. Therefore, the entire algorithm runs in $O(n)$ time.

This algorithm was run on the 10,190,782 distinct fullerene graphs on 120 or fewer vertices. The results agreed with computations by the backtracking algorithm used in [23].

A number of open problems still remain. These include the following problems.

**Speed Improvements.** During the development of this algorithm, the author wrote a simpler version (shorter algorithm) that considered all of the clear fields, not just the proper clear fields. This resulted in a running time that was in $O(n^6)$, however the implementation timing results showed that it actually ran faster on average than the $O(n)$ algorithm. Figure 4.15 shows a plot of the running times of the implementation of the $O(n)$ algorithm, the $O(n^6)$ algorithm and the backtracking algorithm used in [23]. The slower running time on average of the $O(n)$ algorithm over the $O(n^6)$ algorithm is likely due to the overhead required to only discover the proper clear fields and the algorithmic tricks used to decrease the theoretical running time in Phases 2 and 3. It is an open problem to improve the running time of the

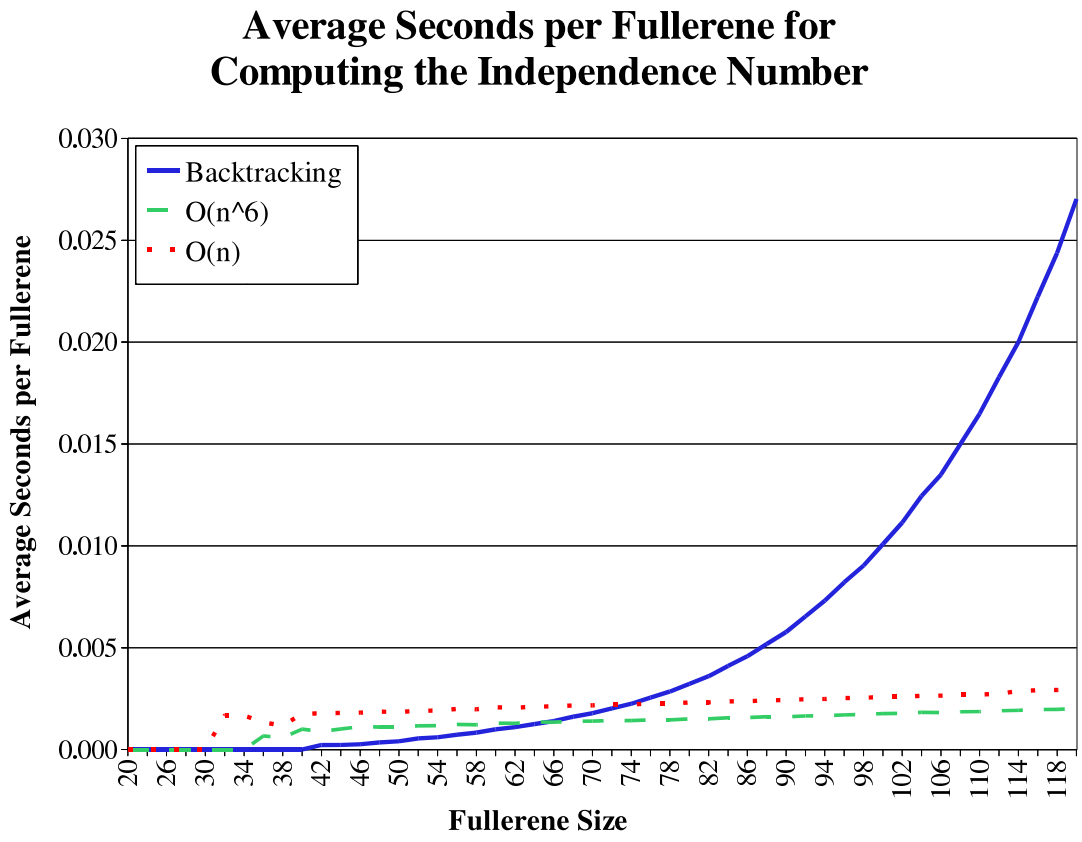



Figure 4.15: Comparison of implementation timings of independence number algorithms.

$O(n)$ algorithm to make it competitive with the $O(n^6)$ algorithm. It is not clear how practical it is to put effort into this area when the difference is only a thousandth of a second, though this equates to a difference in the running time for all fullerenes on 200 vertices of approximately 2.5 days. One possible area of improvement is to combine the work in Phase 2 and the pre-processing step of Phase 3 because they have a similar approach of walking the 60 axes and leaving tokens.

**Number of Maximum Independent Sets.** There is some interest [43] in enumerating or counting the number of maximum independent sets of a fullerene. This can be easily accomplished by modifying the algorithm to find all optimal pairings instead of just the first one it encounters. If only a count is desired, Theorem 3.2.3(i) can be applied to each optimal pairing and Phase 4 can be omitted since there is no need to construct the sets. Such a counting algorithm would have a running time in $O(n)$. Note that the number of sets counted or enumerated in this manner would not filter out isomorphic copies in the case where the fullerene has symmetry. It would

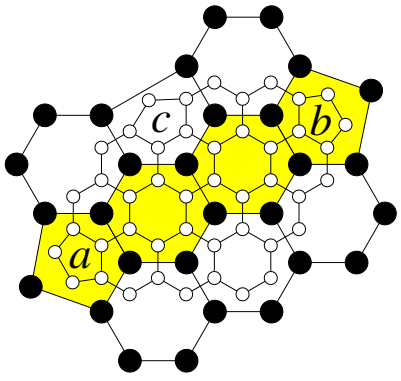
be interesting to see if any fullerenes have a unique (up to isomorphism) maximum independent set.

# Chapter 5
# Independent Sets of Some Fullerene Families

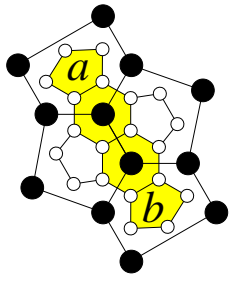
This chapter looks at several families of fullerenes and their related properties. These families are viewed in the context of a maximum independent set. The families discussed here are collections of fullerenes that have a property in common. These properties, for example, may be that the fullerenes are the result of a transformation from another fullerene, or that the fullerenes attain the maximum possible independence number for their size.

## 5.1 Leapfrogs

The *leapfrog transformation* is a well-defined one-to-one transformation that converts any fullerene into a fullerene with three times the number of vertices [25, 3.4]. The procedure to create a leapfrog fullerene $F^1$ from fullerene $F^0$ is to first add two vertices on each side of an edge of $F^0$ and join them with an edge. For $n$ vertices of $F^0$, there are $3n/2$ edges and therefore $3n$ vertices of $F^1$. Adding vertices in this manner makes each face of $F^0$ correspond with five or six vertices of $F^1$, depending on whether the face is a pentagon or a hexagon, respectively. For each face of $F^0$, the vertices of




(a) A clear field in $F^0$ between $a$ and $b$ (shaded) does not imply the existence of a clear field in $F^1$ between $a$ and $b$ due to $c$.




(b) A clear field in $F^1$ between $a$ and $b$ (shaded) does not imply the existence of a clear field in $F^0$ between $a$ and $b$.

Figure 5.1: Examples showing that a clear field may exist in the original ($F^0$, black) or leapfrog ($F^1$, white) fullerenes without a corresponding clear field in the other.

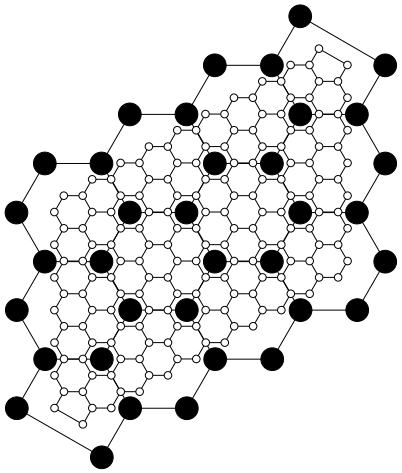


Figure 5.2: An example of a clear field (black) with dimensions $(3, 2)$ and its corresponding double-leapfrog clear field (white) with dimensions $(9, 6)$.

$F^1$ are joined by an equal number of edges by adding an edge between each pair of vertices that were assigned to adjacent edges of $F^0$.

When the leapfrog fullerene $F^1$ is created from a fullerene $F^0$, the structure is changed. Each pentagon of $F^0$ corresponds with exactly one pentagon of $F^1$. However, a clear field in $F^0$ between two pentagons does not necessarily correspond with a clear field in $F^1$. Figure 5.1(a) shows a shaded clear field in $F^0$ between pentagons $a$ and $b$ with dimensions $(3, 0)$. The leapfrog patch that corresponds to this clear field would form a clear field with dimensions $(3, 3)$, except that pentagon $c$ is in the region. Conversely, Figure 5.1(b) shows a shaded clear field in $F^1$ between pentagons $a$ and $b$ with dimensions $(3, 0)$. There is, however, no corresponding clear field in $F^0$.

Knowledge of the existence and locations of the clear fields of $F^0$ does not give knowledge about the clear fields in $F^1$. This makes it difficult to determine a maximum independent set for an arbitrary leapfrog fullerene, even if the optimal set of clear fields is known for the original fullerene.

## 5.2 Double Leapfrogs

A double-leapfrog fullerene $F^2$ is created from a fullerene $F^0$ by applying the leapfrog transformation twice. If $F^0$ has $n$ vertices, then $F^2$ has $9n$ vertices. When a double-leapfrog fullerene is created from a fullerene, it retains the original structure and the distances between the original faces are three times the original distances. Thus, odd distances remain odd and even distances remain even.

Because $F^2$ has the same structure as $F^0$, the quintants of $F^0$ map directly to the quintants of $F^2$. The double leapfrog transformation triples the distances between

dual vertices, so the clear fields of $F^0$ and $F^2$ have a one-to-one correspondence and a clear field in $F^0$ with dimensions $(x, y)$ corresponds with a clear field in $F^2$ with dimensions $(3x, 3y)$. Figure 5.2 shows an example of a clear field subgraph in $F^0$ and the corresponding clear field subgraph in $F^2$.

The strong relationship between the clear fields in $F^0$ and $F^2$ imply that the optimal clear fields in the original fullerene give the optimal clear fields in the double-leapfrog. If a set of six optimal clear fields of the original fullerene is known along with the colors of the clear fields for the optimal arrangement, the same selection and colors can be used on the dual-leapfrog fullerene to construct a maximum independent set. If one is only concerned with the size of the maximum independent set, it is easy to find $\alpha(F^2)$ from $\alpha(F^0)$, as shown in the next theorem.

**Theorem 5.2.1.** *Let $F^0$ be a fullerene on $n_0$ vertices and let $F^2$ be the $9n_0$-vertex fullerene that is the result of two successive leapfrog transformations of $F^0$. Then*

$$\alpha(F^2) = 3(n_0 + \alpha(F^0)).$$

*Proof.* When two leapfrog transformations are applied, the structure is the same and the distances in the dual are tripled. Thus, the penalty of each clear field (either $2x+y$ or $2y + x$ for a clear field with dimensions $(x, y)$) is also tripled. Let $p_0$ be the sum of the penalties of a set of six optimal clear fields of $F^0$. Then $\alpha(F^0) = n_0/2 - p_0/3$ (Equation (3.2)) and so $\alpha(F^2) = 9n_0/2 - 3p_0/3 = 3(n_0 + \alpha(F^0))$. □

This result can be generalized for any even number of successive leapfrog transformations.

**Theorem 5.2.2.** *Let $F^0$ be a fullerene on $n_0$ vertices and let $F^{2d}$ be the $9^d n_0$-vertex fullerene that is the result of $2d$ successive leapfrog transformations of $F^0$. Then*

$$\alpha(F^{2d}) = \frac{3^d}{2}(3^d - 1)n_0 + 3^d\alpha(F^0).$$

*Proof.* Theorem 5.2.1 can be applied repeatedly where the fullerene after $2i$ leapfrog transformations has $9^i$ vertices. This yields the recurrence

$$f(d) = 3(9^{d-1}n_0 + f(d-1)) \text{ with } f(0) = \alpha(F^0),$$

the solution of which gives the result. □

**Theorem 5.2.3.** *Let $F^{2d}$ be a fullerene on $n_{2d}$ vertices that is the result of $2d$ successive leapfrog transformations from some other fullerene. Then,*

$$\alpha(F^{2d}) \geq \frac{n_{2d}}{2} - \frac{n_{2d}}{8 \cdot 3^d}.$$

*Proof.* It has been shown that $\alpha(F^0) \geq 3n_0/8$ for any fullerene $F^0$ on $n_0$ vertices [34]. Substituting this into the result of Theorem 5.2.2 gives

$$\begin{aligned} \alpha(F^{2d}) &\geq \frac{3^d}{2}(3^d - 1)n_0 + \frac{3^d 3n_0}{8} \\ &= \frac{9^d n_0}{2} - \frac{3^d n_0}{8}. \end{aligned} \tag{5.1}$$

Substituting $n_{2d}/9^d = n_0$ yields the result. □

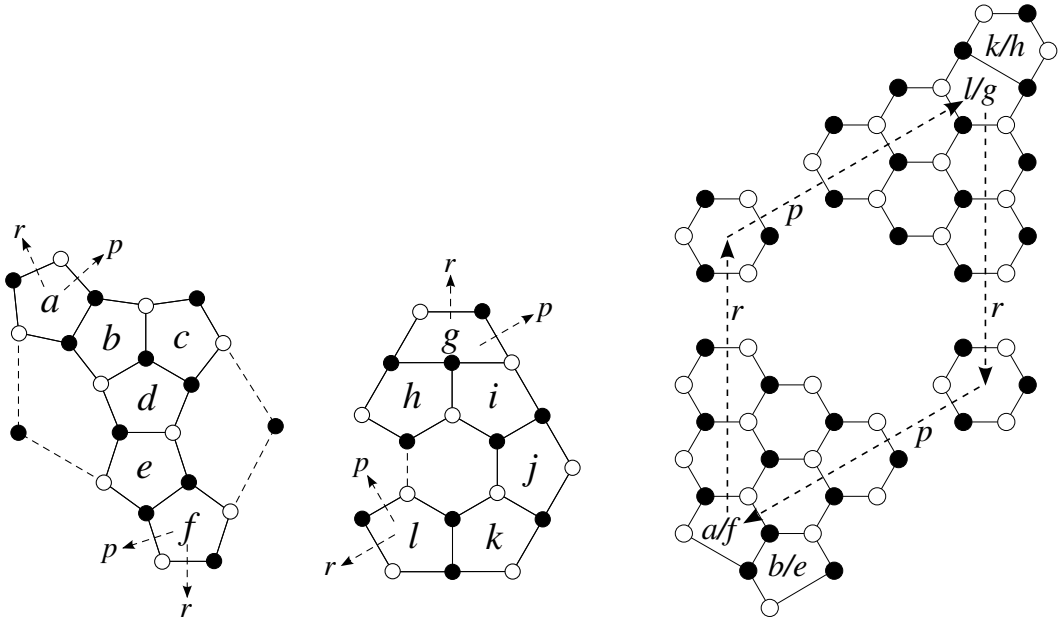
This lower bound has the form $n/2 - k$. At first glance, it may appear that one can use this new bound to find a fullerene with an arbitrarily small $k > 0$ by a sufficient choice of $d$. This is not the case because $n$ and $d$ are not independent; $n$ grows exponentially with $d$. Equation (5.1) shows that the distance $k$ from $n/2$ grows exponentially with $d$. However, in another sense, this bound improves because the ratio $k/n = 1/(8 \cdot 3^d)$ decays exponentially with increasing $d$. This is to be expected because as $d$ increases, the distance between the pentagons grows linearly while the vertex count grows exponentially.

Furthermore, notice that $n_0 \geq 20$ because the smallest fullerene has 20 vertices. This gives the following lower bound that relies only on $d$:

$$\alpha(F) \geq 10 \cdot 9^d - \frac{5 \cdot 3^d}{2}.$$

## 5.3 An Upper-Bound Achieving Family

There is a fullerene family that attains the upper bound of $n/2 - 2$ and the family includes a fullerene at all values of $n$ for which a fullerene exists except 20, 24, 28, and 32. A fullerene in this family has its twelve pentagons arranged in two patches of six pentagons, with the patches at opposite ends of the fullerene. Figure 5.3(a) shows these two patches as well as the vertices in the independent set (white). The twelve pentagons are labeled with letters $a$ through $l$. In order to attain the upper bound, a Graver coloring must have a total penalty of 6. The smallest penalty that a clear field can have is 1 and this only occurs if the clear field has dimensions $(1, 0)$ and is colored black such that its penalty is $1 + 2 \cdot 0$. A fullerene in this family attains the




(a) Two pentagon patches (b) Clear fields joining the patches

Figure 5.3: The two patches of pentagons used to construct a fullerene with $\alpha(F) = n/2 - 2$. To construct the fullerene, pair pentagons $a$ and $l$ as well as $f$ and $g$ with clear fields of size $(p, r)$ as indicated.

upper bound by using six such clear fields. These clear fields are $\square\!/ab$, $\square\!/cd$, $\square\!/ef$, $\square\!/gh$, $\square\!/ij$, and $\square\!/kl$.

In addition to the six clear fields that form the optimal pairing, two clear fields $\square\!/al$ and $\square\!/fg$ also exist that use the quintants indicated by the arrows in Figure 5.3(a). A fullerene in this family can be specified by giving $\square\!/al$ and $\square\!/fg$ dimensions $(p, r)$ for any choice of $p$ and $r$ such that $p \geq 0$, $r \geq 0$, and $p + r > 0$. Figure 5.3(b) illustrates $\square\!/al$ ($\square\!/fg$) as well as the locations of pentagons $b$ ($e$) and $k$ ($h$). The coloring pattern in Figure 5.3(b) is aligned with that in Figure 5.3(a). Any choice of $p$ and $r$ results in a coloring in which every hexagon contains three white vertices and every pentagon contains two white vertices. Hence, the fullerene has an independent set order as large as possible with $n/2 - 2$ white vertices.

The complete fullerene signature (see [29, 30]) is shown in Figure 5.4. For $n = 26$, 30, and 34, the symmetry-group orders are 12, 4, and 6, respectively. For all other choices of $n$, the constructed fullerenes have no symmetry.

In Figure 5.5(a), the faces of the signature of Figure 5.4 are drawn on a triangular tessellation of the plane. To build a three-dimensional model of the fullerene, use scissors to cut along the outside boundary and tape the result by making the identifications indicated by the vertex labels. Taking the dual of the graph on the

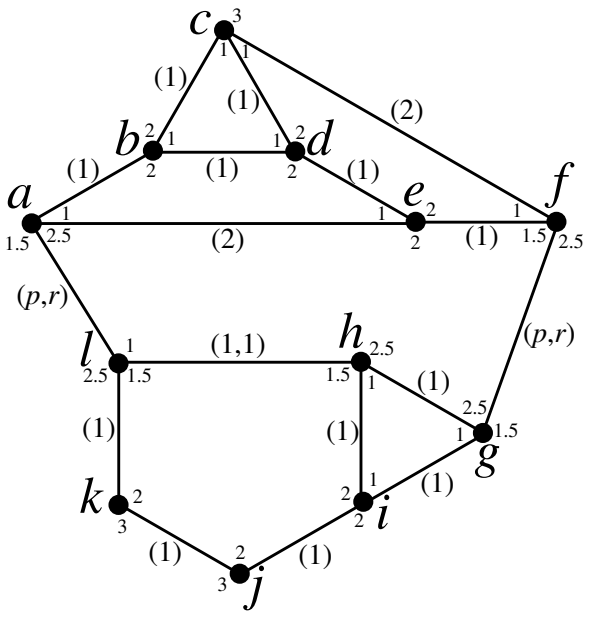
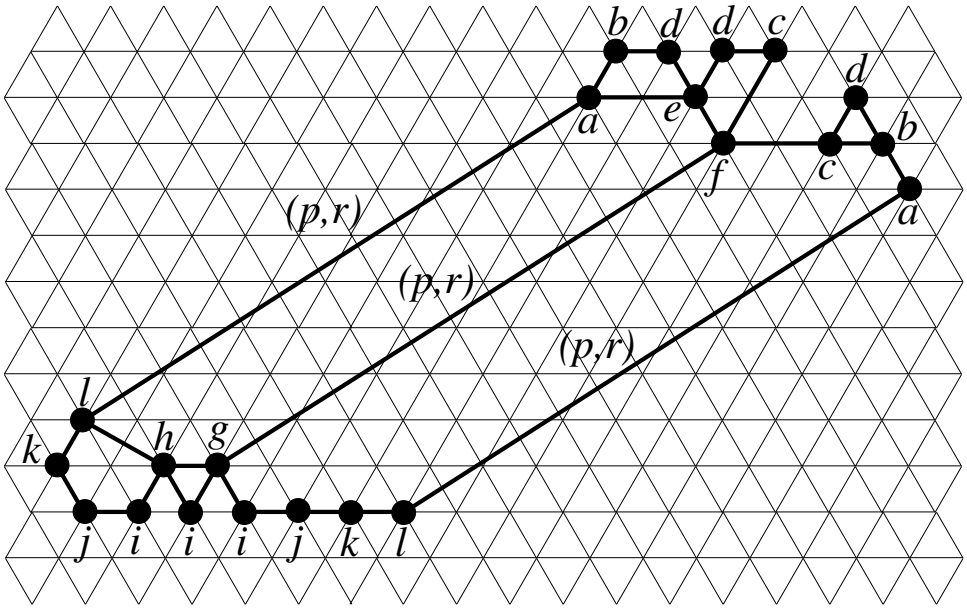



Figure 5.4: A signature-style drawing (see [30]) of a fullerene family that maximizes $\alpha(G)$. The Coxeter coordinates between the pentagons are shown along with the angles between them.

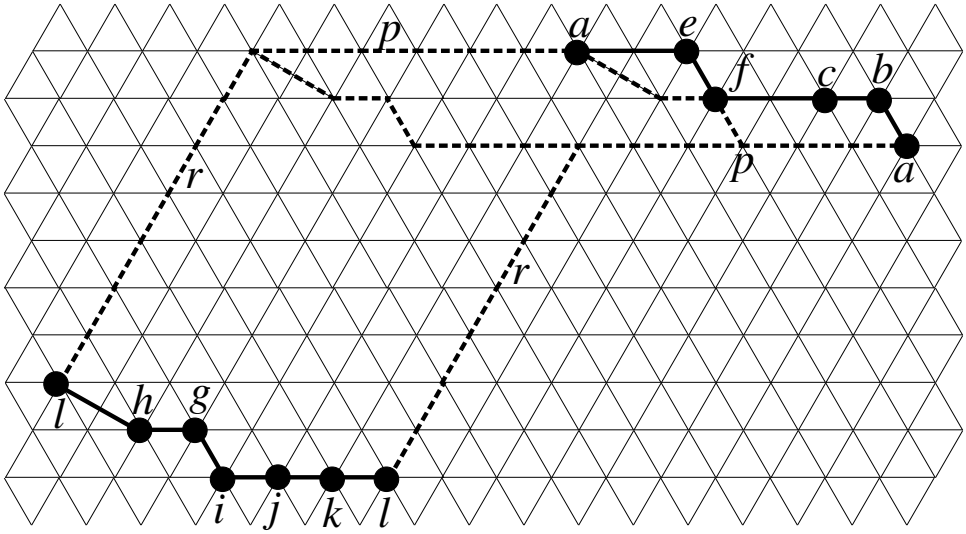
tessellation gives the resulting fullerene.

The fullerene family has so far been described using the parameters $p$ and $r$, but a formula for $n$ can be derived. To find the formula, count the number of hexagons present in the fullerene. This is the number of triangle corners inside the faces of Figure 5.5(a). Figure 5.5(b) shows the larger two faces with their boundaries shifted for no net change in area. From this, it can be seen that the larger two faces contain $7r + 2(p-1)$ hexagons. There are three additional hexagons in the fullerene: one in the signature face $hijkl$, one between $a$ and $e$, and one between $f$ and $c$ (the same three shown in Figure 5.3(a)). Therefore, there are $7r + 2(p-1) + 3$ hexagons, which together with the 12 pentagons make $7r + 2p + 13$ faces, so $n = 22 + 4p + 14r$. No fullerenes exist for $p = r = 0$, however any other non-negative choice of $p$ and $r$ results in a valid construction. When $r = 0$, $n = 22 + 4p$ and when $r = 1$, $n = 36 + 4p$, which together cover every value of $n$ with a fullerene except 20, 24, 28, and 32.

It is possible to determine a face spiral sequence for a fullerene of this family from the knowledge of $p$ or $n$. A fullerene in this family may be thought of as a tube with a patch of six pentagons on each end. There are seven possible alignments of the two patches, with two fullerenes having the same alignment if their $p$ values differ by a multiple of 7. Because of these seven possible alignments, the spiral sequence has seven possibilities. The face spiral sequence is given numerically below followed by




(a) The fullerene drawn on a triangular tessellation




(b) A diagram to help count the number of hexagons

Figure 5.5: A fullerene in the family described by Figure 5.3. Here, $p = 6$ and $r = 7$.

the letters of the pentagons listed in the order they are visited by the sequence. Also included is a list of the five smallest family members for each case.

**Case $\mathbf{p \equiv 0 \pmod 7}$ i.e. $\mathbf{n \equiv 1 \pmod 7}$:** $C_{36}$:10, $C_{50}$:79, $C_{64}$:454, $C_{78}$:1527, $C_{92}$:4857

$$0,\ 1,\ 2,\ 4,\ 6,\ 10,\ \frac{n}{2}-5,\ \frac{n}{2}-4,\ \frac{n}{2}-3,\ \frac{n}{2}-2,\ \frac{n}{2}-1,\ \frac{n}{2}$$

$$d,\ b,\ c,\ e,\ a,\ f,\ l,\ k,\ j,\ i,\ g,\ h$$

**Case $\mathbf{p \equiv 1 \pmod 7}$ i.e. $\mathbf{n \equiv 5 \pmod 7}$:** $C_{26}$:1, $C_{40}$:22, $C_{54}$:125, $C_{68}$:682, $C_{82}$:2150

$$0,\ 1,\ 2,\ 4,\ 6,\ 10,\ \frac{n}{2}-6,\ \frac{n}{2}-5,\ \frac{n}{2}-4,\ \frac{n}{2}-2,\ \frac{n}{2}-1,\ \frac{n}{2}+1$$

$$d,\ b,\ c,\ e,\ a,\ f,\ l,\ k,\ j,\ g,\ h,\ i$$

**Case $\mathbf{p \equiv 2 \pmod 7}$ i.e. $\mathbf{n \equiv 2 \pmod 7}$:** $C_{30}$:3, $C_{44}$:43, $C_{58}$:207, $C_{72}$:975, $C_{86}$:2974

$$0,\ 1,\ 2,\ 4,\ 6,\ 10,\ \frac{n}{2}-7,\ \frac{n}{2}-6,\ \frac{n}{2}-3,\ \frac{n}{2}-2,\ \frac{n}{2},\ \frac{n}{2}+1$$

$$d,\ b,\ c,\ e,\ a,\ f,\ l,\ k,\ g,\ h,\ j,\ i$$

**Case $\mathbf{p \equiv 3 \pmod 7}$ i.e. $\mathbf{n \equiv 6 \pmod 7}$:** $C_{34}$:6, $C_{48}$:69, $C_{62}$:341, $C_{76}$:1389, $C_{90}$:3943

$$0,\ 1,\ 2,\ 4,\ 6,\ 10,\ \frac{n}{2}-8,\ \frac{n}{2}-4,\ \frac{n}{2}-3,\ \frac{n}{2}-1,\ \frac{n}{2},\ \frac{n}{2}+1$$

$$d,\ b,\ c,\ e,\ a,\ f,\ l,\ g,\ h,\ k,\ j,\ i$$

**Case $\mathbf{p \equiv 4 \pmod 7}$ i.e. $\mathbf{n \equiv 3 \pmod 7}$:** $C_{38}$:11, $C_{52}$:115, $C_{66}$:505, $C_{80}$:1975, $C_{94}$:5262

$$0,\ 1,\ 2,\ 4,\ 6,\ 10,\ \frac{n}{2}-5,\ \frac{n}{2}-4,\ \frac{n}{2}-2,\ \frac{n}{2}-1,\ \frac{n}{2},\ \frac{n}{2}+1$$

$$d,\ b,\ c,\ e,\ a,\ f,\ g,\ h,\ l,\ k,\ j,\ i$$

**Case $\mathbf{p \equiv 5 \pmod 7}$ i.e. $\mathbf{n \equiv 0 \pmod 7}$:** $C_{42}$:26, $C_{56}$:194, $C_{70}$:751, $C_{84}$:2684, $C_{98}$:6856

$$0,\ 1,\ 2,\ 4,\ 6,\ 10,\ \frac{n}{2}-6,\ \frac{n}{2}-5,\ \frac{n}{2}-3,\ \frac{n}{2}-2,\ \frac{n}{2}-1,\ \frac{n}{2}$$

$$d,\ b,\ c,\ e,\ a,\ f,\ g,\ h,\ l,\ k,\ j,\ i$$

**Case $\mathbf{p \equiv 6 \pmod 7}$ i.e. $\mathbf{n \equiv 4 \pmod 7}$:** $C_{46}$:48, $C_{60}$:294, $C_{74}$:1080, $C_{88}$:3652, $C_{102}$:8828

$$0,\ 1,\ 2,\ 4,\ 6,\ 10,\ \frac{n}{2}-7,\ \frac{n}{2}-4,\ \frac{n}{2}-3,\ \frac{n}{2}-2,\ \frac{n}{2}-1,\ \frac{n}{2}$$

$$d,\ b,\ c,\ e,\ a,\ f,\ g,\ l,\ k,\ j,\ i,\ h$$

Note that for $n = 26$, 30, and 34 (those with some symmetry), the correct face spiral sequence values are computed by the given formulas, but not in sorted order. For $n \geq 42$, the sequence produced is the lexicographically smallest of all possible face spiral sequences.

Finally, it was observed that for $n = 28$ and 32, the face spiral sequences given by the appropriate cases correctly describe fullerenes $C_{28}$:2 (symmetry-group order 24) and $C_{32}$:4 (order 2), which also achieve the upper bound. There is no corresponding value pair of $p$ and $r$ to generate these fullerenes, so they are technically not included in the family. The result, though, is a remarkable group of seven cases that specify face spiral sequences for an upper bound achieving fullerene at all values of $n$ except 20 and 24 (there are no fullerenes that achieve the bound for $n = 24$).

## 5.4 Symmetrical Upper-Bound Achieving Families

The previous subsection detailed a fullerene family that achieves the upper bound of $n/2 - 2$ for all but five values of $n$, however the fullerenes in this family have no symmetry except for the smallest three members. Symmetrical families with independent set order $n/2 - 2$ can be found by looking in the catalog of highly symmetric fullerenes [29]. This section covers five such families. One of these families has symmetry-group order 20 and two have order 12. Together, these first three families cover 40% of the values of $n$. The last two families of the five have symmetry-group order 24 (the highest possible) and cover $n$ sparsely.

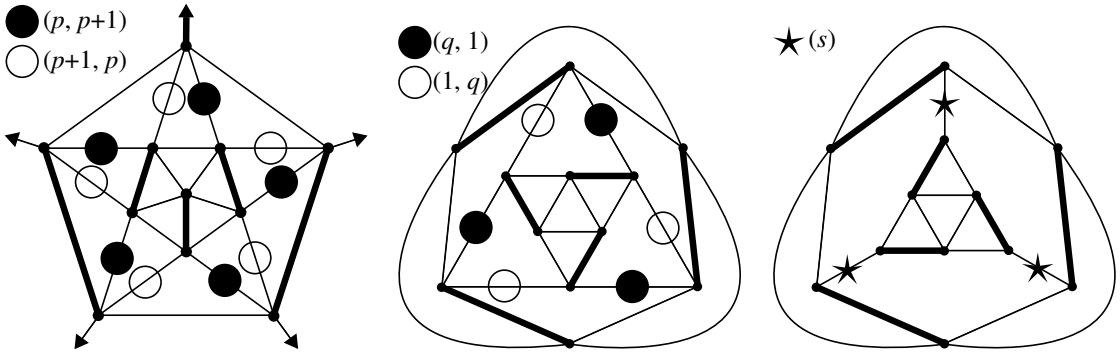
The catalog of all fullerenes with ten or more symmetries [29] may be used to find families with symmetry-group order at least 10 that achieve the upper bound of $n/2 - 2$ by looking at the signature graphs in the catalog. A Coxeter coordinate of $(p)$ corresponds with a clear field of size $(p, 0)$ and a Coxeter coordinate of $(p, q)$ corresponds with a clear field of size $(p, q)$. When reading through the catalog to locate upper bound families, the first requirement is that the pentagons can be paired by six clear fields of size $(1, 0)$. Secondly, the angles must be checked to verify that each of these six clear fields can be colored black to each achieve the minimum penalty of 1.

From analysis of [29], there are three families that attain the upper bound of $n/2 - 2$ and contain fullerenes at equally spaced values of $n$:

1. Family $\mathcal{A}_4$ with $r = 1$ is shown in Figure 5.6(a). This family has symmetry-group order 20. A fullerene exists in this family for all values $p \geq 1$ and has $20p + 20$ vertices.

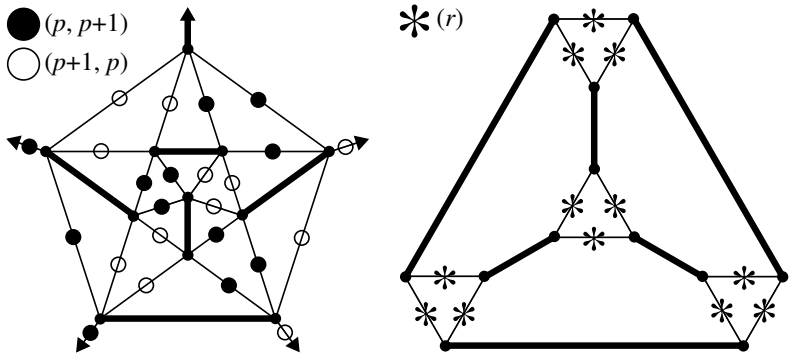
2. Family $\mathcal{R}_6$ with $p = 1$ is shown in Figure 5.6(b). This family has symmetry-group order 12. A fullerene exists in this family for all values $q \geq 1$ and has $12q + 20$ vertices.

3. Family $\mathcal{K}_3$ with $r = 1$ is shown in Figure 5.6(c). This family has symmetry-group




(a) Family $\mathcal{A}_4$ with $r = 1$ (b) Family $\mathcal{R}_6$ with $p = 1$ (c) Family $\mathcal{K}_3$ with $r = 1$

Figure 5.6: Three fullerene families from [29]. The Coxeter coordinate(s) of labeled edges are as stated and of an unlabeled edge is (1). Bold edges represent an optimal pairing of pentagons.




(a) Family $\mathcal{A}_8$ with $r = 1$ (b) Family $\mathcal{C}_1$ with $s = 1$

Figure 5.7: Two fullerene families from [29] with symmetry-group order 24. The Coxeter coordinate(s) of labeled edges are as stated and of an unlabeled edge is (1). Bold edges represent an optimal pairing of pentagons.

order 12. A fullerene exists in this family for all values $s \geq 1$ and has $12s + 14$ vertices.

Other than the fullerene on 20 vertices having symmetry-group order 120, the fullerenes with the highest symmetry-group order that achieve the $n/2 - 2$ upper bound are found in two families with symmetry-group order 24. The first family, shown in Figure 5.7(a), is named $\mathcal{A}_8$ with $r = 1$ in [29]. A fullerene exists in this family for all values $p \geq 1$ and has $24p^2 + 48p + 20$ vertices, the first few values of which are 92, 212, 380, 576, and 840.

The only other family with symmetry-group order 24 is shown in Figure 5.7(b). The family name is $\mathcal{C}_1$ with $s = 1$ in [29]. A fullerene exists in this family for all values $r \geq 1$ and has $8r^2 + 16r + 4$ vertices, the first few values of which are 28, 68, 124, 196,

and 284. A face spiral sequence (not necessarily lexicographically-smallest) for this family is given by:

$$0,\ 1,\ 2r^2+r,\ 2r^2+2r,\ 2r^2+3r+1,\ 2r^2+4r+1,$$

$$2r^2+4r+2,\ 2r^2+5r+2,\ 2r^2+6r+3,\ 2r^2+7r+3,\ 4r^2+8r+2,\ 4r^2+8r+3.$$

## 5.5 Open Problems

The following related problem remains open.

**Formulas for Other Families.** Graver found a formula for $\alpha(G)$ on icosahedral fullerenes [31] from the knowledge of where the optimal clear fields are. One could analyze some families in Graver's catalog of fullerenes with ten or more symmetries [29] to determine if the optimal set of clear fields is completely determined by its catalog entry, regardless of the parameter values. If this is the case, formulas can be found for other families of fullerenes in terms of the parameters that determine the size. When approaching this problem, it would be helpful to determine if the Coxeter coordinates that are listed in a family's signature always include the clear fields that are in the optimal set.

**Fullerenes that Minimize the Independence Number.** Some families described in this chapter consist of fullerenes that maximize the independence number over all fullerenes of the same size. An open problem is to find families that instead minimize the independence number. Attempts were made in this area that included looking at the outliers in Table 3.1 at $n$ = 40, 54, 56, 58, 60, 70, 72, 76, 78, 80, 82, 86, 100, 108, 110, 112, and 114. Finding a family of fullerenes that maximizes the independence number is much easier than a family that minimizes because the upper bound on the independence number is tight whereas the best known lower bound ($3n/8$ from [34]) is far from being tight. It is easy to prove that a fullerene maximizes the independence number by proving it attains the upper bound, whereas a proof for minimization by attaining the lower bound is not possible without improving the bound.

In order for a fullerene to minimize the independence number over all fullerenes of the same size, the clear fields that pair the pentagons must have relatively large penalties, yet still produce the minimum penalty. That is, the distances between the pentagons must be large relative to the other fullerenes with the same number of vertices. Intuitively, this occurs when the pentagons are most uniformly spread out, which is the case for the icosahedral fullerenes. Graver's work on the icosahedral

fullerenes [31] showed that an icosahedral fullerene $F$ with Coxeter coordinates $(p+r,p)$ has $n = 60p^2 + 60pr + 20r^2$ vertices and $\alpha(F) = n/2 - (6p + 2r)$. This is because each clear field that corresponds with the pairing paths has dimensions $(p+r,p)$. To minimize $\alpha(F)$, one seeks the situation $\alpha(F) = n/2 - k$ where $k$ is the largest possible function of $n$. This occurs when $r = 0$ for which $\alpha(F) = n/2 - 3\sqrt{n/15}$. There is strong enough evidence and intuition that the author feels comfortable making the following conjecture:

**Conjecture 5.5.1.** *A family of fullerenes consisting of fullerenes that minimize the independence number over all fullerenes of the same size is the family of icosahedral fullerenes having Coxeter coordinates* $(p,p)$ *(family* $\mathcal{A}_2$ *in [29]). (A fullerene* $F$ *in this family has* $n = 60p^2$ *vertices and* $\alpha(F) = n/2 - 3\sqrt{n/15}$*.)*

**Lower Bound on the Independence Number.** The best known lower bound is $3n/8$, which applies to all triangle-free cubic planar graphs [34]. Based on the discussions of the previous open problem, a conjecture is made that the lower bound found for the icosahedral fullerenes holds for all fullerenes. This conjecture is illustrated by Figure 5.8.

**Conjecture 5.5.2.** *A fullerene* $F$ *satisfies* $\alpha(F) \geq n/2 - 3\sqrt{n/15}$.

Discussions with Daniel Král have led to an interesting approach that may improve the lower bound of $3n/8$ in a minor way. He took the approach of noting that the dual graph is a planar graph and so it can be four-colored [45]. Therefore, there exists a selection of at least $1/4$ of the $n/2+2$ fullerene faces such that no two selected faces are adjacent. For each face in the selection, assign three independent vertices to a vertex set if the face is a hexagon and two vertices if the face is a pentagon. This vertex set is an independent set that differs from $3n/8$ by some small constant. Probabilistic arguments could potentially be used to show that some number of vertices not on one of the selected faces may be added to the set. This may result in a lower bound of the form $3n/8 + k$ for sufficiently large fullerenes, or perhaps improve the ratio to $2n/5$ for sufficiently large fullerenes, where computer search can verify the remainder.

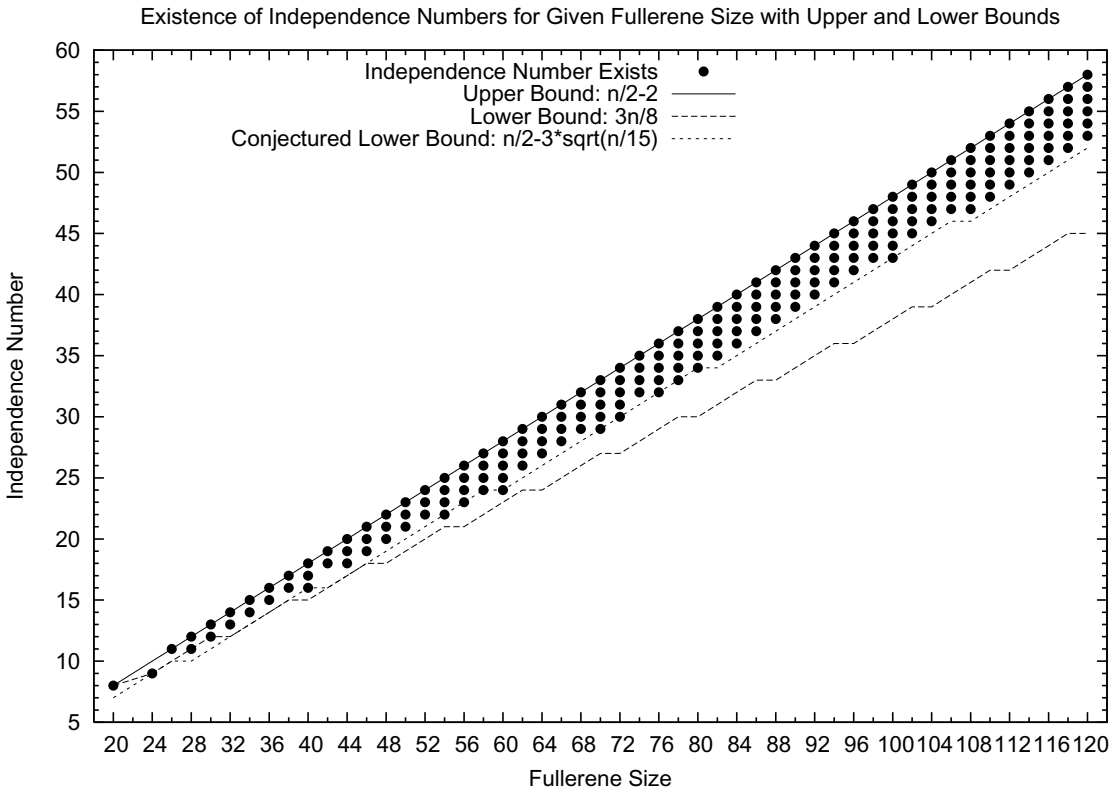



Figure 5.8: Existence of independence numbers for given fullerene size with the best-known theoretical bounds of $n/2 - 2$ and $\lceil 3n/8 \rceil$ and conjectured lower bound $\left\lceil n/2 - 3\sqrt{n/15} \right\rceil$

# Chapter 6
# Closed Shells and Inertia of Unicyclic Graphs

The *inertia* of a graph is an integer triple specifying the numbers of negative, zero, and positive eigenvalues of the adjacency matrix of the graph. A *unicyclic* graph is a simple connected graph with an equal number of vertices and edges. This chapter is adapted from [17] and presents a method of calculating the inertia of a unicyclic graph from its maximum matchings.

Many articles have recently appeared about the eigenvalues of unicyclic graphs in the context of matchings. Topics for such articles include general analysis [15, 39], the spectral radius [11], energy [38, 53], largest eigenvalues [55], and nullity [44]. One paper [32] uses methods similar to those presented here to determine the nullity of a unicyclic graph, but not the number of positive and negative eigenvalues.

According to Hückel theory, the eigenvalues of a *chemical graph* (connected graph with maximum degree at most three) specify the allowed energies of the $\pi$ molecular orbitals available for occupation by electrons. Such a graph or corresponding molecule is said to be (properly) *closed-shell* if exactly half of its eigenvalues are positive (requiring an even number of vertices), which indicates a stable $\pi$-system [25, p. 47]. The study of the inertia of chemical graphs and its relation to predicted stability has been present for at least thirty-five years [33], though the predictors of stability differ slightly from the ones presented here.

## 6.1 Notation

A unicyclic graph contains $n$ vertices and hence $n$ edges. Let $C_q$ be the subgraph induced by the vertices of the unique $q$-cycle of a unicyclic graph. Then $G - V(C_q)$ is the vertex-deleted subgraph created by removing the vertices of $C_q$ and their incident edges.

Let $A$ be the adjacency matrix of $G$. Then the characteristic polynomial of $G$ is $P_G(\lambda) = a_0\lambda^n + a_1\lambda^{n-1} + \cdots + a_n$, the characteristic polynomial of $A$. For any graph, $a_0 = 1$. The multiplicity of $\lambda = 0$ as a root of this polynomial is given by $n - i$ where $i$ is the largest value such that $a_i$ is non-zero.

Let $m(G)$ denote the size of the maximum matching of a graph $G$, counting edges. Hence $m(G) = n/2$ means $G$ has a perfect matching. A matching using $i$ edges

$(0 \le i \le n/2)$ is called an $i$-matching. Let $m_i(G)$ denote the number of $i$-matchings of $G$. By definition, $m_0(G) = 1$ for any $G$.

The inertia of a graph $G$, $\mathrm{In}(G) = (p_-, p_0, p_+)$, is a triple composed of the number of negative, zero, and positive eigenvalues of $A(G)$, respectively. The function $\mathrm{sgn}(x)$ is the standard signum function:

$$\mathrm{sgn}(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$

## 6.2 Inertia

The inertia of a unicyclic graph is completely determined by the size $q$ of its cycle $C_q$, characteristics of the maximum matchings and, in the case $q$ is odd, the size of a maximum matching of $G - V(C_q)$. The details are given in the main result, Theorem 6.2.6, which is proven in this section. The approach for the proof is to determine the sign of a sufficient number of the coefficients $a_i$ in the characteristic polynomial $P_G(\lambda)$. Descartes' sign rule may then be applied to find $p_+$, the number of positive eigenvalues. Along the way, $p_0$ is discovered and hence $p_-$ as well. This work uses two well-known theorems, the latter of which can be found in almost any book on the theory of equations, such as [52, p.124].

**Theorem 6.2.1.** *[16, Theorem 1.3 p.32] Call an "elementary figure"*

*a) the graph $K_2$, or*

*b) a cycle $C_q$ $(q \ge 3)$,*

*and call a "basic figure" $U$ any graph all of whose components are elementary figures. Let $\mathcal{U}_i$ denote the set of all basic figures contained in $G$ having exactly $i$ vertices. Define the "contribution" $b(E)$ of an elementary figure $E$ by*

$$b(K_2) = -1, \qquad b(C_q) = 2(-1)^{q+1}$$

*and a basic figure $U$ by*

$$b(U) = \prod_{E \in U} b(E).$$

*Then for $i > 0$,*

$$a_i = (-1)^i \sum_{U \in \mathcal{U}_i} b(U).$$

**Theorem 6.2.2** (Descartes' Sign Rule)**.** *The number of positive roots of a polynomial* $f(x) = f_0x^n + f_1x^{n-1} + \cdots + f_n$ *with all real roots is equal to the number of sign changes of* $f_i$ *proceeding from* $f_0$ *to* $f_n$*, ignoring* $f_i = 0$*.*

Descartes' sign rule may be applied to $P_G(\lambda)$ because $A$ has all real eigenvalues since $A$ is real and symmetric. The proof of the main result begins by finding the sign of the coefficients of $P_G(\lambda)$ with even index. Note that here and henceforth an edge "incident to the cycle" refers to such an edge that is not on the cycle.

**Lemma 6.2.3.** *If* $G$ *is a unicyclic graph with cycle* $C_q$*, then for* $0 \le i \le \lfloor n/2 \rfloor$*,*

$$\operatorname{sgn}(a_{2i}) = \begin{cases} (-1)^i & \text{if } q \equiv 0 \pmod 4,\ 0 < i \le m(G), \text{ and there exists an} \\ & i\text{-matching containing an edge incident to the cycle,} \\ (-1)^i & \text{if } q \equiv 0 \pmod 4 \text{ and } i = 0, \\ (-1)^i & \text{if } q \equiv 2 \pmod 4 \text{ and } i \le m(G), \\ (-1)^i & \text{if } q \text{ is odd and } i \le m(G), \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

*Proof.* Trivially $a_0 = 1$, which is equal to the stated result $(-1)^i$ for any value of $q$ since $m(G) \ge 0$.

Assume $q$ is odd. Then any basic figure on an even number of vertices consists only of copies of $K_2$. Such a matching of $2i$ vertices exists only if $i \le m(G)$. Therefore, for $i > m(G)$, no basic figure exists and so $\operatorname{sgn}(a_{2i}) = 0$. Otherwise, each basic figure contributes $(-1)^i$ to the sum for $a_{2i}$, so $\operatorname{sgn}(a_{2i}) = (-1)^{2i}(-1)^i = (-1)^i$.

Assume $q$ is even and $i \ge 1$. Since any even cycle can be decomposed into a matching, $\operatorname{sgn}(a_{2i}) = 0$ when $i > m(G)$. Suppose $2i < q$, then any basic figure on $2i$ vertices only contains copies of $K_2$ and $\operatorname{sgn}(a_{2i}) = (-1)^i$, as before. Finally, suppose $q \le 2i \le 2m(G)$ so that some basic figures on $2i$ vertices contain $C_q$ and $i - q/2$ copies of $K_2$ and some basic figures contain only $i$ copies of $K_2$. Thus,

$$\begin{aligned} a_{2i} &= (-1)^{2i}\left(m_{i-q/2}(G - V(C_q))\left[2(-1)^{q+1}(-1)^{i-q/2}\right] + m_i(G)\left[(-1)^i\right]\right) \\ &= (-1)^i\left(2m_{i-q/2}(G - V(C_q))(-1)^{q/2+1} + m_i(G)\right). \end{aligned}$$

When $q \equiv 2 \pmod 4$, this simplifies to show $\operatorname{sgn}(a_{2i}) = (-1)^i$. When $q \equiv 0 \pmod 4$, this implies that $\operatorname{sgn}(a_{2i}) = (-1)^i \operatorname{sgn}(m_i(G) - 2m_{i-q/2}(G - V(C_q)))$. But $m_i(G) \ge 2m_{i-q/2}(G - V(C_q))$ since $2m_{i-q/2}(G - V(C_q))$ matchings of $G$ of size $i$ can be found by using the two matching in the cycle. Furthermore, $m_i(G) > 2m_{i-q/2}(G - V(C_q))$

only when there exists a matching of $G$ of size $i$ that uses an edge between $C_q$ and $G - V(C_q)$. □

Lemma 6.2.3 is now simplified by determining when it is possible to have $q \equiv 0 \pmod 4$, $i \leq m(G)$, and the existence of an $i$-matching containing an edge between $C_q$ and $G - V(C_q)$.

**Lemma 6.2.4.** *For $G$ a unicyclic graph with cycle $C_q$, let*

$$k = \begin{cases} m(G) - 1 & \text{if } q \equiv 0 \pmod 4 \text{ and no maximum matching} \\ & \text{contains an edge incident to the cycle, and} \\ m(G) & \text{otherwise.} \end{cases}$$

*Then for $0 \leq i \leq \lfloor n/2 \rfloor$,*

$$\operatorname{sgn}(a_{2i}) = \begin{cases} (-1)^i & \text{if } i \leq k, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

*Proof.* The statement follows immediately from Lemma 6.2.3 except in the case where $q \equiv 0 \pmod 4$ and $0 < i \leq m(G)$. For the remainder of the proof, these two conditions are assumed and used to prove that there exists an $m(G) - 1$ matching containing an edge incident to the cycle.

Notice that if an $i$-matching exists containing an edge incident to the cycle for $i > 1$ then such an $(i-1)$-matching also exists. Likewise, if no $i$-matching exists containing an edge incident to the cycle then no $(i+1)$-matching exists containing such an edge. Thus, there is some maximum value $k$ such that there exists a $k$-matching containing an edge incident to the cycle, but no such $(k+1)$-matching exists. Clearly, $k \leq m(G)$. Also, $k = 0$ if and only if $G = C_q$.

Suppose $0 < k < m(G)$. That is, no maximum matching contains an edge incident to the cycle. Consider a maximum matching $M$ and an edge $e = (u, v)$ incident to the cycle such that $u \in V(C_q)$ and $v \notin V(C_q)$. Hence, for all such choices of $M$ and $e$, $e \notin M$. Furthermore, since $M$ is maximum and $q$ is even, every vertex on the cycle is incident to a matched edge on the cycle, as shown in Figure 6.1. Let $f$ be the matched cycle edge incident to $u$. There also must exist an edge $g \in M$ incident to $v$, otherwise the matching $M - f + e$ would be a maximum matching that contradicts $k \neq m(G)$. Note that $M - f - g + e$ is a matching of size $m(G) - 1$ containing an edge incident to the cycle and hence $k = m(G) - 1$.

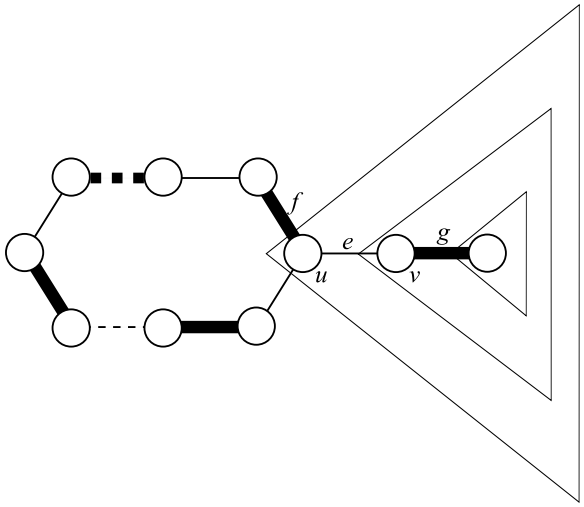



Figure 6.1: A case considered in the proof of Lemma 6.2.4. Bold edges are in the maximum matching $M$. Triangles represent unknown tree-like parts of the graph. Tree-like additions are not restricted to $u$ and may be attached to any other vertex on the cycle.

Finally, suppose $k = 0$ and hence $G = C_q$. The argument in the proof of Lemma 6.2.3 shows that $a_{2i} = (-1)^i$ for $2i < q$. When $2i = q$, there are exactly three basic figures: two perfect matchings and one containing only $C_q$. Direct application of Theorem 6.2.1 shows $a_q = 0$. □

Theorem 6.2.2 is applied to count the number of positive eigenvalues of $A$ by counting the number of sign changes from $a_0$ to $a_n$. Lemma 6.2.4 shows that the coefficients $a_{2i}$ begin at $a_0 = 1$ and alternate in sign, ending at a point where the remaining coefficients are all 0. Let $k$ be the maximum value such that $a_{2k} \neq 0$. Then it does not matter what the values of $a_{2i+1}$ are for $0 \leq i < k$, as they do not affect the number of sign changes since the sign of $a_{2i+1}$ is either the same as $a_{2i}$ or $a_{2i+2}$, or $a_{2i+1} = 0$. Thus, only consider the odd coefficients $a_{2i+1}$ when $i \geq k$. The odd coefficients are characterized now.

**Lemma 6.2.5.** *Let $G$ be a unicyclic graph with cycle $C_q$. Let $k$ be the maximum value such that $a_{2k} \neq 0$. Then $a_{2i+1} = 0$ for all $i > k$.*

*Proof.* This is trivial in the case where $q$ is even, because there exist no basic figures on an odd number of vertices. Assume $q$ is odd. If there exists a basic figure on $2i+1$ vertices, then there exists a basic figure on $2i$ vertices, which is found by replacing $C_q$ (which must be included) in the basic figure with $(q-1)/2$ copies of $K_2$ to get a matching of size $i$. Thus, $a_{2i+1} = 0$ for $i > k$. □

Our search for the number of sign changes has now been reduced to finding the sign of $a_{2k+1}$. All the odd coefficients are 0 when $q$ is even. The only case to consider is $q$ odd, and recall that $k$ has been defined in this case to equal $m(G)$. A basic figure on $2m(G)+1$ vertices must contain $C_q$ and $(2m(G)+1-q)/2$ copies of $K_2$ from $G-V(C_q)$. Therefore,

$$\begin{aligned} a_{2m(G)+1} &= (-1)^{2m(G)+1}\left(m_{m(G)-(q-1)/2}(G-V(C_q))\right)\left(2(-1)^{q+1}(-1)^{m(G)-(q-1)/2}\right) \\ &= -2\left(m_{m(G)-(q-1)/2}(G-V(C_q))\right)\left((-1)^{m(G)-(q-1)/2}\right). \end{aligned}$$

So, $\operatorname{sgn}(a_{2m(G)+1}) =$

$$\begin{cases} (-1)^{m(G)-(q-1)/2+1} & \text{if } q \text{ is odd and } m_{m(G)-(q-1)/2}(G-V(C_q)) > 0, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, $\operatorname{sgn}(a_{2m(G)+1})$ is the opposite of $\operatorname{sgn}(a_{2m(G)}) = (-1)^{m(G)}$ when $q \equiv 1 \pmod 4$ and $m_{m(G)-(q-1)/2}(G-V(C_q)) > 0$. They have the same sign when $q \equiv 3 \pmod 4$ and $m_{m(G)-(q-1)/2}(G-V(C_q)) > 0$. The requirement $m_{m(G)-(q-1)/2}(G-V(C_q)) > 0$ means there exists a maximum matching of $G$ that does not use any edge between $C_q$ and $G-V(C_q)$. This is equivalent to $2m(G)+1 = 2m(G-V(C_q))+q$.

The number of positive eigenvalues can now be determined by counting the number of sign changes of the $a_i$'s. The number of zero eigenvalues is $n-i$ where $i$ is the largest value such that $a_i \neq 0$. The number of negative eigenvalues, and hence the inertia of a unicyclic graph, can then be computed by considering the size of a maximum matching of $G$ and, if necessary, $G-V(C_q)$. The following is proved from the above analysis.

**Theorem 6.2.6.** *For $G$, a unicyclic graph containing the cycle $C_q$,*
$\operatorname{In}(G) = (p_-, p_0, p_+) =$

$$\begin{cases} \textbf{(a)}\ (m(G)-1, n-2m(G)+2, m(G)-1) & \text{if } q \equiv 0 \pmod 4 \text{ and no} \\ & \text{maximum matching contains} \\ & \text{an edge incident to } C_q, \\ \textbf{(b)}\ (m(G), n-2m(G)-1, m(G)+1) & \text{if } q \equiv 1 \pmod 4 \text{ and} \\ & 2m(G)+1=2m(G-V(C_q))+q, \\ \textbf{(c)}\ (m(G)+1, n-2m(G)-1, m(G)) & \text{if } q \equiv 3 \pmod 4 \text{ and} \\ & 2m(G)+1=2m(G-V(C_q))+q, \text{ and} \\ \textbf{(d)}\ (m(G), n-2m(G), m(G)) & \text{otherwise.} \end{cases}$$

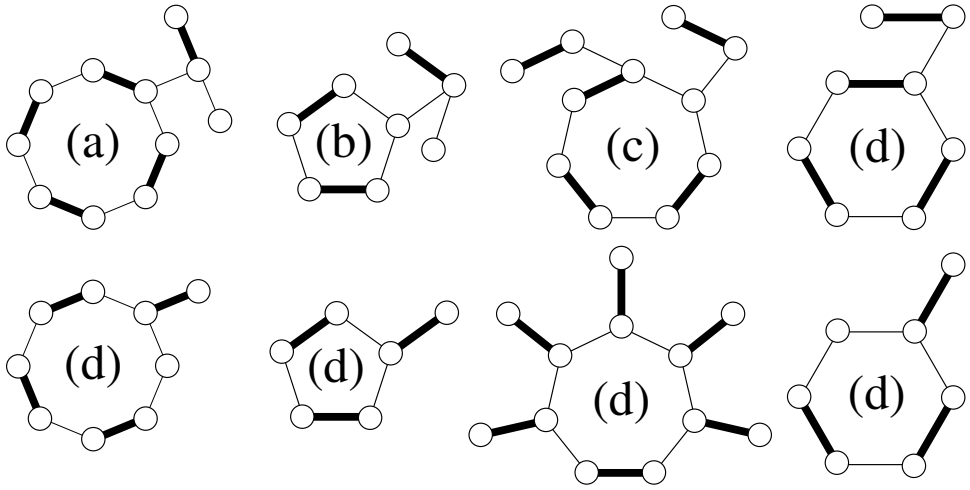



Figure 6.2: Some unicyclic graphs, each labeled with the applicable case from Theorem 6.2.6 and an illustrative maximum matching. The inertia for each graph in clockwise order from the top left is: $(4,3,4)$, $(3,1,4)$, $(6,0,5)$, $(4,0,4)$, $(3,1,3)$, $(6,0,6)$, $(3,0,3)$, $(4,1,4)$.

Example graphs that illustrate the cases of Theorem 6.2.6 are shown in Figure 6.2. Where multiple maximum matchings exist, a maximum matching is shown that illustrates the applicable case. The top row shows examples with a maximum matching that does not contain an edge incident to $C_q$ and the bottom row shows examples with a maximum matching that does contain such an edge.

## 6.3 Inertia Algorithm

It is possible to compute the inertia of a unicyclic graph in linear time using a simple variation of the Karp-Sipser leaf-removal algorithm [36]. The first phase of the algorithm repeatedly removes a leaf, its neighbor, and any incident edges from the graph. The edge incident to the leaf is added to the matching. It was shown that this phase of the algorithm makes no "mistakes." That is, a maximum matching of the remaining graph, the *core*, produces a maximum matching of the original graph when combined with the edges selected by the leaf-removal phase. The next phase of the algorithm involves selecting a random edge when no leaf exists then continuing again with leaf-removal. However, this is unnecessary for unicyclic graphs as discussed below.

In the case of unicyclic graphs, after leaf-removal one is left with a core of either (1) the null graph, (2) isolated vertices, or (3) the cycle and possibly some isolated vertices. In case (3), a maximum matching contains $\lfloor q/2 \rfloor$ more edges than were selected by the leaf-removal. To find the inertia when $q$ is odd, the size of a maximum

matching of the forest $G - V(C_q)$ can also be determined by the Karp-Sipser algorithm. In this case, it is easy to locate and remove $C_q$ from $G$ in linear time using a breadth-first search.

When $q \equiv 0 \pmod 4$, the algorithm can be used to determine if no maximum matching contains an edge incident to the cycle, since this occurs exactly when the algorithm results in case (3) above. The reason is as follows. Let $V(C_q) = \{v_1, v_2, \ldots v_q\}$, and let $T_i$ be the maximal-sized tree rooted at $v_i$ using no edges of the cycle. Finally, let $t_i$ be the size of a maximum matching of $T_i$. Because the leaf removal is mistake-free, the algorithm accurately finds each $t_i$, regardless of the order in which the leaves are removed. Assuming that the algorithm found a maximum matching of $G$ containing no edges incident to the cycle, it resulted in case (3). In that case, the size of a maximum matching of $G$ is $t_1 + t_2 + \cdots + t_q + q/2$. If a different maximum matching of $G$ (one not found by the algorithm) exists using an edge incident to the cycle, there cannot be $q/2$ cycle edges in the matching and so such a matching can have at most $t_1 + t_2 + \cdots + t_q + q/2 - 1$ edges, and therefore cannot also be maximum. Thus, when the algorithm results in case (3), no maximum matching exists containing an edge incident to the cycle.

The maximum matchings shown in Figure 6.2 are examples of those that may be found by the inertia algorithm. Other matchings may be found, depending on the order in which leaves are selected for removal and the chosen matching of the remaining cycle in the event of case (3).

## 6.4 Closed Shells

A graph $G$ is called *properly closed-shell* if $n$ is even and exactly half of its eigenvalues are positive because the eigenvalues of a graph correspond directly to the molecular-orbital energy levels in the simplified model of electronic structure given by Hückel theory [50]. The following formal definition refers to a *properly closed shell* [25, p. 47], which is shortened to *closed shell* throughout this dissertation.

**Definition 6.4.1** (Closed Shell)**.** *A graph $G = (V, E)$ is said to be* closed shell *if*

*(i) $n$ is even and*

*(ii) exactly half of the eigenvalues of the adjacency matrix of $G$ are positive.*

It is easy to determine whether or not a unicyclic graph is closed-shell if the inertia is already known. However, if one is only concerned with whether or not a unicyclic graph is closed-shell and does not need to know the inertia, such a conclusion can be reached using less information than is required by Theorem 6.2.6.

**Corollary 6.4.2.** *Let $G$ be a unicyclic graph with cycle $C_q$. Then $G$ is closed-shell if and only if it has an even number of vertices and one of the following mutually-exclusive cases holds:*

**Case 1.** $q \equiv 0 \pmod 4$ *and $G$ has a unique perfect matching,*

**Case 2.** $q \equiv 1 \pmod 4$ *and either*

**2.1.** $2m(G) - 2m(G - V(C_q)) = q - 1$ *and* $m(G) = n/2 - 1$, *or*

**2.2.** $2m(G) - 2m(G - V(C_q)) \neq q - 1$ *and $G$ has a perfect matching,*

**Case 3.** $q \equiv 2 \pmod 4$ *and $G$ has a perfect matching, or*

**Case 4.** $q \equiv 3 \pmod 4$ *and $G$ has a perfect matching.*

*Proof.* Recall that a graph is closed-shell if and only if $p_+ = n/2$. For each value of $q \pmod 4$, determine the conditions in which this holds true by considering the four cases of Theorem 6.2.6.

**Case 1:** $q \equiv 0 \pmod 4$

Either case (a) or (d) applies from Theorem 6.2.6. A graph $G$ cannot be closed-shell if case (a) applies since $p_+ = m(G) - 1 < n/2$. If case (d) applies, then $p_+ = m(G)$ so $G$ is closed-shell if and only if it has at least one perfect matching. Case (d) applies only when some maximum matching contains an edge incident to $C_q$. Note that a unicyclic graph has at most two perfect matchings, which follows from the algorithm for calculating the inertia since only isolated vertices and the cycle may remain after leaf-removal. Furthermore, a perfect matching is unique if and only if it contains an edge incident to $C_q$ (such that the cycle does not remain after leaf-removal). Therefore $G$ is closed-shell if and only if it has a unique perfect matching.

**Case 2:** $q \equiv 1 \pmod 4$

Either case (b) or (d) applies, so $G$ is closed-shell if and only if it has a perfect matching (case (d)), unless $2m(G) + 1 = 2m(G - V(C_q)) + q$ (case (b)), in which case the requirement is that $m(G) + 1 = n/2$.

**Case 3:** $q \equiv 2 \pmod 4$

Only case (d) applies, so $G$ is closed-shell if and only if it has at least one perfect matching.

**Case 4:** $q \equiv 3 \pmod 4$

Either case (c) or (d) applies and $p_+ = m(G)$ in each case so $G$ is closed-shell if and only if it has at least one perfect matching. □

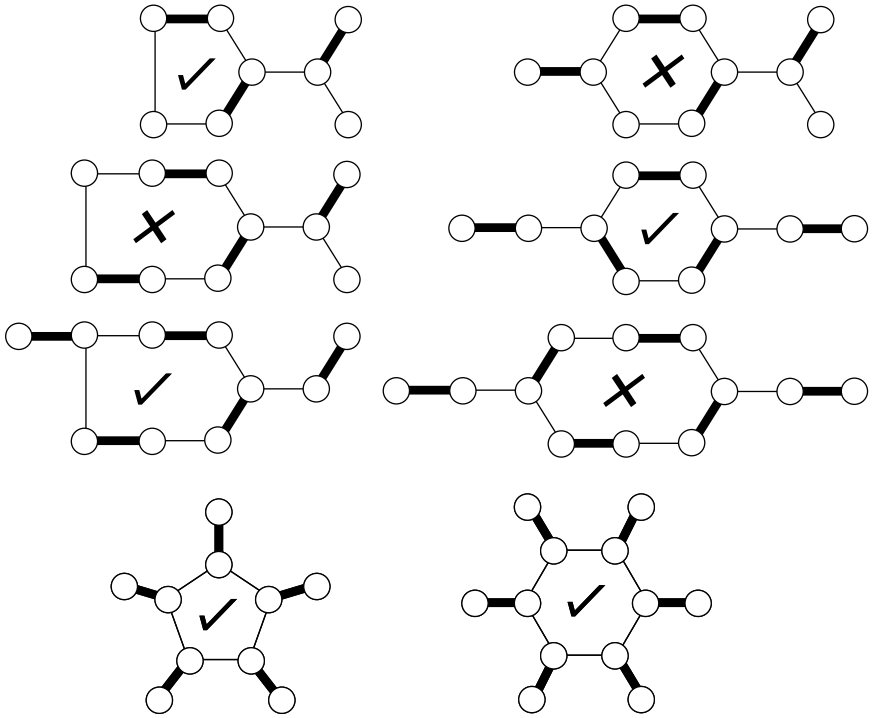


Figure 6.3: Some unicyclic graphs, with a maximum matching of each shown in bold. Examples that are closed-shell are checked.

Examples of unicyclic graphs that are or are not closed-shell are given in Figure 6.3. This includes some graphs that are not closed-shell yet contain a perfect matching and some graphs that are closed-shell yet have no perfect matching.

Clearly, determining if a unicyclic graph is closed-shell can also be computed in linear time. In many cases, this calculation can be performed slightly faster than it was for the general inertia. Repeated leaf removal can be used to determine the size of a maximum matching as well as the number of perfect matchings. When $q \equiv 1 \pmod 4$ and $m(G) \geq n/2 - 1$, the size of a maximum matching of $G - V(C_q)$ must be determined to apply Corollary 6.4.2. This can be computed by running the Karp-Sipser algorithm on the forest $G - V(C_q)$.

Two early-termination conditions can be applied to the leaf removal algorithm because only perfect or near-perfect matchings are allowed for closed-shell unicyclic graphs. When $q \not\equiv 1 \pmod 4$, a perfect matching is required and the algorithm can terminate with the answer "not closed-shell" when an isolated vertex appears. When $q \equiv 1 \pmod 4$, the algorithm can terminate with the response "not closed-shell" when three isolated vertices appear, which implies $m(G) < n/2 - 1$.

Chapter 7 introduces the concept of a closed-shell independent set for fullerenes. There, an algorithm is presented that searches for a maximum closed-shell indepen-

dent set. This linear-time algorithm for unicyclic graphs is used as a subroutine in order to increase the speed of the search.

# Chapter 7
# Closed-Shell Independent Sets

Observations in [24, 27] found that addends bonded to a maximum independent set of carbon atoms did not always produce a stable configuration; only one of the 1085 non-isomorphic maximum independent sets on $C_{60}$:1812 results in a stable molecular configuration. Beyond the requirement that bulky addends bond to carbon atoms that represent an independent set, chemical theory imposes a further restriction in that the set of carbon atoms that do not bear an addend constitute a (possibly disconnected) unsaturated system and should have a stable distribution of $\pi$ electrons. This amounts to the requirement that every component of the graph induced by the complement of the independent set must have a closed-shell, i.e., an adjacency matrix in which exactly half of the eigenvalues are positive. An independent set with this additional property in the complement is called a *closed-shell independent set* and has been studied in the context of fullerenes [14, 24, 26, 27, 43]. The cardinality of such a set is the *closed-shell independence number* of the fullerene $F$ and is denoted by $\alpha^-(F)$.

There is evidence that the closed-shell restriction may be more important than the non-adjacency restriction, especially in the cases where there is a small number of addends. Experimental results [1, 3, 4, 5] noted in [24, 47] show some cases on fullerenes with 60 and 70 vertices that exhibit adjacent addend locations for small numbers of addends. As noted in [24], it is significant that as more addends bond to the fullerenes, non-adjacency is again observed. Thus, the model presented here seems reasonable when considering the maximum number of bulky addends.

At least one other model [2] has been proposed for determining which addend configurations result in a stable molecule, though it was not applied to bulky addends. This other model uses the simpler valence-bond view (Kekulé structure) that the graph induced by the vertices not bonding to an addend must have a perfect matching. This ensures that the electrons of each atom are accounted for by localized single and double bonds. This is in contrast to the more directly quantum-mechanical molecular orbital view (Hückel theory) that considers the electrons to occupy delocalized orbitals, each having a defined energy and (potentially, at least) spreading over the whole molecule. The requirement for stability is then that the occupation should lead to a closed shell. It is generally accepted in theoretical chemistry that

while the simpler valence-bond view is useful for some situations, it does not provide as accurate of a description as the molecular orbital view, and can lead to incorrect results if applied in its simplest form [10, 9.7]. For example, the bare fullerene $C_{20}$ has a plethora of perfect matchings yet it is not closed-shell, whereas the molecular orbital view correctly identifies the molecule as unstable. Chapter 6 illustrated the similarities between the perfect matchings and the closed-shell properties, but it was seen that having a perfect matching is neither a necessary nor sufficient condition to be closed-shell.

Before this work, the closed-shell independence number was known for only seven fullerenes: $C_{60}$:1812, $C_{70}$:8149, $C_{76}$:19151, $C_{84}$:51590 and $C_{84}$:51591 from [27], and $C_{20}$:1 and $C_{80}$:31924 from [26]. Crane also independently computed $\alpha^-(F)$ for $C_{70}$:8149 [14]. It is clear that the closed-shell independence number is at most the independence number, since the former is a restricted version of the latter. Of these seven fullerenes, only $C_{20}$:1 and $C_{60}$:1812 had $\alpha(F) = \alpha^-(F)$ [27, 26]. A theoretical study of the value $\alpha(F) - \alpha^-(F)$ for the icosahedral fullerenes was given in [26]. The problem of finding a reasonable algorithm to compute the closed-shell independence number on general fullerenes remained open.

This chapter includes a description of a backtracking algorithm that was used to calculate the closed-shell independence number of all fullerenes with 100 or fewer atoms. The algorithm seeks early detection of the nonexistence of a closed-shell independent set of a given order or larger. This is done using a combination of dynamic upper bounds, prioritized vertex selection, and the identification of special cases to avoid lengthy eigenvalue calculations when possible.

This chapter is organized as follows. After formalizing the problem ( 7.1), a new upper bound is given on the closed-shell independence number ( 7.2). Then the algorithm description begins with a discussion of the high-level pseudo-code ( 7.3). Next, specially designed data structures used to implement the algorithm are described ( 7.4) as well as some additional approaches to reduce the run time ( 7.5). Some observations are made on the computational results ( 7.6) and the results are compared with previous calculations of the independence number ( 7.7). Finally, ideas for further improvement and research are discussed ( 7.8). Sections 7.3 through 7.7 can largely be found in [18].

## 7.1 Definitions and Notation

A closed-shell graph was defined formally by Definition 6.4.1. Adding to the independent set requirement the requirement that the graph induced by the vertices not in the

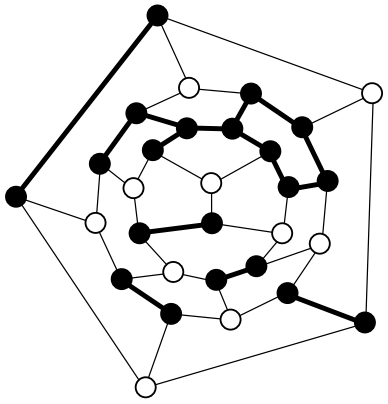


Figure 7.1: A maximum closed-shell independent set on $C_{30}$:2 in white

independent set be a closed-shell graph gives the model of a closed-shell independent set.

**Definition 7.1.1** (Closed-Shell Independent Set)**.** *On a graph $F = (V, E)$, a vertex subset $S \subseteq V$ is called a* closed-shell independent set *if*

*(i)* $\forall u, v \in S$, $(u, v) \notin E$ *(independence) and*

*(ii) each component of $F - S$ is closed shell.*

The largest such set is useful for studying fullerene reactivity. The following definition gives a corresponding invariant for a graph.

**Definition 7.1.2** (Closed-Shell Independence Number)**.** *The* closed-shell independence number *of a graph $F$ is:*

$$\alpha^{-}(F) = \max_{S \subseteq V} |S|$$

*where $S$ is a closed-shell independent set.*

Figure 7.1 gives an example of a maximum closed-shell independent set. The vertices not in the set form six components, five of which are copies of $P_2$ (the path on two vertices) and the last is a unicyclic graph.

The algorithm to be presented relies on two important properties of closed-shell independent sets. The first property follows from the facts that every fullerene has an even number of vertices and that for a closed-shell independent set $S$, each component of $F - S$ has an even number of vertices.

**Property 7.1.3.** *Every closed-shell independent set of a fullerene has an even number of vertices.*

The next property follows from the previous one and the theorem in [27, eqn. (7)] that states $\alpha^-(F) \leq 2n/5$, a bound that is obtained if and only if every component of the graph induced by the vertices not in the set is a copy of $P_2$ (the path on two vertices).

**Property 7.1.4.** *For a fullerene,* $\alpha^-(F) \leq 2\lfloor n/5 \rfloor$.

Note that $2\lfloor n/5 \rfloor$ is the largest even integer less than or equal to $2n/5$.

## 7.2 A New Upper Bound

This section proves a new upper bound for $\alpha^-(F)$ for fullerenes that improves on the bound given in Property 7.1.4 for $n \geq 126$. In doing so, an upper bound on the number of hexagons contributing three vertices to a closed-shell independent set is also given.

Given a closed-shell independent set, let $\mathcal{F}_i$ be the set of faces with $i$ vertices in set and let $F_i$ be the number of faces in $\mathcal{F}_i$. There are $\frac{n}{2} + 2$ faces in a fullerene, so $F_0 + F_1 + F_2 + F_3 = \frac{n}{2} + 2$. Clearly $\mathcal{F}_3$ contains only hexagons.

**Lemma 7.2.1.** *No two faces in* $\mathcal{F}_3$ *are adjacent.*

*Proof.* If two faces in $\mathcal{F}_3$ are adjacent then the vertex that is both not in the set and is incident with the shared edge must be an isolated vertex, which violates the closed-shell independent set properties. □

**Lemma 7.2.2.** *Let* $f$ *be a face in* $\mathcal{F}_i$ *for some* $i \leq 2$. *Then* $f$ *is adjacent to at most* $i$ *faces in* $\mathcal{F}_3$.

*Proof.* Each edge of a face in $\mathcal{F}_3$ is incident with one vertex in the set and one not in the set. By Lemma 7.2.1, each vertex of $f$ that is in the set corresponds with at most one face in $\mathcal{F}_3$. □

**Theorem 7.2.3.** *The* $F_i$ *values satisfy the inequality* $F_3 \leq (n+4)/8 - F_1/8 - F_0/4$. *Equality is obtained if and only if each face in* $\mathcal{F}_2$ *is adjacent to two faces in* $\mathcal{F}_3$ *and each face in* $\mathcal{F}_1$ *is adjacent to one face in* $\mathcal{F}_3$.

*Proof.* Let $a_{i,j}$ be the number of edges such that one of the two faces containing the edge is in $\mathcal{F}_i$ and the other is in $\mathcal{F}_j$ where $0 \leq j \leq i \leq 3$. Then

$$6F_3 = 2a_{3,3} + a_{3,2} + a_{3,1} + a_{3,0}$$

because the left side counts each edge once per hexagon in $\mathcal{F}_3$ and the right side counts the number of times an edge lies on a face in $\mathcal{F}_3$.

From Lemma 7.2.1, $a_{3,3} = 0$ and from Lemma 7.2.2, $a_{3,2} \leq 2F_2$ (each edge on a face in $\mathcal{F}_2$ is also on at most two faces in $\mathcal{F}_3$), $a_{3,1} \leq F_1$, and $a_{3,0} = 0$. This gives the following inequality.

$$\begin{aligned}
6F_3 &\leq 2F_2 + F_1 \\
&= 2(\frac{n}{2} + 2 - F_3 - F_1 - F_0) + F_1 \qquad (F_3 + F_2 + F_1 + F_0 = \frac{n}{2} + 2) \\
&= n + 4 - 2F_3 - F_1 - 2F_0 \\
F_3 &\leq (n+4)/8 - F_1/8 - F_0/4 \\
&\leq (n+4)/8 \qquad \square
\end{aligned}$$

There are $(n+4)/2$ faces in a fullerene, so at most $1/4$ of them are in $\mathcal{F}_3$.

**Theorem 7.2.4.** *An upper bound on the closed-shell independence number is*

$$\alpha^-(F) \leq 3n/8 + 3/2 - 3F_1/8 - 3F_0/4$$

*with equality if and only if equality holds in Theorem 7.2.3.*

*Proof.* Counting the vertices on each face counts each vertex three times because each vertex appears on three faces, which gives the following equation.

$$\begin{aligned}
\alpha^-(F) &= (3F_3 + 2F_2 + F_1 + 0F_0)/3 \\
&= F_3 + \frac{2}{3}F_2 + \frac{1}{3}F_1 \\
&= F_3 + \frac{2}{3}(\frac{n}{2} + 2 - F_3 - F_1 - F_0) + \frac{1}{3}F_1 \qquad (F_3 + F_2 + F_1 + F_0 = \frac{n}{2} + 2) \\
&= \frac{n+4}{3} + \frac{1}{3}F_3 - \frac{1}{3}F_1 - \frac{2}{3}F_0 \qquad (7.1) \\
&\leq \frac{n+4}{3} + \frac{1}{3}\left(\frac{n+4}{8} - \frac{1}{8}F_1 - \frac{1}{4}F_0\right) - \frac{1}{3}F_1 - \frac{2}{3}F_0 \qquad \text{(Theorem 7.2.3)} \\
&= \frac{3}{8}n + \frac{3}{2} - \frac{3}{8}F_1 - \frac{3}{4}F_0 \qquad \square
\end{aligned}$$

The following simplifies this bound to create one that is only dependent on $n$.

**Corollary 7.2.5.** *An upper bound on the closed-shell independence number is*

$$\alpha^-(F) \leq 3n/8 + 3/2$$

*with equality if and only if $F_0 = 0$, $F_1 = 0$, $F_2 = 3(n+4)/8$, $F_3 = (n+4)/8$.*

Requiring that $\alpha^-(F)$ be an even integer improves the bound slightly as in the following.

**Corollary 7.2.6.** *An upper bound on the closed-shell independence number is*

$$\alpha^{-}(F) \leq 2\lfloor 3n/16 + 12/16 \rfloor .$$

Note that the bound in Corollary 7.2.5 is equal to the bound in Corollary 7.2.6 if and only if $n \equiv 12 \pmod{16}$. Comparison of the new bound with the previously known bound (Property 7.1.4), shows that the new bound is at least as good as the old for $n \geq 46$ and is strictly better for $n \geq 126$.

## 7.3 The Backtracking Algorithm

This section discusses the general structure of the algorithm to find the closed-shell independence number of a fullerene. The heart of the backtracking algorithm is recursive, as shown in the pseudo-code of Algorithm 7.1.

The algorithm maintains the status of each vertex as white, black, or gray. *White* denotes vertices in the closed-shell independent set and *black* denotes those not in the set. Gray vertices are the ones for which set membership has not yet been decided. A *black component* is a maximal connected subgraph in which all vertices are black. A closed-shell independent set has no adjacent whites and any black component surrounded by whites must be closed shell.

Initially, all of the vertices are gray. At each step of the algorithm, a gray vertex is selected. It is first colored white and then black. For each case, extensions to the partial coloring are explored by applying the algorithm recursively. When a coloring is complete, it corresponds to a closed-shell independent set. In order to achieve faster performance, the algorithm aborts exploration of partial colorings for which it can be concluded that either there are no feasible completions or that any feasible completions result in a closed-shell independent set size that is less than or equal to the best found so far.

Three parts of Algorithm 7.1 will now be described in more detail: how to select a gray vertex (line 6), how to determine if a partial coloring is feasible (lines 8 and 12), and how to detect when it is no longer possible to complete a partial coloring to obtain a new larger-sized set (lines 9 and 13).

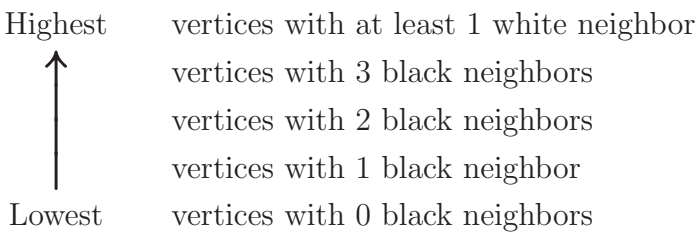
### 7.3.1 Selecting a Gray Vertex

Gray vertices are stored in a priority queue as they await coloring. The priority is assigned such that vertices with the least color flexibility are chosen first. Thus, infeasible colorings are detected faster. A vertex is assigned the highest possible priority according to the following descriptions:

```
ClosedShell (F)
Input    : A fullerene F
Returns: Closed-shell independence number of F
begin
1   | mark all vertices of F as gray;
2   | return ClosedShellLevel(F, 0, 0);
end

ClosedShellLevel (F, NumColored, LargestSeen)
Input    : A fullerene F with a feasible partial coloring having NumColored colored
           vertices and LargestSeen: the size of the largest c.s.i. set found previously
Returns: The larger of (i) the input value of LargestSeen and (ii) the size of the
           largest c.s.i. set possible by completing the given partial coloring
begin
3   | if NumColored = n then no gray vertices remain
4   |   | LargestSeen ← max(LargestSeen, white vertex count);
5   |   | return LargestSeen;
6   | Select a gray vertex v from F;
7   | Color v white;
8   | if current coloring is feasible then
9   |   | if larger set size may be possible then
10  |   |   | LargestSeen ← ClosedShellLevel(F, NumColored + 1, LargestSeen);
11  | Color v black;
12  | if current coloring is feasible then
13  |   | if larger set size may be possible then
14  |   |   | LargestSeen ← ClosedShellLevel(F, NumColored + 1, LargestSeen);
15  | Color v gray;
16  | return LargestSeen;
end
```

**Algorithm 7.1**: Essentials of the backtracking algorithm




A vertex with a white neighbor has the highest priority, since those must be colored black to avoid adjacent whites. The priority of a vertex is increased as appropriate until it is eventually selected and removed from the queue. Vertices are selected from the highest occupied priority in a first-in first-out manner.

### 7.3.2 Feasibility

After a gray vertex is selected, it is colored first white then black. Then the algorithm proceeds with this coloring only if it is feasible (lines 8 and 12 of Algorithm 7.1). This section gives the conditions under which a coloring is feasible and how feasibility is determined.

At all times, the fullerene has a partial coloring with each vertex being either white, black, or gray. Such a coloring is said to be *feasible* if it does not violate any requirements of a closed-shell independent set. Thus, a feasible partial coloring obeys the following rules:

1. No two white vertices are adjacent.

2. All black components not adjacent to gray vertices (surrounded by white vertices) are closed shell.

To maintain feasibility, these rules are checked after a selected gray vertex is colored. Rule 1 is only checked after a vertex is colored white.

Rule 2 is checked after a vertex is colored either white or black. If the selected vertex was colored black, then the algorithm examines the black component containing the selected vertex. A black component is first checked to see if it is surrounded by white vertices. In such a case, the black component is tested to see if it is closed shell. If a black component is not surrounded by white vertices, then one of the adjacent gray vertices may be colored black in the future, so no decision is made at this time as to whether or not it is closed shell. The application of rule 2 after coloring a selected vertex white is similar to the black case. Here, the algorithm considers the black components that contain the black neighbors of the selected vertex.

### 7.3.3 Improvement After Coloring

Once a coloring is determined to be feasible, two dynamic upper bounds are calculated based on the current partial coloring (lines 9 and 13 of Algorithm 7.1). These values bound the total number of white vertices that can be present if the remaining gray vertices are colored according to the feasibility rules. The bounds are compared to $LargestSeen$ and if they indicate that $LargestSeen$ cannot be improved by coloring the rest of the vertices, then the algorithm does not continue with the current partial coloring.

The first upper bound is calculated after a vertex is colored either white or black. Define an *gray edge* to be an edge incident to at least one gray vertex. Each additional white vertex will require 3 gray edges and because white vertices are non-adjacent, these edges are disjoint. Hence, the number of additional white vertices is at most 1/3 the number of gray edges. Furthermore, by Property 7.1.3, every closed-shell independent set must have an even number of vertices. Thus, the algorithm proceeds with the current coloring only when the current white vertex count plus 1/3 the number of gray edges is at least $LargestSeen + 2$.

The second upper bound is calculated only after a vertex is colored black (line 13 only). This bound relies on the following property [27].

**Property 7.3.1.** *Every closed-shell independent set of a fullerene satisfies*

$$w = (2n - b_2 - 2b_3)/5$$

*where $w$ is the number of white vertices and $b_i$ is the number of black vertices with $i$ black neighbors.*

A new maximum-sized set would have cardinality $w$ such that $w \geq LargestSeen+2$. Hence, the algorithm proceeds with the current coloring only when

$$5(LargestSeen + 2) \leq 2n - b_2 - 2b_3.$$

Black vertices with gray neighbors are included in this calculation because the future addition of black neighbors cannot increase the right-hand side. Implementation of this second upper bound resulted in the algorithm running approximately 20 times faster than when only the first upper bound is used.

## 7.4 Data Structures

As mentioned earlier, the algorithm considers components induced by the black vertices in order to detect violations to feasibility. As well, there is a priority queue used to order the selection of the gray vertices. This section describes the specialized data structures used to maintain and access this information.

### 7.4.1 Priority Queue

As described earlier, the gray vertex selection order is maintained in a priority queue. The algorithm requires the ability to remove the highest-priority vertex, change (increase) the priority of a vertex, and undo either of these changes when the algorithm backs up. Each of these operations has been implemented to run in constant time. This is crucial because approximately one quarter of the execution time of the algorithm is spent on these operations.

Because there are only five priority levels, the priority queue is implemented as five queues. The vertex at the head of each queue, if any, is stored. Predecessor and successor values for each vertex are used to create a circular doubly-linked list for each queue. Arrays of size $n$ are used to store the priority, predecessor, and successor of each vertex.

Each operation is performed in constant time. When a vertex is removed, it is taken from the head of the highest-priority nonempty queue. When the priority of a vertex changes, due to the coloring of a neighbor, the vertex is removed from its current linked list and added to the tail of the list for the new priority. When the algorithm backs up, either of these changes is undone by remembering the old priority, predecessor, and successor of the vertex and using these values to re-insert the vertex into its earlier position. By returning vertices to their original position in the queue instead of simply adding them to the tail of the appropriate queue, the algorithm runs in less than a third of the time.

### 7.4.2 Black Components

The algorithm maintains information about the components induced by the black vertices. This information includes the vertices in the component as well as descriptive values such as whether or not a closed-shell calculation should be run on the component, the cached result of such a calculation, and other values that identify components with faster methods of calculation as described in Section 7.5.

Each black vertex is in exactly one black component, allowing a disjoint-set data structure using rooted trees to store these components. For an overview of disjoint

sets and their use in connected components, see [13, Ch. 21]. An implementation without path compression was chosen because the data structure must be able to quickly undo previous union operations when the algorithm recolors a vertex gray. For the same reason, each parent-child relationship is doubly-linked instead of only the child-to-parent direction.

Neighboring black vertices are always in the same component. In the union operation, a newly colored black vertex $v$ is merged with the black components that contain $v$'s black neighbors. Vertex $v$ is assigned to be the root of the merged component and its children are the unique roots of its black neighbors' components.

When the algorithm backs up by coloring a black vertex gray, the previous union operation must be undone. The algorithm guarantees that vertices are recolored gray in the opposite order in which they were colored. Hence, a newly gray vertex is always the root of its component and its removal splits the component into the original components.

Values that describe the component and aid future closed-shell calculations are associated with the root. When performing a union, these values are calculated from the corresponding values of the children. When a union is undone, the previous values are recovered because they are still associated with the roots of the resulting components. For example, each root stores the number of vertices of each degree (within the component), 0 to 3. This compact storage of the degree sequence can be used to identify paths, cycles, trees, and unicyclic graphs.

A black component with no adjacent gray vertices cannot be extended and therefore may be tested to see if it is closed shell. To identify this situation, a count of the number of edges from a vertex in the component to a gray vertex is stored at the root. When this count is zero, the closed-shell calculation is performed.

The last value stored at a root is a cache of the result of the relatively slow closed-shell calculation. This avoids re-calculations that would otherwise occur when multiple neighbors of a newly colored white vertex are in the same black component.

## 7.5 Decreasing the Run Time

A large run time bottleneck is the closed-shell calculation that is performed each time a black component becomes surrounded by white vertices. From the definition of a closed-shell independent set, the distribution of the eigenvalues must be known to determine if exactly half of them are positive. It is desirable to avoid lengthy explicit eigenvalue calculations, which is possible for a large percentage of the cases that are encountered.

For paths and cycles, the closed-shell calculation is very easy. It has been shown [48] that the eigenvalues of a path satisfy the equation

$$\lambda_k = 2\cos\left[\frac{\pi k}{n+1}\right]$$

and the eigenvalues of a cycle satisfy the equation

$$\lambda_k = 2\cos\left[\frac{2\pi k}{n}\right]$$

for $1 \leq k \leq n$. One may assume that $n$ is even because this is a requirement for closed shells. It follows from the equations above that all even-length paths are closed shell and an even-length cycle is closed shell if and only if $n$ is not divisible by 4. Hence, the calculation only requires knowledge of $n$ for paths and cycles.

Although not as fast as paths and cycles, the closed shell calculation for trees is also quick. For any bipartite graph, if $\lambda$ is an eigenvalue then $-\lambda$ is an eigenvalue with the same multiplicity [16, Theorem 3.11]. A direct application of [16, Theorem 1.3] shows that 0 is an eigenvalue if and only if the tree has no perfect matching. Because trees are bipartite, their eigenvalues are symmetric and therefore a tree is closed shell if and only if it has a perfect matching. This reduces the calculation to one taking linear time.

Section 6.4 described a method to decide whether or not a unicyclic graph is closed shell. Such a calculation also takes linear time.

After filtering out the black-component graphs having fast eigenvalue computations, some graphs remain to be tested using numerical eigenvalue computations. Many methods exist to compute the eigenvalues, however the fastest method was to compute the inertia instead of the actual eigenvalues. The DSPTRF subroutine of the LAPACK libraries, which factors the matrix using Bunch-Kaufman diagonal pivoting, was used for the computation. The computations were carried out using double precision for the highest precision possible using LAPACK.

There are ways to improve the run time other than avoiding or improving the eigenvalue calculations. Note that the dynamic upper bounds discussed in Section 7.3.3 become more restrictive as larger-sized sets are found. However, test runs of the algorithm showed that for most isomers large-sized sets are not found quickly. It is desirable to have restrictive bounds so that more partial colorings are eliminated from consideration, thereby decreasing the run time. From Corollary 7.2.6 and Property 7.1.4, $\alpha^-(F) \leq \min\{2\lfloor n/5\rfloor, 2\lfloor 3n/16 + 12/16\rfloor\}$. The test runs indicated that the

closed-shell independence number is not far below this value for fullerenes with 100 or fewer vertices. Furthermore, by definition, every closed-shell independent set has an even number of vertices.

This information leads to a method that decreases the run time by looking for a set of a given large size rather than any size. Set a variable *UpperBound* to $\min\{2\lfloor n/5\rfloor, 2\lfloor 3n/16 + 12/16\rfloor\}$. Instead of setting *LargestSeen* to be 0 initially, it is set to *UpperBound* − 2. This makes the algorithm search only for sets of size *UpperBound*. The bounds are then more restrictive and the algorithm runs faster. If such a set is found, then the search can stop immediately because it matches the theoretical upper bound of $\alpha^-(F)$. If no such set is found, the algorithm is restarted with $LargestSeen = UpperBound - 4$. Then if a set of size $UpperBound - 2$ is found, the algorithm can stop immediately because it already checked for larger sets. Otherwise, the initial value of *LargestSeen* will continue to decrease by 2 until a set is found. One disadvantage of this method is that the run time may increase if either the closed-shell independence number is far below *UpperBound* or if a large set would be found quickly by the original ($LargestSeen = 0$) method. However, in most cases of interest this method decreased the run time of the algorithm.

## 7.6 Computational Results

The algorithm was run on all 1,456,598 fullerene isomers on up to 100 vertices. Table 7.1 shows the number of fullerene isomers on $n$ vertices with $\alpha^-(F)$ a given distance below the theoretical upper bound of $2\lfloor n/5\rfloor$. Table 7.2 shows the same data against the new bound from Corollary 7.2.6. Figure 7.2 shows a plot of the bound in Corollary 7.2.6 along with the bound from Property 7.1.4 and a dot at each location where there exists a fullerene with the specified $n$ and $\alpha^-(F)$ values. These data sets are useful for checking how the quality of the upper bounds and the spread of $\alpha^-(F)$ vary with increasing $n$. The largest-known $n$ where the new bound is tight $n = 96$, whereas the largest-known $n$ where the previous bound is tight is $n = 84$.

A characterization of the fullerenes that maximize $\alpha^-(F)$ for a given $n$ is useful for understanding the closed-shell independent set problem. Notice that $2\lfloor n/5\rfloor$ increases when $n \equiv 0$ or $n \equiv 6 \pmod{10}$. As expected, Table 7.1 shows a step away from $2\lfloor n/5\rfloor$ for most isomers at these values of $n$. The shifts are somewhat larger at multiples of 10, as expected.

An interesting observation is that there seems to be a correlation between $\alpha^-(F)$ and the isomer number, i.e., the index of the isomer in the list of canonical face spirals [25, 2.6]. For values of $n$ with relatively few isomers having the largest $\alpha^-(F)$, the

| $n$ | $x=6$ | $x=4$ | $x=2$ | $x=0$ | $n$ | $x=6$ | $x=4$ | $x=2$ | $x=0$ |
|---|---|---|---|---|---|---|---|---|---|
| 20 | - | - | - | 1 | 62 | - | 1 | 2347 | 37 |
| 24 | - | - | - | 1 | 64 | - | - | 2789 | 676 |
| 26 | - | - | 1 | - | 66 | - | 651 | 3827 | - |
| 28 | - | - | - | 2 | 68 | - | 4 | 6200 | 128 |
| 30 | - | - | 3 | - | 70 | - | 6005 | 2144 | - |
| 32 | - | - | 3 | 3 | 72 | - | 1072 | 10107 | 11 |
| 34 | - | - | - | 6 | 74 | - | 2 | 13841 | 403 |
| 36 | - | - | 13 | 2 | 76 | - | 12681 | 6470 | - |
| 38 | - | - | 2 | 15 | 78 | - | 1308 | 22773 | 28 |
| 40 | - | - | 39 | 1 | 80 | 2 | 30753 | 1169 | - |
| 42 | - | - | 35 | 10 | 82 | - | 23058 | 16660 | - |
| 44 | - | - | 2 | 87 | 84 | - | 1641 | 49854 | 97 |
| 46 | - | - | 115 | 1 | 86 | 1 | 60544 | 3216 | - |
| 48 | - | - | 117 | 82 | 88 | - | 41480 | 40258 | - |
| 50 | - | 2 | 268 | 1 | 90 | 1748 | 98095 | 75 | - |
| 52 | - | - | 416 | 21 | 92 | - | 118707 | 7702 | - |
| 54 | - | - | 211 | 369 | 94 | - | 67341 | 86152 | - |
| 56 | - | - | 924 | - | 96 | 1958 | 189627 | 254 | - |
| 58 | - | - | 1074 | 131 | 98 | 1 | 213326 | 17690 | - |
| 60 | - | 460 | 1344 | 8 | 100 | 107378 | 178536 | - | - |

Table 7.1: Number of fullerenes $C_n$ with $\alpha^-(F) = 2\lfloor n/5 \rfloor - x$

| $n$ | $x=6$ | $x=4$ | $x=2$ | $x=0$ | $n$ | $x=6$ | $x=4$ | $x=2$ | $x=0$ |
|---|---|---|---|---|---|---|---|---|---|
| 20 | - | - | - | 1 | 62 | - | 1 | 2347 | 37 |
| 24 | - | - | 1 | - | 64 | - | - | 2789 | 676 |
| 26 | - | - | 1 | - | 66 | - | 651 | 3827 | - |
| 28 | - | - | 2 | - | 68 | - | 4 | 6200 | 128 |
| 30 | - | - | 3 | - | 70 | - | - | 6005 | 2144 |
| 32 | - | - | 3 | 3 | 72 | - | 1072 | 10107 | 11 |
| 34 | - | - | 6 | - | 74 | - | 2 | 13841 | 403 |
| 36 | - | - | 13 | 2 | 76 | - | 12681 | 6470 | - |
| 38 | - | - | 2 | 15 | 78 | - | 1308 | 22773 | 28 |
| 40 | - | - | 39 | 1 | 80 | - | 2 | 30753 | 1169 |
| 42 | - | - | 35 | 10 | 82 | - | 23058 | 16660 | - |
| 44 | - | 2 | 87 | - | 84 | - | 1641 | 49854 | 97 |
| 46 | - | - | 115 | 1 | 86 | - | 1 | 60544 | 3216 |
| 48 | - | - | 117 | 82 | 88 | - | 41480 | 40258 | - |
| 50 | - | 2 | 268 | 1 | 90 | - | 1748 | 98095 | 75 |
| 52 | - | - | 416 | 21 | 92 | - | 118707 | 7702 | - |
| 54 | - | - | 211 | 369 | 94 | - | 67341 | 86152 | - |
| 56 | - | - | 924 | - | 96 | - | 1958 | 189627 | 254 |
| 58 | - | - | 1074 | 131 | 98 | 1 | 213326 | 17690 | - |
| 60 | - | 460 | 1344 | 8 | 100 | - | 107378 | 178536 | - |

Table 7.2: Number of fullerenes $C_n$ with $\alpha^-(F) = 2\lfloor 3n/16 + 12/16 \rfloor - x$

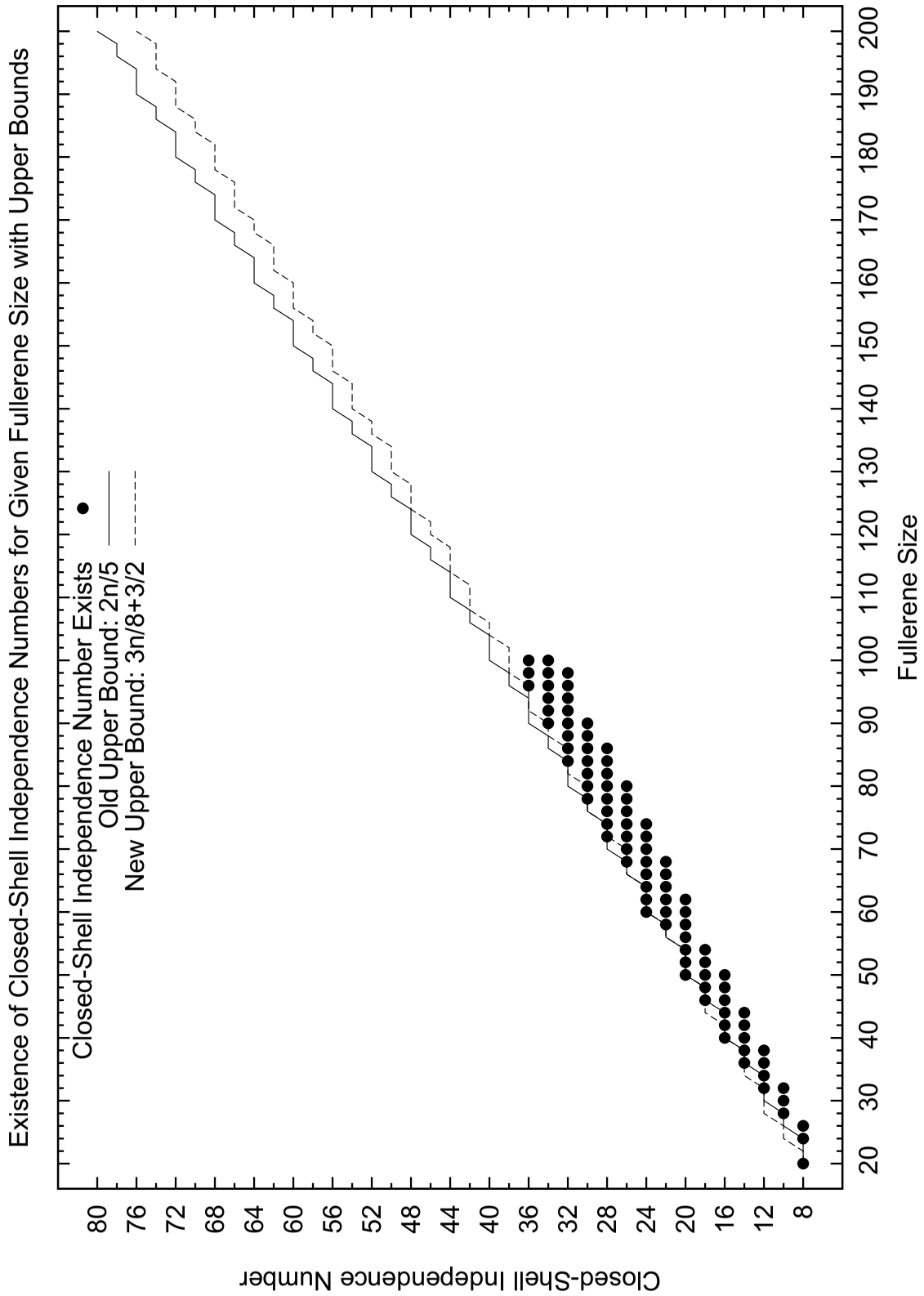



Figure 7.2: A plot of the new closed-shell independence number upper bound.

isomer numbers tend to be larger. Fullerenes with larger isomer numbers tend to have fewer *pentagon adjacencies*: the number of edges incident to two pentagons. Now consider only the values of $n$ in which relatively few isomers achieve the maximum $\alpha^-(F)$: 60, 62, 68, 72, 74, 78, 84, 90 and 96. Of the 1041 isomers that maximize $\alpha^-(F)$ for these values of $n$, the number of pentagon adjacencies ranges from 0 to 7 with a mean of only 2.68. Note, however, that on these values of $n$ there is a plethora of isomers that have few pentagon adjacencies but do not achieve the maximum $\alpha^-(F)$. Furthermore, for values of $n$ not in this group, the isomer numbers with maximum $\alpha^-(F)$ are more evenly distributed.

The next natural question is whether or not there is a correlation with the isomer number or number of pentagon adjacencies for those isomers that minimize $\alpha^-(F)$ for a given $n$. Consider only the values of $n$ for which relatively few isomers achieve the minimum $\alpha^-(F)$: 50, 62, 68, 74, 80, 86 and 98. In these cases the isomer numbers have no preference toward higher or lower values, but the number of pentagon adjacencies tends to be high. Of the 13 isomers that minimize $\alpha^-(F)$ for these values of $n$, the number of pentagon adjacencies ranges from 8 to 20 with a mean of 12.85. However, there are many isomers with a large number of pentagon adjacencies that do not achieve the minimum.

## 7.7 Relationship to the Independence Number

There is a strong interest in the relationship between the independence number, $\alpha(F)$, and the closed-shell independence number. The numerical difference between the two is of particular interest. Table 7.3 summarizes this relationship by giving the number of isomers on $n$ vertices with a given distance between $\alpha^-(F)$ and $\alpha(F)$.

Table 7.3 shows that there are exactly three isomers in which $\alpha(F) = \alpha^-(F)$. These are $C_{20}$, $C_{40}$:40, and $C_{60}$:1812[1]. Though the calculations here only go as far as 100 vertices, $\alpha(F)$ has been calculated for all fullerenes on 120 or fewer vertices [26]. These results confirm that no additional fullerenes exist with $\alpha(F) = \alpha^-(F)$ through 120 vertices because all values of $\alpha(F)$ are strictly greater than $2\lfloor n/5 \rfloor$ for fullerenes on at least 100 vertices. Hence, the following conjecture is made.

**Conjecture 7.7.1.** *The only fullerene isomers with* $\alpha^-(F) = \alpha(F)$ *are* $C_{20}$, $C_{40}$:40, *and* $C_{60}$:1812.

It has already been shown [26] that the only icosahedral fullerene isomers with $\alpha^-(F) = \alpha(F)$ are $C_{20}$ and $C_{60}$:1812. Data given in Table 7.3 and [26] further support

[1] $C_{60}$:1812 is the icosahedral Buckminsterfullerene often referred to simply as $C_{60}$.

| $n$ | $x=0$ | $x=1$ | $x=2$ | $x=3$ | $x=4$ | $x=5$ | $x=6$ | $x=7$ | $x=8$ | $x=9$ | $x=10$ | $x=11$ | $x=12$ | $x=13$ | $x=14$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 1 | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 24 | - | 1 | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 26 | - | - | - | 1 | - | - | - | - | - | - | - | - | - | - | - |
| 28 | - | 1 | 1 | - | - | - | - | - | - | - | - | - | - | - | - |
| 30 | - | - | 2 | 1 | - | - | - | - | - | - | - | - | - | - | - |
| 32 | - | 2 | 1 | 1 | 2 | - | - | - | - | - | - | - | - | - | - |
| 34 | - | - | 5 | 1 | - | - | - | - | - | - | - | - | - | - | - |
| 36 | - | 2 | - | 12 | 1 | - | - | - | - | - | - | - | - | - | - |
| 38 | - | - | 13 | 2 | 2 | - | - | - | - | - | - | - | - | - | - |
| 40 | 1 | - | - | 36 | 3 | - | - | - | - | - | - | - | - | - | - |
| 42 | - | - | 10 | - | 32 | 3 | - | - | - | - | - | - | - | - | - |
| 44 | - | - | 3 | 79 | 5 | 1 | 1 | - | - | - | - | - | - | - | - |
| 46 | - | - | 1 | 10 | 101 | 4 | - | - | - | - | - | - | - | - | - |
| 48 | - | - | 8 | 74 | 9 | 103 | 5 | - | - | - | - | - | - | - | - |
| 50 | - | - | 1 | 25 | 238 | 5 | 2 | - | - | - | - | - | - | - | - |
| 52 | - | - | 6 | 15 | 77 | 332 | 7 | - | - | - | - | - | - | - | - |
| 54 | - | - | 1 | 100 | 268 | 22 | 180 | 9 | - | - | - | - | - | - | - |
| 56 | - | - | - | 1 | 257 | 656 | 10 | - | - | - | - | - | - | - | - |
| 58 | - | - | 2 | 84 | 45 | 295 | 771 | 8 | - | - | - | - | - | - | - |
| 60 | 1 | 3 | 4 | 3 | 644 | 697 | 88 | 361 | 11 | - | - | - | - | - | - |
| 62 | - | - | 2 | 32 | 7 | 1049 | 1284 | 10 | 1 | - | - | - | - | - | - |
| 64 | - | - | - | 18 | 521 | 137 | 1278 | 1494 | 17 | - | - | - | - | - | - |
| 66 | - | - | - | - | 24 | 2370 | 1433 | 188 | 448 | 15 | - | - | - | - | - |
| 68 | - | - | - | 13 | 112 | 49 | 3843 | 2295 | 16 | 2 | 2 | - | - | - | - |
| 70 | - | - | - | 1 | 81 | 1817 | 261 | 3578 | 2394 | 17 | - | - | - | - | - |
| 72 | - | - | - | 6 | 6 | 199 | 7358 | 2549 | 433 | 615 | 24 | - | - | - | - |
| 74 | - | - | - | - | 100 | 297 | 261 | 9957 | 3611 | 18 | 2 | - | - | - | - |
| 76 | - | - | - | - | 1 | 580 | 5469 | 540 | 8845 | 3688 | 28 | - | - | - | - |
| 78 | - | - | - | 3 | 25 | - | 1104 | 17747 | 3923 | 642 | 639 | 26 | - | - | - |
| 80 | - | - | - | - | 1 | 416 | 747 | 1677 | 23734 | 5318 | 29 | 2 | - | - | - |
| 82 | - | - | - | - | - | 3 | 2669 | 13303 | 1261 | 17283 | 5168 | 31 | - | - | - |
| 84 | - | - | - | - | 13 | 77 | 11 | 5496 | 38503 | 5858 | 908 | 686 | 40 | - | - |
| 86 | - | - | - | - | - | 7 | 1697 | 1502 | 6773 | 46497 | 7255 | 29 | 1 | - | - |
| 88 | - | - | - | - | - | - | 27 | 10827 | 28314 | 3674 | 31798 | 7055 | 43 | - | - |
| 90 | - | - | - | - | - | 8 | 64 | 37 | 19135 | 70685 | 8258 | 1026 | 658 | 47 | - |
| 92 | - | - | - | - | - | - | 80 | 5079 | 2556 | 23748 | 84998 | 9901 | 47 | - | - |
| 94 | - | - | - | - | - | - | - | 150 | 32555 | 51897 | 9011 | 50764 | 9066 | 50 | - |
| 96 | - | - | - | - | - | - | 59 | 181 | 283 | 57477 | 120769 | 11157 | 1195 | 659 | 59 |
| 98 | - | - | - | - | - | - | - | 318 | 13253 | 4225 | 63871 | 136465 | 12841 | 43 | 1 |
| 100 | - | - | - | - | - | - | - | 1 | 1040 | 86672 | 88633 | 21709 | 76178 | 11611 | 70 |

Table 7.3: Number of fullerenes $C_n$ with $\alpha(F) - \alpha^{-}(F) = x$

Conjecture 7.7.1 by showing that $\alpha^-(F)$ and $\alpha(F)$ move further away from $2\lfloor n/5 \rfloor$ as $n$ increases.

The three isomers in Conjecture 7.7.1 have several things in common. They each have $\alpha^-(F) = 2n/5$. Therefore, Property 7.3.1 implies that every black component is a copy of $K_2$. The fullerene $C_{20}$ has only one maximum closed-shell independent set up to isomorphism, since the selection of two vertices for a black component completely determines the set. In [27], uniqueness (up to isomorphism) was shown for $C_{60}$:1812. Enumeration verifies that $C_{40}$:40 also has a unique maximum-sized set.

These isomers differ in the distribution of their pentagons. The dodecahedron $C_{20}$ has only 12 faces, all of which are pentagons. The pentagons of $C_{40}$:40 are in four groups of 3, evenly spaced throughout the fullerene. The truncated icosahedron $C_{60}$:1812 has isolated pentagons evenly spaced throughout the fullerene. The isomer $C_{40}$:40 has 24 automorphisms and its point group is $T_d$, whereas the others have 120 automorphisms and point group $I_h$.

## 7.8 Future Work

This chapter has described an algorithm with an implementation that is fast, despite all of the eigenvalue calculations. This enabled the calculation of the closed-shell independence number of all fullerene isomers with 100 and fewer vertices. From this data set, some interesting observations were made including a correlation with pentagon adjacencies. Most notable is the conjecture that there are only three fullerene isomers that have a maximum independent set that is closed shell.

Several opportunities exist to extend this work, including a more comprehensive data analysis and speeding up the implementation to obtain fast calculations for larger fullerenes. A faster implementation may be possible by the use of better data structures or solving some of the open problems below. It is desirable to have calculations that extend as far as the 120-vertex fullerenes. Though due to the exponential growth rate of both the number of fullerenes per $n$ and the time to calculate $\alpha^-(F)$ for increased $n$, such computations would take tens of CPU years with current machines.

There are many questions regarding closed-shell independent sets that remain open. Answers to these questions would likely require expertise in graph theory, computing, and chemistry, to varying degrees. The following are a sample of the open problems.

**Complexity.** The complexity of computing the closed-shell independence number has not yet been determined. This is a difficult problem due to the eigenvalue requirement. If the problem is $NP$-hard, it is not clear which $NP$-complete problem

could be used for a proof. If the problem has a polynomial-time algorithm, then the discovery of a solution would likely benefit from knowledge of how to find a maximum closed-shell independent set of a large hexagonal region of a fullerene as well as how the pentagons disrupt the region. In either case, a characterization of the closed-shell subgraphs that does not require eigenvalue calculations would be very useful, if such a characterization exists.

**Existence.** Although many closed-shell independent sets have been found of varying sizes for every fullerene studied, there is no proof that a closed-shell independent set exists for every fullerene. Many bare fullerenes are not closed-shell, so not even the trivial empty set is a solution. The difficulty with finding such a proof is due to the eigenvalue requirement. Possible candidates for non-existence are fullerenes having more negative than positive eigenvalues, however the smallest such known fullerene has 628 vertices [22], which may hinder computations.

**Families.** Identify infinite families of fullerenes that maximize or minimize $\alpha^-(F)$ over all isomers of the associated $n$, similar to the work in Chapter 5. Such an exploration may give insight as to how the set behaves on large sections of hexagons.

**Proof of Conjecture 7.7.1.** The new upper bound in Corollary 7.2.6 is interesting in light of Conjecture 7.7.1, which states that only three fullerene isomers satisfy $\alpha^-(F) = \alpha(F)$, the largest of which is at $n = 60$. Comparing the upper bound of $\alpha^-(F) \leq 3n/8 + 3/2$ with the lower bound of $\alpha(F) \geq 3n/8$ [34] yields a curious gap. If it can be shown that for suitably large $n$, $\alpha^-(F) < 3n/8$, then this, along with a computer search, can provide evidence to prove the conjecture. Any suitably large $n$ must be greater than 96 because there exists at least one fullerene with $\alpha^-(F) = 3n/8$ at $n = 96$.

Theorem 7.2.4 shows that if $\alpha^-(F) < 3n/8$ for some fullerene, then $3/2 < 3F_1/8 + 3F_0/4$, which is equivalent to $4 < F_1 + 2F_0$. Therefore, if one could show that for suitably large $n$, $4 < F_1 + 2F_0$, it would imply Conjecture 7.7.1. One possible approach may be to find a restriction on the number of faces in $\mathcal{F}_3$ that can be distance two (in the dual graph) away from each other. From Lemma 7.2.1, there is a limit of six faces in $\mathcal{F}_3$ that can be distance two away from another face in $\mathcal{F}_3$. However, some sketches that exclude the non-closed-shell components $P_1$, $P_3$, and $K_{1,3}$ indicate that the bound may be four or five instead of six, but further investigation is needed. Not enough is currently known about the optimal way to select a maximum closed-shell independent set on a large hexagon grid.

**Tighter Upper Bound.** DeLaVina's conjecturing software Graffiti.pc [19] was given data similar to that in Tables 7.1 and 7.2 and made the following conjecture.

**Conjecture 7.8.1.** *For a fullerene $F$, $\alpha^-(F) \leq (n+12)/3$.*

Using the computed data, it can be seen that this bound is tight for $58 \leq n \leq 100$ except for $n = 66$. Using the requirement that $\alpha^-(F)$ and $n$ must be even integers, one can note that $2\lfloor (n+12)/6 \rfloor = 2\lfloor (n+13)/6 \rfloor$ for even values of $n$, so the conjecture can be changed to $\alpha^-(F) \leq (n+13)/3$.

**Conjecture 7.8.2.** *For a fullerene $F$, $\alpha^-(F) \leq (n+13)/3$.*

Another observation is that $2\lfloor (n+13)/6 \rfloor < \lceil 3n/8 \rceil$ for all $n \geq 98$. Together with the computer search already executed, a proof of Conjecture 7.8.2 immediately proves Conjecture 7.7.1.

Equation (7.1) can be rewritten as

$$\alpha^-(F) = \frac{n+13}{3} - \frac{9}{3} + \frac{1}{3}F_3 - \frac{1}{3}F_1 - \frac{2}{3}F_0,$$

which implies that

$$\alpha^-(F) \leq \frac{n+13}{3}$$

if and only if $F_3 \leq F_1 + 2F_0 + 9$. So another approach to proving Conjecture 7.7.1 could be to find a suitably large $n$ such that $F_3 \leq F_1 + 2F_0 + 9$.

**Closed-Shell Identification.** Finding fast methods to determine if trees and unicyclic graphs are closed-shell improved the running time of the CSI algorithm [17, 18]. If similar methods are found for other classes of graphs such as bicyclic or cactus graphs, there is a potential to improve the running time. However, empirical evidence suggests that only a very small fraction (0.02%) of the tested components are not trees or unicyclic graphs. Identification of and closed-shell testing of additional classes of graphs will likely be of little practical benefit, if any. However, such study will likely benefit the search for algorithms with different approaches by better understanding the structure of closed-shell components.

**Closed-Shell Subgraph Structure.** The linear-time algorithm for the independence number for fullerenes was found by studying the optimal way to find a maximum independent set on an hexagonal tessellation of the plane, and then observing how it is affected by twelve pentagons. To find an optimal arrangement for the closed-shell independent set, and prove that it is optimal, it may be useful to enumerate the fullerene subgraphs that are closed-shell. There is an infinite number of such subgraphs, however it is an interesting problem to enumerate all closed-shell fullerene subgraphs on, say, 20 or fewer vertices. Such a list should only include minimal subgraphs: graphs for which the removal of an even number of trivalent

vertices does not break the graph into components that are all closed-shell. It may also be possible to give a theoretical characterization of all the closed-shell fullerene subgraphs by specifying a set of graph classes.

Another approach to this problem would be to modify the CSI algorithm to output the components that it tests for the closed-shell property along with the results of the tests. A canonical form may be calculated (using `nauty` [40], for example) to remove duplicates. This would provide a large, though not provably exhaustive, database of fullerene subgraphs that could be used to make closed-shell or non-closed-shell conjectures for graph classes. Such conjectures may be made by visual inspection or by computing some relevant invariants such as the degree sequence, maximum matching size, the number of perfect and/or maximum matchings, and the number of spanning trees and then using Graffiti.pc to mine the data.

**Only Trees.** The closed-shell components that compose the optimal solutions found by the CSI algorithm have varied structure including paths, cycles, trees and unicyclic graphs. One question for further study is whether or not it is possible to achieve the optimal solution using only trees as the closed-shell components. If so, can it also be done only using paths? It is easy to search for counterexamples by making some simple modifications to the CSI algorithm and running it again on all of the fullerenes of at most 100 vertices. Such an algorithm would likely run faster because it is more restrictive.

It is interesting to note that if a maximum closed-shell independent set can be found where the closed-shell components are only trees, then it implies that every component has a perfect matching. This would satisfy both the valence-bond (Kekulé structures) and molecular-orbital chemical models and please both sides of the debate.

**Icosahedral Fullerenes.** Analyzing the icosahedral fullerenes is often a good start before attempting to understand fullerenes in general. These fullerenes are the most symmetric and therefore are the most predictable. Some theoretical results on $\alpha(F) - \alpha^-(F)$ have been published [26]. The closed-shell independence number has been computed for the first few icosahedral fullerenes using the exponential-time algorithm: $\alpha^-(C_{20}) = 8$, $\alpha^-(C_{60}{:}1812) = 24$, $\alpha^-(C_{80}{:}31924) = 28$, $\alpha^-(C_{140}{:}7341204) = 48$, and $56 \leq \alpha^-(C_{180}{:}79538725) \leq 62$. The latter bounds were found by a mix of computer search and theoretical argument. The algorithm currently does not consider symmetry, so many isomorphic sets are tested unnecessarily. Implementing a method to eliminate isomorphic searches could greatly speed up the algorithm in the case of the icosahedral fullerenes.

**Bromination Sequence.** To form a final fullerene derivative with Br atoms

bonded to the surface of a fullerene at a (maximum) closed-shell independent set, the Br atoms do not bond simultaneously, rather they are added until the final bonding configuration is reached [51]. It is an open question to find sequences of sets that build to the final configuration [26]. As mentioned at the beginning of this chapter, these intermediate sets need not be independent, as the steric constraints may be temporarily relaxed, however the sets are likely closed-shell [3, 4, 5, 24, 47, 1]. Determining which constraints may be relaxed, when they may be relaxed, and how much they may be relaxed is a subject of further study, so this question has a flavor more fitting for chemical study than graph theory.

**Minimum Closed-Shell Independent Sets.** Related to the questions of the existence of a closed-shell independent set and finding a bromination sequence is the question of finding the minimum closed-shell independent set for a fullerene. That is, what is the smallest number of addends required to bond to a fullerene to make it stable? This is an interesting related problem from a graph theoretical point of view. Many bare fullerenes are not closed-shell and are therefore not stable on their own.

**Gaps from Minimum to Maximum.** Does there exist a closed-shell independent set of every (even) size between the minimum and maximum? Answering such a question may give insight to the bromination sequence. It would be worthwhile to study every non-isomorphic closed-shell independent set of $C_{60}$:1812 and $C_{70}$:8149 in a manner similar to the study of all independent sets in [26]. Due to the large number of computations require to investigate such a question, this analysis can likely only be performed on a few fullerenes.

# Chapter 8

# Conclusion

Studies of maximum independent sets and maximum closed-shell independent sets of fullerenes have resulted in a number of new contributions. This chapter highlights the most important contributions and discusses some of the key approaches used in this research. A number of open problems have been presented throughout this dissertation and the most interesting related problems are highlighted here.

## 8.1 Main Results

The most difficult result of this work was the development of the algorithm for finding a maximum independent set of any fullerene with a running time that is a linear function of $n$. This result was made possible by J. Graver's discovery that a maximum independent set is related to a fullerene's clear fields [31]. Chapter 3 extended his results and defined a proper clear field. It was shown that the number of clear fields is proportional to a linear function of $n$ for some fullerenes, whereas the number of proper clear fields is a logarithmic function of $n$ in the worst case (Theorem 3.2.8). This is the main observation that allowed a linear-time algorithm for finding a maximum independent set of any fullerene. Chapter 4 combined graph-theoretical results with clever algorithmic ideas to create an algorithm that runs in four phases. First, the algorithm locates the clear fields, then it detects which clear fields overlap. This information is used in the third phase to check all possible pentagon pairings using clear fields that do not overlap in order to find a optimal selection. Such a selection is then used to construct a maximum independent set.

The resulting linear-time algorithm was run on all fullerenes with 120 or fewer vertices and the results are given in Table 3.1. As mentioned in Chapter 3, this data has been used to refute [23, 26] a claim [21] that minimization of the independence number is a predictor of bare fullerene stability.

Other noteworthy results related to maximum independent sets are mentioned in Chapter 5. These include the formula for finding the independence number of a fullerene that is created from an even number of leapfrog transformations, provided the independence number of the original fullerene is known (Theorem 5.2.2). Section 5.3 gave a fullerene family that attains the upper bound of $n/2 - 2$ and the family

includes a fullerene at all values of $n$ for which a fullerene exists except 20, 24, 28, and 32. In most cases fullerenes in this family exhibit no symmetry, so families of symmetrical fullerenes that attain the upper bound are given in Section 5.4.

The other major focus of this work was the study of closed-shell independent sets and closed-shell graphs. The main result here is an exponential-time backtracking algorithm for finding a maximum closed-shell independent set of a fullerene that is fast enough for practical use (Chapter 7). Part of this algorithm tests fullerene subgraphs to see if they are closed-shell and in general uses some relatively computationally-expensive eigenvalue calculations. Faster calculation methods were discussed for basic graphs like paths, trees, and cycles. Chapter 6 discusses a linear-time algorithm for determining if a unicyclic graph is closed-shell, which came from the study of the inertia of these graphs.

Other significant results for maximum closed-shell independent sets include an improved upper bound of $3n/8 + 3/2$ for fullerene graphs. In Figure 7.2, this bound is plotted along with the old bound of $2n/5$ and the existence of a fullerene with given size and closed-shell independence number after the algorithm was run on all fullerenes having 100 or fewer vertices. Collecting data on the independence number and the closed-shell independence number for fullerenes allowed some comparisons to be made and led to Conjecture 7.7.1 that states that there are only three specific fullerenes for which these two values are equal.

## 8.2 Approaches

Various approaches were taken in order to achieve the results presented here. The most significant approach was the iterative process that was used to discover the linear-time algorithm for finding a maximum independent set. This process alternated between algorithmic and theoretical advances. First was Graver's graph theoretical results that led the author to the discovery of an algorithm that had a running time in $O(n^6)$, which was the first known polynomial-time algorithm. Then graph theoretical results related to the proper clear fields enabled the algorithm to be modified slightly to have a running time in $O(n^2)$. Finally, some algorithmic improvements reduced the running time to $O(n)$. Between each of these improvements, the algorithm was run on a large data set that allowed me to evaluate its performance and identify critical cases that had to be handled. It was analysis of these cases, which were never envisioned at the beginning, that allowed for the discovery of the proper clear fields.

Much of the work that involve analyzing special cases involved the use of FuiGui [42], which is a program that allows quick visualization of fullerene graphs. This

made it easy to visualize cases such as the clear fields that contain repeated vertices and how clear fields can be used to determine the colors of the pentagons in Phase 3. FuiGui also enabled the families in Chapter 5 to be easily discovered and viewed by browsing the fullerenes that maximize the independence number. This allowed patterns to be easily recognized. FuiGui was also instrumental in drawing the various figures throughout the text.

Having the two related but different problems of the independent sets and closed-shell independent sets enabled progress to be made by switching between the problems. When stuck on one issue, it was helpful to have another problem to work on. Often, coming back to a problem and figuring out where it was left resulted in new discoveries by realizing how pieces fit together in a bigger picture. One such discovery was the characterization of proper clear fields by realizing how to prove Theorem 3.2.5.

The approach for conjecturing involved a blend of computations and graph theory. Instead of looking at special cases and making conjectures based on them, programs were written to collect data. These programs were used in two ways. First, before conjectures were made, the programs were run to create a large collection of data, such as all fullerenes with 120 or fewer vertices. Then analysis of the data was used to make conjectures by methods such as observing the outlying cases or by using conjecturing software such as Graffiti.pc [19]. The other way that software was used was in testing conjectures that resulted from graph theoretical analysis by searching for counterexamples. For example, at one point it was conjectured that there is a constant upper bound on the number of proper clear fields. This was later shown to be false when several counterexamples were found that indicated a procedure to construct a fullerene with an arbitrary number of proper clear fields. Again FuiGui was instrumental in this analysis by facilitating visualization of the construction.

## 8.3 Open Problems

Open problems have been presented throughout this dissertation at the end of the appropriate chapters. Problems dealing with the maximum independent set algorithm are found in Section 4.6 and maximum independent sets in Section 5.5. Open problems dealing with maximum closed-shell independent sets can be found in Section 7.8.

Perhaps the most interesting open problem is Conjecture 7.7.1 that states that there are only three specific fullerenes with equal independence and closed-shell independence numbers. There are several possible approaches to proving this conjec-

ture. Section 7.8 discussed several approaches that involve improvement of the upper bound for the closed-shell independence number. Another possible approach is to improve the lower bound on the independence number, as discussed in Section 5.5. The conjectured best-possible lower bound on the independence number was given by Conjecture 5.5.2 and a proof would immediately prove Conjecture 7.7.1. Also mentioned was Daniel Král's approach that may improve the lower bound to $3n/8 + k$ for some small constant $k$, given sufficiently large $n$. If this $k$ is greater than $3/2$, then it would prove Conjecture 7.7.1 provided the "sufficiently large $n$" is within reason for a computer search on smaller values of $n$.

The other open problem of high interest is embodied by Conjecture 5.5.2, which is an improved lower bound of the independence number. Figure 5.8 showed that it is tight, and indeed it is attained by the infinite class of icosahedral fullerenes with Coxeter coordinates $(p, p)$.

Section 7.8 gave a large number of open problems related to closed-shell independent sets. The most interesting of these are the questions of existence and determining a maximum closed-shell independent set on a hexagonal tessellation of the plane. The question of existence is fascinating because many closed-shell independent sets can be found for a fullerene, yet no proof is known that one exists for an arbitrary fullerene. The tessellation question is interesting because results in this area may lead to knowing how to choose an closed-shell independent set of vertices for large patches of hexagons on a fullerene, much like the clear fields were used in the independent set problem.

# Bibliography

[1] S. J. Austin, P. W. Fowler, J. P. B. Sandall, P. R. Birkett, A. G. Avent, A. D. Darwish, H. W. Kroto, R. Taylor, and D. R. M. Walton. Theoretical characterisation of $C_{70}Cl_{10}$: the rôle of 1,4-addition across hexagonal rings. *Journal of the Chemical Society, Perkin Transactions 2*, pages 1027–1028, 1995.

[2] D. Babić, T. Došlić, D. J. Klein, and A. Misra. Kekulénoid addition patterns for fullerenes and some lower homologs. *Bulletin of the Chemical Society of Japan*, 77:2003–2010, 2004.

[3] P. R. Birkett, A. G. Avent, A. D. Darwish, H. W. Kroto, R. Taylor, and D. R. M. Walton. Preparation and $^{13}C$ NMR spectroscopic characterisation of $C_{60}Cl_6$. *Journal of the Chemical Society, Chemical Communications*, pages 1230–1232, 1993.

[4] P. R. Birkett, A. G. Avent, A. D. Darwish, H. W. Kroto, R. Taylor, and D. R. M. Walton. Formation and characterisation of $C_{70}Cl_{10}$. *Journal of the Chemical Society, Chemical Communications*, pages 683–684, 1995.

[5] P. R. Birkett, P. B. Hitchcock, H. W. Kroto, R. Taylor, and D. R. M. Walton. Preparation and characterization of $C_{60}Br_6$ and $C_{60}Br_8$. *Nature*, 357:479–481, 1992.

[6] I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo. The maximum clique problem. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 4. Boston, MA: Kluwer Academic Publishers, 1999.

[7] G. Brinkmann. *Fullgen Manual*. University of Ghent. Accessed on March 25, 2009 at `http://cs.anu.edu.au/~bdm/plantri/fullgen-guide.txt`.

[8] G. Brinkmann and A. W. M. Dress. A constructive enumeration of fullerenes. *Journal of Algorithms*, 23:345–358, 1997.

[9] G. Brinkmann and A. W. M. Dress. Penthex puzzles: a reliable and efficient top-down approach to fullerene-structure enumeration. *Advances in Applied Mathematics*, 21:473–480, 1998.

[10] T. L. Brown, H. E. LeMay, and B. E. Bursten. *Chemistry: The Central Science*. Upper Saddle River, NJ: Prentice Hall, tenth edition, 2006.

[11] A. Chang and F. Tian. On the spectral radius of unicyclic graphs with perfect matchings. *Linear Algebra and its Applications*, 370:237–250, 2003.

[12] V. Chvátal. Determining the stability number of a graph. *SIAM Journal on Computing*, 6:643–662, 1977.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. Cambridge, MA: MIT Press, second edition, 2001.

[14] J. D. Crane. Maximal non-adjacent addition to fullerene-70: Computation of all the closed shell isomers of $C_{70}X_{26}$. *Fullerene Science and Technology*, 7:427–435, 1999.

[15] D. Cvetković and P. Rowlinson. Spectra of unicyclic graphs. *Graphs and Combinatorics*, 3:7–23, 1987.

[16] D. M. Cvetković, M. Doob, and H. Sachs. *Spectra of Graphs: Theory and Application*. New York: Academic Press, 1980.

[17] S. Daugherty. The inertia of unicyclic graphs and the implications for closed shells. *Linear Algebra and its Applications*, 429:849–858, 2008.

[18] S. Daugherty, W. Myrvold, and P. W. Fowler. Backtracking to compute the closed shell independence number of a fullerene. *MATCH Commun. Math. Comput. Chem.*, 58(2):385–401, 2007.

[19] E. DeLaVina. Graffiti.pc: a variant of graffiti. *Graphs and Discovery, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 69:71–79, 2005.

[20] L. Euler. Demonstratio nonnullarum insignium proprietatum, quibus solida hedris planis inclusa sunt praedita. *Novi Commentarii Academiae Scientiarum Petropolitanae*, 4:140–160, 1758.

[21] S. Fajtlowicz and C. E. Larson. Graph-theoretic independence as a predictor of fullerene stability. *Chemical Physics Letters*, 377:485–490, 2003.

[22] P. W. Fowler. Fullerene graphs with more negative than positive eigenvalues: The exceptions that prove the rule of electron deficiency? *Journal of the Chemical Society, Faraday Transactions*, 93:1–3, 1997.

[23] P. W. Fowler, S. Daugherty, and W. Myrvold. Independence number and fullerene stability. *Chemical Physics Letters*, 448:75–82, 2007.

[24] P. W. Fowler, P. Hansen, K. M. Rogers, and S. Fajtlowicz. $C_{60}Br_{24}$ as a chemical illustration of graph theoretical independence. *Journal of the Chemical Society, Perkin Transactions 2*, pages 1531–1534, 1998.

[25] P. W. Fowler and D. E. Manolopoulos. *An Atlas of Fullerenes*. Oxford: Clarendon Press, 1995.

[26] P. W. Fowler and W. Myrvold. Some chemical applications of independent sets. In P. Hansen, N. Mladenovic, J. A. M. Perez, B. M. Battista, and J. M. Moreno-Vega, editors, *Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search*, Tenerife, Spain, 2005.

[27] P. W. Fowler, K. M. Rogers, K. R. Somers, and A. Troisi. Independent sets and the prediction of additional patterns for higher fullerenes. *Journal of the Chemical Society, Perkin Transactions 2*, pages 2023–2027, 1999.

[28] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[29] J. E. Graver. Catalog of all fullerenes with ten or more symmetries. *Graphs and Discovery, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 69:167–188, 2005.

[30] J. E. Graver. The structure of fullerene signatures. *Graphs and Discovery, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 69:137–165, 2005.

[31] J. E. Graver. The independence number of fullerenes and benzenoids. *European Journal of Combinatorics*, 27:850–863, 2006.

[32] J.-M. Guo, W. Yan, and Y.-N. Yeh. On the nullity and the matching number of unicyclic graphs. *Linear Algebra and its Applications*, In Press, 2009.

[33] I. Gutman and N. Trinajstić. A graph-theoretical classification of conjugated hydrocarbons. *Naturwissenschaften*, 60(10):475, 1973.

[34] C. C. Heckman and R. Thomas. Independent sets in triangle-free cubic planar graphs. *Journal of Combinatorial Theory, Series B*, 96:253–275, 2006.

[35] D. S. Johnson and M. A. Trick, editors. *Cliques, Colouring, and Satisfiability, Second DIMACS Implementation Challenge, Oct. 11-13, 1993*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. AMS, 1996.

[36] R. M. Karp and M. Sipser. Maximum matchings in sparse random graphs. *Proc. 22nd Annual IEEE Symp. on Foundations of Computer Science*, pages 364–75, 1981.

[37] H. W. Kroto, J. R. Heath, S. C. O'Brien, R. F. Curl, and R. E. Smalley. $C_{60}$: Buckminsterfullerene. *Nature*, 318:162–163, 1985.

[38] X. Li, J. Zhang, and B. Zhou. On unicyclic conjugated molecules with minimal energies. *Journal of Mathematical Chemistry*, 42:729–740, 2007.

[39] X. Li, J. Zhang, and B. Zhou. The spread of unicyclic graphs with given size of maximum matchings. *Journal of Mathematical Chemistry*, 42:775–788, 2007.

[40] B. D. McKay. *nauty Users Guide (Version 2.2)*, 2004. `http://cs.anu.edu.au/~bdm/nauty/`.

[41] S. G. Mohanty. *Lattice Path Counting and Applications*. Academic Press, New York, 1979.

[42] W. Myrvold, B. Bultena, S. Daugherty, B. Debroni, S. Girn, M. Minchenko, J. Woodcock, and P. W. Fowler. Fuigui: A graphical user interface for investigating conjectures about fullerenes. *MATCH Commun. Math. Comput. Chem.*, 58(2):403–422, 2007.

[43] W. Myrvold and P. W. Fowler. Fast enumeration of all independent sets of a graph. Manuscript, 20 pages, April 2007.

[44] M. Nath and B. K. Sarma. On the null-spaces of acyclic and unicyclic singular graphs. *Linear Algebra and its Applications*, 427:42–54, 2007.

[45] N. Robertson, D. P. Sanders, P. Seymour, and R. Thomas. A new proof of the four-colour theorem. *Electronic Research Announcements of the American Mathematical Society*, 2:17–25, 1996.

[46] J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7:425–440, 1986.

[47] K. M. Rogers and P. W. Fowler. A model for pathways of radical addition to fullerenes. *Chemical Communications*, pages 2357–2358, 1999.

[48] D. H. Rouvray. Chapter 7. In A. T. Balaban, editor, *Chemical Applications of Graph Theory*. London: Academic Press, 1976.

[49] C.-H. Sah. A generalized leapfrog for fullerene structures. *Fullerenes, Nanotubes and Carbon Nanostructures*, 2:445–458, 1997.

[50] A. Streitwieser, Jr. *Molecular Orbital Theory for Organic Chemists*. New York: John Wiley & Sons, 1961.

[51] R. Taylor, A. G. Avent, P. R. Birkett, T. J. S. Dennis, J. P. Hare, P. B. Hitchcock, J. H. Holloway, E. G. Hope, H. W. Kroto, G. J. Langley, M. F. Meidine, J. P. Parsons, and D. R. M. Walton. Isolation, characterization, and chemical reactions of fullerenes. *Pure and Applied Chemistry*, 65:135–142, 1993.

[52] J. V. Uspensky. *Theory of Equations*. McGraw-Hill, New York, 1948.

[53] W.-H. Wang, A. Chang, L.-Z. Zhang, and D.-Q. Lu. Unicyclic Hückel molecular graphs with minimal energy. *Journal of Mathematical Chemistry*, 39(1):231–241, 2006.

[54] H. Whitney. Non-separable and planar graphs. *Transactions of the American Mathematical Society*, 34:339–362, 1932.

[55] Y.-R. Zheng and A. Chang. An upper bound on the second largest eigenvalue of unicyclic graphs with perfect matchings. *J. Xinjiang Univ. Natur. Sci.*, 22:393–399, 2005.