



Michigan Technological University
Create the Future Digital Commons @ Michigan Tech

Dissertations, Master's Theses and Master's
Reports - Open

Dissertations, Master's Theses and Master's
Reports

2011

Cyclic automorphic graph decompositions

Michael Li Misson

Michigan Technological University

Follow this and additional works at: <https://digitalcommons.mtu.edu/etds>



Part of the [Mathematics Commons](#)

Copyright 2011 Michael Li Misson

Recommended Citation

Misson, Michael Li, "Cyclic automorphic graph decompositions ", Master's report, Michigan Technological University, 2011.

<https://doi.org/10.37099/mtu.dc.etds/547>

Follow this and additional works at: <https://digitalcommons.mtu.edu/etds>



Part of the [Mathematics Commons](#)

CYCLIC AUTOMORPHIC GRAPH DECOMPOSITIONS

by

Michael Li Misson

A REPORT

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

(Mathematical Sciences)

MICHIGAN TECHNOLOGICAL UNIVERSITY

2011

© 2011 Michael L. Misson

This report, “Cyclic Automorphic Graph Decompositions,” is hereby approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE IN MATHEMATICAL SCIENCES.

Mathematical Sciences

Signatures:

Report Advisor _____
Melissa Keranen, PhD

Committee Member _____
Donald Kreher, PhD

Committee Member _____
Dean Johnson, PhD

Department Chair _____
Mark S. Gockenbach, PhD

Date _____

Contents

List of Figures	7
List of Tables	9
Acknowledgements	11
Chapter 1 Introduction	15
1.1 Graph Theory	15
1.2 Automorphic Graph Decompositions	18
1.3 An introduction to graph labelings	20
Chapter 2 Graph Labelings	25
Chapter 3 ρ -labelings	30
Chapter 4 Conclusion	35
Bibliography	37
Appendix A	39
Appendix B	44

List of Figures

1.1.1	Two isomorphic simple graphs G , H on four vertices.	15
1.1.2	G a subgraph of H	16
1.1.3	Decomposition of K_4	17
1.1.4	G -decomposition of K_4	17
1.2.1	Decomposition \mathcal{D} of G and intersection graph $I(\mathcal{D})$	18
1.2.3	Two simple automorphic decompositions	19
1.2.6	The bow tie graph G	20
1.3.1	Induced edge labeling on a tree, given \mathbb{Z}_{11}	21
1.3.4	Closed \mathbb{Z}_n -valuations of non-graceful graphs on five vertices	23
1.3.5	A graceful labeling on C_4	23
1.3.6	\mathcal{D} , a C_4 -decomposition of K_9	24
2.2	Finding a \mathbb{Z}_{21} -valuation of C_{10}	29
3.5	Figure 3.5: Cycle on 5 vertices with a closed \mathbb{Z}_{11} -valuation	32
3.6	Base cycles for a cyclic 3-cycle system of K_{13} .	33

List of Tables

Table 1	Graphs with ρ -labelings	31
Table 2	Summary of Graceful Results	36
Table 3	Results on rC_m , $m \equiv 1 \pmod{4}$	39
Table 4	Results on rC_m , $m \equiv 2 \pmod{4}$	40

Acknowledgements

I would like to thank my advisor, Missy Keranen. She has provided the much needed support to achieve success here at Michigan Technological University. She also agreed to do research with me on a topic in which she was not originally well versed. I thank Donald Kreher for agreeing to be on my committee and for providing much guidance and insight into this topic. Also, I would like to thank Dean Johnson, for agreeing to be my outside committee member and reading my report.

Also, I thank my family for the unending support and their belief in me to succeed. My mother and father, Laia and Don, you have been a great example to me in what it means to love sacrificially and I cannot thank you enough for that. I will always remember the day my computer broke a week before my defense, and you were willing to drive a twenty hour round trip to make sure that I could finish. To my brothers, thank you for your wonderful example you have set for me. You have made my path through life easier and have set such high standards for me. For this I cannot thank you guys enough. I would also like to thank all of my extended family, those here in the United States, and those far away, in Hong Kong, Taiwan, and Kenya. You have always kept me well fed and I thank you for your unwavering thoughts and prayers of support.

I would also like to thank my fellow graduate students and friends who have been there for me through thick and thin, making sure I didn't go insane. Thank you Mel and Bill Laffin, Joshua Ruark, Janelle Doebel, Andrew Azzam, Richard Fears, James Wright, David Kamin, and John Asplund. Thank you to my friends away from Houghton, Justin Youngheim, Andrea Dashe, Mychal Ivancich, Lara Dent, Rachel Jacquin, David Michael Smeenge, Hannah Smith, Tom Day, Fred Kraft, and Daniel Bennett, who have all had an amazing impact on my life and have encouraged me throughout this season of my life. I also thank my friends who have been with me and close to me during this entire process. I thank Brandon Pereles, Sanchai Kuboon, Ake and Ochin, Aaron Green, John Hausman, and Tristan McKay.

Most importantly, I thank Jesus Christ, my Lord and Saviour. He has showered me with His grace and mercy, and without Him I am nothing. He has given me life, and with it the strength and perseverance necessary to complete this research and to bring Him glory.

Abstract

Chapter 1 introduces the tools and mechanics necessary for this report. Basic definitions and topics of graph theory which pertain to the report and discussion of automorphic decompositions will be covered in brief detail. An *automorphic decomposition* \mathcal{D} of a graph H by a graph G is a G -decomposition of H such that the intersection graph $I(\mathcal{D}) \cong H$. H is called the *automorphic host*, and G is the *automorphic divisor*. We seek to find classes of graphs that are automorphic divisors, specifically ones generated cyclically.

Chapter 2 discusses the previous work done mainly by Beeler. It also discusses and gives in more detail examples of automorphic decompositions of graphs. Chapter 2 also discusses labelings and their direct relation to cyclic automorphic decompositions. We show basic classes of graphs, such as cycles, that are known to have certain labelings, and show that they also are automorphic divisors.

In Chapter 3, we are concerned with 2-regular graphs, in particular rC_m , r copies of the m -cycle. We seek to show that rC_m has a ρ -labeling, and thus is an automorphic divisor for all r and m . We discuss methods including Skolem type difference sets to create cycle systems and their correlation to automorphic decompositions.

In the Appendix, we give classes of graphs known to be graceful and our java code to generate ρ -labelings on rC_m .

Chapter 1 Introduction

In this chapter we discuss necessary definitions and ideas from the field of graph theory that will be used extensively throughout this text. We also discuss basic definitions about automorphic decompositions, which is the focus of this report.

1.1 Graph Theory

A *graph* G is an ordered pair $(V(G), E(G))$, where $V(G)$ is a set called the *vertex set*, and $E(G)$ is a set of 2 element subsets of $V(G)$ called the *edge set*. An *edge* $e = \{x, y\}$, where $x, y \in V$, will be denoted xy . If $xy \in E(G)$, then we say that vertices x and y are *adjacent*. A vertex v is *incident* with an edge e if and only if e contains the vertex v . A graph G is *finite* if and only if V and E are finite sets. A graph G is *simple* if it contains no multiple edges and no loops. A *loop* is an edge that connects a vertex to itself. In this report all graphs will be finite simple graphs.

Given a graph G , and $v \in V(G)$, we define the *neighborhood* of v , $N(v)$ as the set of all vertices that form an edge with v . The *degree* of v , denoted $d_G(v)$ or $d(v)$, is $|N(v)|$, the cardinality of neighborhood of v . A graph where every vertex has the same degree k is called a k -regular graph.

Two graphs G and H are said to be *isomorphic* if there exists a bijection $f : V(G) \rightarrow V(H)$ such that $f(u)f(v) \in E(H)$ if and only if $uv \in E(G)$. If G and H are isomorphic graphs, we write $G \cong H$.

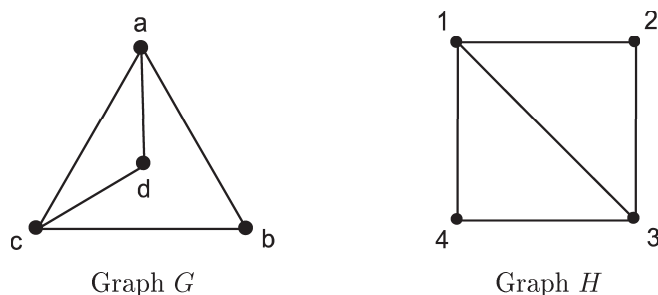


Figure 1.1.1: Two isomorphic simple graphs G , H on four vertices.

Figure 1.1.1 shows two graphs that are isomorphic to each other. Graph G has vertex set $V(G) = \{a, b, c, d\}$ and edge set $E(G) = \{\{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{c, d\}\}$. Graph H has vertex set $V(H) = \{1, 2, 3, 4\}$ and edge set $E(H) = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{3, 4\}\}$. The bijection f from G to H is $f(a) = 1$, $f(b) = 2$, $f(c) = 3$, $f(d) = 4$. We observe that this bijection maps edges to edges and keeps the necessary structure.

Given a graph G , let $n(G)$ denote the number of vertices of G , and $e(G)$ denote the number of edges in G . The *order* of the graph is $n(G)$. A graph of order n with exactly one edge between every pair

of vertices is called the *complete graph* and is denoted K_n . A complete graph on four vertices, K_4 , is seen in example 1.1.3.

A graph G is said to be an *edge induced subgraph* of the graph H if $V(G) \subseteq V(H)$ and $E(G) \subseteq E(H)$.

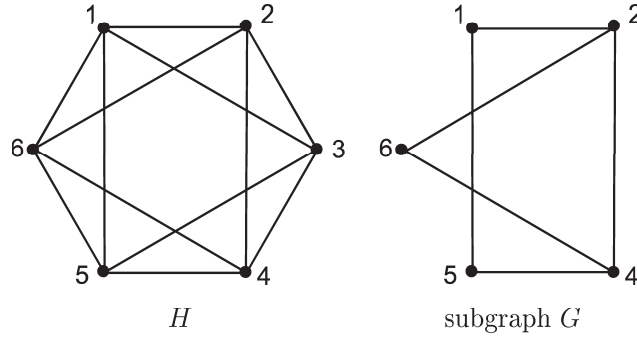


Figure 1.1.2: G a subgraph of H

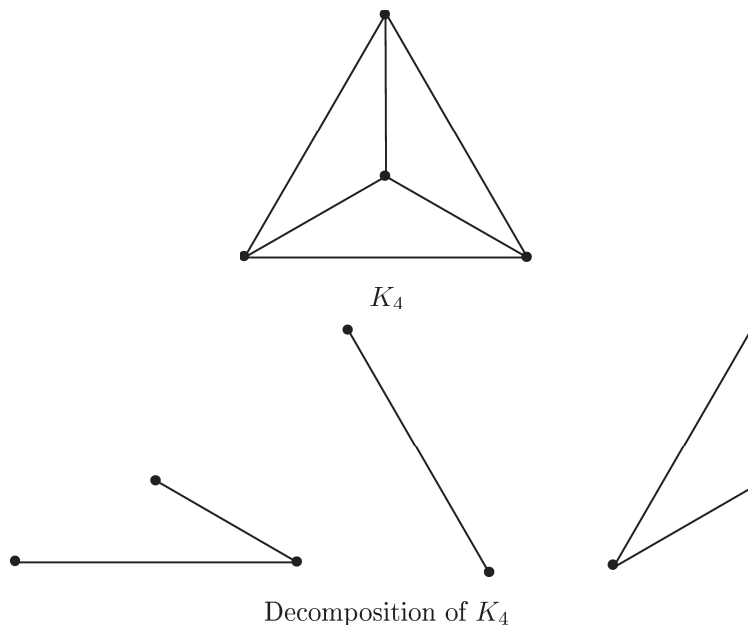
Figure 1.1.2 shows an example of an edge induced subgraph G of H . We note that not every edge of H incident to vertices of G needs to be included in the subgraph G .

A *walk* on a graph is an alternating series of vertices and edges, which begins and ends with a vertex. A *trail* is a walk in which no edges are repeated. A *path* is defined as being a trail with no repeated vertices, except possibly the first and the last ones. A *cycle* is a path in which the first and the last vertices are the same. We represent the isomorphism class of cycles on m vertices as C_m . An example of a C_8 can be seen in Figure 1.2.3. A subgraph G' of the graph G is *connected* if given any two vertices $u, v \in V(G')$, there exists a path from u to v . A connected subgraph G' of the graph G is called a *component* if G' is not contained in any larger connected subgraph of G . A graph G is connected if it has only one component. A *tree* is a simple connected graph without any cycles. An example of a tree can be seen in Figure 1.3.1.

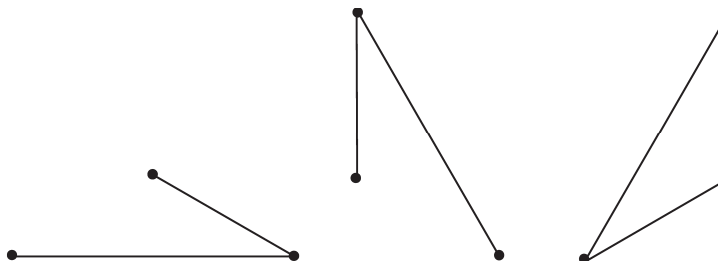
A *decomposition* $D = [H_1, H_2, \dots, H_r]$ of the graph H is a collection of subgraphs H_1, H_2, \dots, H_r of H such that $[E(H_1), E(H_2), \dots, E(H_r)]$ is a partition of $E(H)$. The subgraphs H_1, H_2, \dots, H_r are called the *blocks* of the decomposition. If G is a graph, then a *G -decomposition* of the graph H is a decomposition in which each block is isomorphic to G .

We see in Example 1.1.3 and Example 1.1.4 two different decompositions of the same graph K_4 . We note that Example 1.1.4 is a G -decomposition because each partition is isomorphic to one another. We also note that all edges in the host graph K_4 must lie in one of the blocks of the decomposition.

Example 1.1.3 A simple decomposition of K_4 into three non-isomorphic subgraphs.



Example 1.1.4 A G -decomposition of K_4 , where G is a path of length 2.



Graph decompositions are of special interest to many people and have seen use in many fields, including mathematics, computer science, bioinformatics, and internet communications. We now list examples of problems, their fields of study, and a brief description.

- Oberwolfach problem [6]: The problem is one of decomposing the complete graph K_n , n odd, into 2-factors each of which is isomorphic to a given 2-factor Q . In the case of n even, one is asked to decompose $K_n - I$, the complete graph minus a 1-factor into 2-factors isomorphic to Q .
- Hamilton-Waterloo problem [6]: The Hamilton-Waterloo problem is determining if there exists a 2-factorization of K_{2n+1} with r 2-factors isomorphic to a 2-factor R , and s 2-factors isomorphic to a 2-factor S such that $r + s = n$.

- Artificial intelligence: Graph decomposition is used to help solve constraint satisfaction problems (CSPs) [3].
- Bioinformatics: Graph decomposition techniques have been used in DNA sequencing with nanopores [4].
- Internet communications: Graph decompositions are used to analyze and characterize traffic activity graphs (TAGs) [11].

1.2 Automorphic Graph Decompositions

In this section, we will introduce automorphic graph decompositions and give some basic examples. We will also discuss the constraints and parity conditions.

New graphs can also be made from decompositions. The intersection graph, $I(\mathcal{D})$, of a G -decomposition \mathcal{D} is the graph with blocks of \mathcal{D} as vertices, and edges $e = \{a, b\} \in E(I(\mathcal{D}))$ with $a, b \in \mathcal{D}$, if and only if a and b share a common vertex in H .

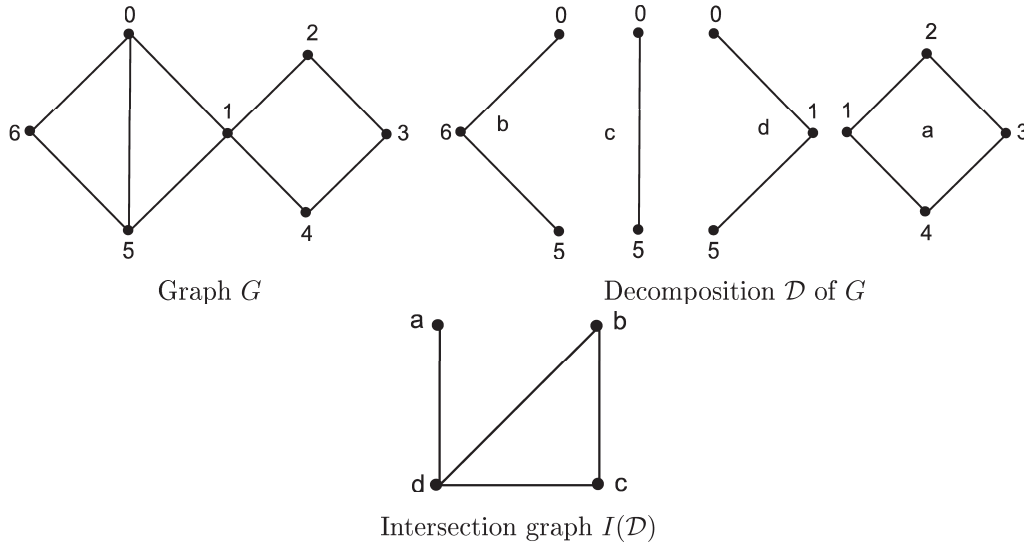


Figure 1.2.1

Definition 1.2.2 A G -decomposition \mathcal{D} of a graph H is an *automorphic decomposition* of the graph H if and only if the intersection graph $I(\mathcal{D}) \cong H$. G is called the *automorphic divisor*, and H is called the *automorphic host*.

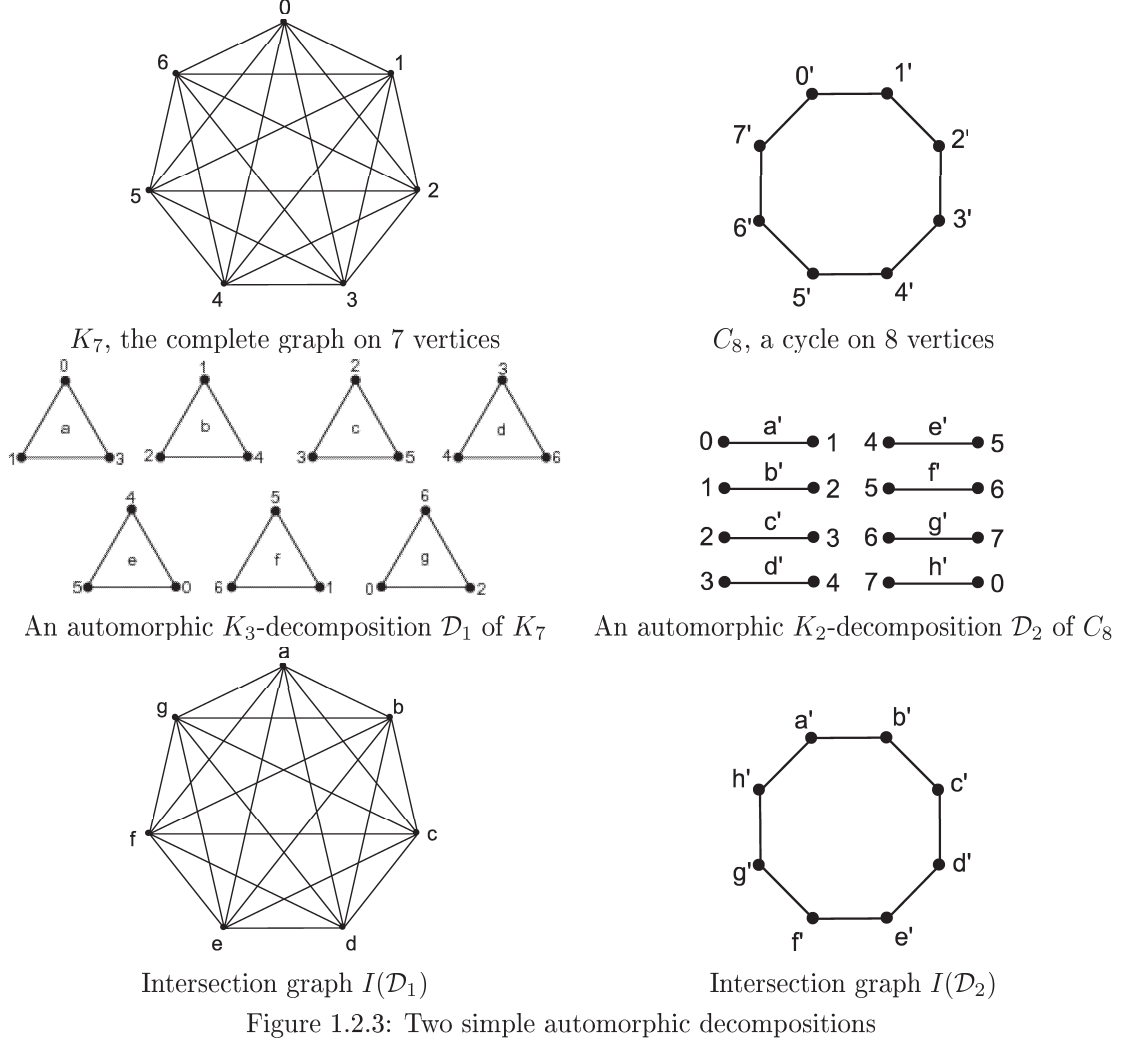


Figure 1.2.3 demonstrates two automorphic decompositions, one of K_7 and another of C_8 . We note by examination that blocks a , b , c , d , e , f , and g share a common vertex with every other block. Thus the intersection graph $I(\mathcal{D}_1) \cong K_7$. It is also clear the the decomposition of C_8 into 8 copies of K_2 yields an automorphic decomposition, as each block, K_2 , is adjacent to exactly two other blocks.

The main problem we approach in this is to determine if a given graph G is an automorphic divisor of some host graph H . Some basic parity conditions exist, and are listed below.

Lemma 1.2.4 (Beeler and Jamison[2]) Suppose \mathcal{D} is an automorphic decomposition of H . Then $e(H) = n(H)e(G)$.

Proof We note that by definition of $I(\mathcal{D})$, $n(I(\mathcal{D}))$ is the number of G -blocks in \mathcal{D} . Since \mathcal{D} partitions the edges of H into G -blocks, we have $n(I(\mathcal{D})) = e(H)/e(G)$. For \mathcal{D} to be automorphic, $n(I(\mathcal{D})) = n(H)$ must be true. Thus we have $e(H) = n(H)e(G)$.

Theorem 1.2.5 Suppose \mathcal{D} is an automorphic G -decomposition of H . Then $n(H) \geq 2e(G) + 1$.

Proof For \mathcal{D} to be automorphic, we must have $n(H) = n(I(\mathcal{D}))$. Thus there are $n(H)$ blocks in \mathcal{D} and each is isomorphic to G . In particular, each block contains $e(G)$ edges. Thus the host graph H will have exactly $e(H) = n(H)e(G)$ edges. The graph has no more edges than the complete graph on $n(H)$ vertices which has $\frac{n(H)(n(H)-1)}{2}$ edges. Thus $\frac{n(H)(n(H)-1)}{2} \geq e(H) = n(H)e(G)$. Consequently $n(H) \geq 2e(G) + 1$.

□

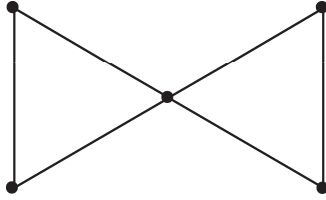


Figure 1.2.6: The bow tie graph G

Figure 1.2.6, the bow tie graph G , has $n(G) = 5$ vertices and $e(G) = 6$ edges. Thus, if G is to be a divisor of some host graph H , H must have $n(H) \geq 13$ vertices. We will show in the next section how to construct a host graph H with divisor G , when G is the bow tie graph, along with the automorphic decomposition \mathcal{D} .

One natural way to approach this problem is through circulants, and using graceful labelings. In the next section, we look at some basic information and definitions about graph labelings.

1.3 An introduction to graph labelings

Graph labelings are of special interest to the study of decompositions of graphs. A dynamic survey of graph labelings can be found in [10]. This branch of graph theory is traced back to Alex Rosa and his 1967 paper [12]. In this paper, he introduced α -, β -, and ρ -labelings.

β -labelings are now more commonly known as graceful labelings, popularized by Golomb [10]. β -labelings were introduced as a means to solve the famous conjecture by Ringel [10], that the complete graph on $2n + 1$ vertices, K_{2n+1} can be decomposed into $2n + 1$ subgraphs that are all isomorphic to a given tree with n edges. Labelings have also seen application in sports tournament scheduling. We now describe labelings and valuations on graphs in more detail.

A *vertex labeling* f on a graph G assigns elements of an algebraic group to vertices on G . In this report, we label our graphs with the integers modulo n , represented by \mathbb{Z}_n . We follow notation in

[2] to describe the “positive” elements of \mathbb{Z}_n , as $\mathbb{Z}_n^+ = \{x \in \mathbb{Z}_n : 0 < x < \frac{n}{2}\}$. We also define the *positive difference between x and y* as

$$|x - y|_n = \begin{cases} |x - y| & \text{if } 0 \leq |x - y| \leq \frac{n}{2} \\ n - |x - y| & \text{if } \frac{n}{2} < |x - y| < n \end{cases}.$$

Suppose we have a vertex labeling f on a graph G . Given \mathbb{Z}_n , the induced edge label of edge $ab \in E(G)$ is given by the positive difference between $f(a)$ and $f(b)$ as defined above. For example, given a graph $G = \mathbb{Z}_{17}$, and vertices $a, b \in V(G)$ with $f(a) = 1$ and $f(b) = 14$, we have edge ab with the induced edge label $|f(a) - f(b)|_{17} = |-13|_{17} = 17 - |-13| = 4$.

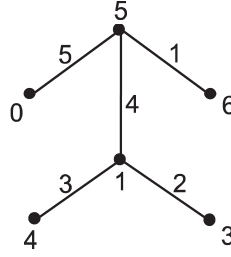


Figure 1.3.1: Induced edge labeling on a tree, given \mathbb{Z}_{11} .

Figure 1.3.1 demonstrates an induced edge labeling on the tree given. We note that the vertex labels used are from the set $\{0, 1, \dots, 6\}$, and the edge labels induced are $\{1, 2, \dots, 5\}$. This type of labeling is called a graceful labeling, and has certain properties that are very important to us, which will be discussed later.

Rosa’s work with graceful labelings to decompose K_{2n+1} is actually quite useful to us in approaching our problem. A labeling f on a graph G with t edges is *graceful* if f is an injection from the vertices of G to the set $\{0, 1, \dots, t\}$ such that each edge $xy \in G$, when assigned the label $|f(x) - f(y)|$ is distinct. Now we develop the vertex labels cyclically, using $P = (0, 1, \dots, 2t)$ as our permutation. Thus the edge $xy \in G$ will be taken to $P(x)P(y) = (x + 1)(y + 1) \pmod{2t + 1}$. We apply the permutation $2t$ times, and thus end up with $2t + 1$ graphs which are all isomorphic to each other. Rosa showed in [12] that the union of these graphs is K_{2t+1} , since every edge difference is covered $2t + 1$ times, and thus we have a G -decomposition of K_{2t+1} . It is shown in [2] that the intersection graph of the decomposition must be isomorphic to the complete graph K_{2t+1} . Therefore, if we have a graph G with a graceful labeling, we are guaranteed to have a host graph $H = K_{2t+1}$ with an automorphic divisor G . We direct the reader to Table 2 in the appendix, which lists all graphs known to have a graceful labeling. Table 2 was originally found in [10].

Suppose we have a set S which contains numbers in \mathbb{Z}_n , all lying between 1 and $n/2$, inclusive. The *circulant* $C_n(S)$ is a graph with vertex set $V = \mathbb{Z}_n$, and edges between vertices $x, y \in V$ if and only if $|f(x) - f(y)|_n \in S$. This notion is useful when we develop a graph G with t edges cyclically with the permutation $P = (0, 1, \dots, 2t)$ discussed above into its host graph H . For example, suppose that we had the circulant $H = C_{13}(\{1, 2, 4\})$. Then the vertices of H are labeled $0, 1, \dots, 12$. Also,

there is an edge between two vertices $a, b \in V(H)$ if and only if $|f(a) - f(b)|_n \in \{1, 2, 4\}$. So for this example the vertex labeled 1 would be adjacent to vertices labeled 0, 2, 3, 12, 5, and 10. Thus, in this case, H could have a G -decomposition, where G is a graph with edge labels 1, 2, and 4 and vertex labels in $\{0, 1, \dots, 12\}$.

Even though graphs that are graceful are guaranteed to have an automorphic host, there are other types of labelings that are less restrictive that can also produce an automorphic decomposition. Following is one such labeling, and is described in [2].

Definition 1.3.2 Let $G = (V, E)$ be a graph. A \mathbb{Z}_n -labeling of G is an injective map $f : V \rightarrow \mathbb{Z}_n$. Any \mathbb{Z}_n -labeling f of G induces an edge labeling f^* on E by $f^*(xy) = |f(x) - f(y)|_n$ for all edges $xy \in E$. We denote the set of edge labels induced by f by:

$$f^*(E) = \{|f(x) - f(y)|_n : x, y \in E\}.$$

Similarly, this concept can be extended to all pairs of vertices with:

$$f^*(G) = \{|f(x) - f(y)|_n : x, y \in V, x \neq y\}.$$

Note that $f^*(E) \subseteq f^*(G)$ for all f and $G = (V, E)$. A \mathbb{Z}_n -labeling f of G is a \mathbb{Z}_n -valuation of G if and only if the induced edge labeling f^* is injective, and if n is even, the involution $\frac{n}{2}$ does not appear as an edge label. A \mathbb{Z}_n -valuation of G is said to be *closed* if $f^*(G) = f^*(E)$.

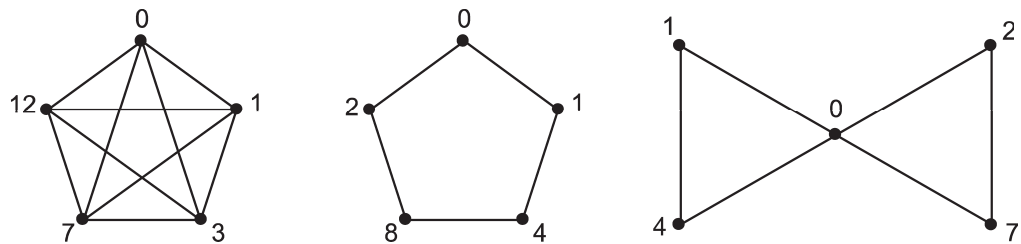
Theorem 1.3.3 (Beeler [2]) A cyclic automorphic divisor of $C_n(S)$ exists if and only if there exists a closed \mathbb{Z}_n -valuation f on G such that $S = f^*(E)$.

In other words, if we have a graph G , and we can find a closed \mathbb{Z}_n -valuation f on G such that $f^*(G) = f^*(E)$, then G is the automorphic divisor of some host graph H , which is the resulting graph developed cyclically (the circulant).

We also note that a graceful labeling on a graph G with t vertices is also a closed \mathbb{Z}_n -valuation, where $n \geq 2t + 1$. This is easily seen because a graceful labeling f on G has edge labels $\{1, 2, \dots, t\}$ and vertex labels in the set $\{0, 1, \dots, t\}$. Thus, we must have $f^*(G) = f^*(E)$.

Because the labeling map f on a graph G is injective from the vertex set $V(G)$ onto \mathbb{Z}_n , we may assume $V(G) \subseteq \mathbb{Z}_n$ and $f(x) = x$. We make this assumption for the remainder of the report.

It is known that almost all connected graphs on up to five vertices have a graceful labeling [10]. The three graphs on five vertices that do not have a graceful labeling are K_5 , C_5 , and the bow tie graph. We provide in Figure 1.3.4 a closed \mathbb{Z}_n -labeling on these graphs, and thus, all graphs on up to five vertices are automorphic divisors.



Closed \mathbb{Z}_{25} -valuation on K_5 Closed \mathbb{Z}_{11} -valuation on C_5 Closed \mathbb{Z}_{13} -valuation on the bow tie

Figure 1.3.4: Closed \mathbb{Z}_n -valuations of non-graceful graphs on five vertices

We now show a simple example of a graph $G = C_4$ with a graceful labeling, and the decomposition of the larger automorphic host $H = K_9$.

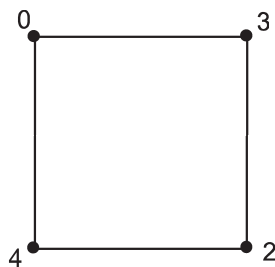


Figure 1.3.5: A graceful labeling on C_4

Figure 1.3.5 shows a graceful labeling on C_4 and thus gives us edge differences 1, 2, 3 and 4. From this labeling, we develop the circulant $C_9(\{1, 2, 3, 4\}) = K_9$. Figure 1.3.6 shows the resulting components of a C_4 -decomposition of K_9 induced by our graceful labeling.

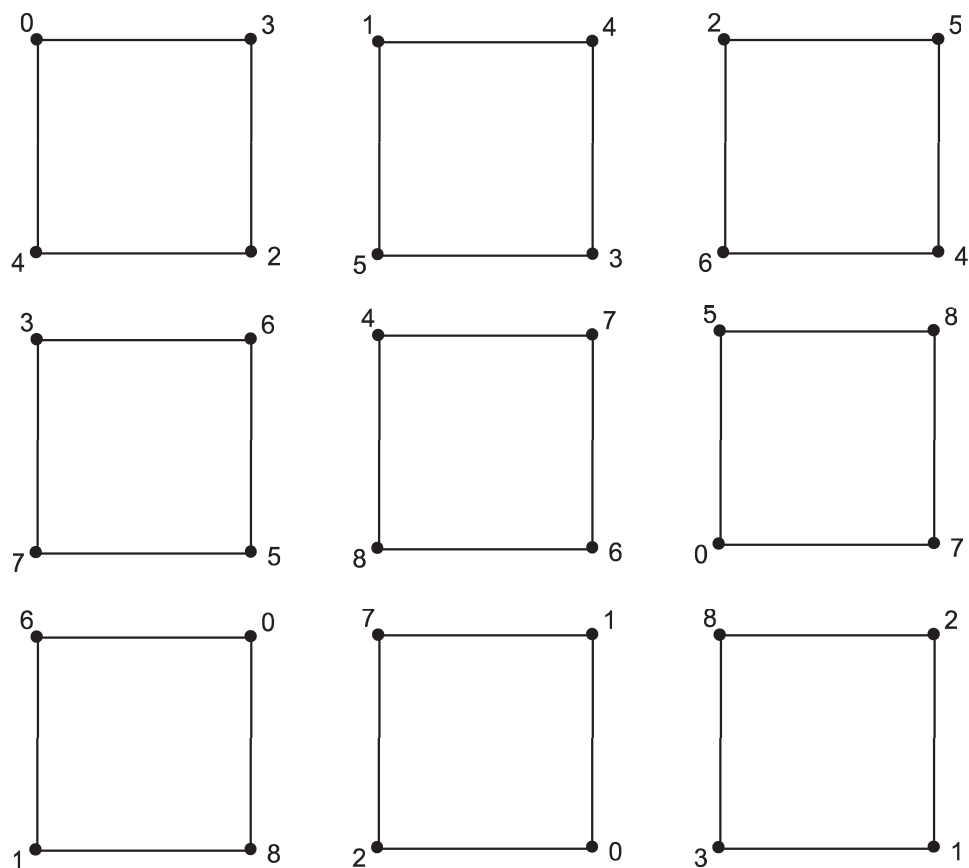


Figure 1.3.6: \mathcal{D} , a C_4 -decomposition of K_9

Upon inspection of Figure 1.3.6, we see that $I(\mathcal{D}) \cong K_9$. Therefore, C_4 is a cyclic automorphic divisor of the host graph K_9 . This method of finding appropriate labelings that induce an automorphic decomposition is the main focus of this report.

Chapter 2 Graph Labelings

Table 2 in Appendix A shows a summary of results on graphs with graceful labelings. Again, we note that a graceful labeling on G is a closed \mathbb{Z}_n -valuation, where $n \geq 2e(G) + 1$. We see that in our table, cycles of length m , C_m , are graceful if and only if $m \equiv 0$ or $3 \pmod{4}$.

The search to find automorphic divisors of some host led us to ask if are all cycles autmorphic divisors, and if so, are all 2-regular graphs automorphic divisors. We now describe one approach to show that every cycle C_m has a closed \mathbb{Z}_n -valuation, for $n = 2m + 1$.

Theorem 2.1 C_n has a closed \mathbb{Z}_{2n+1} -valuation.

Proof: We will show that a closed \mathbb{Z}_{2n+1} -valuation exists for C_n by constructing a cycle on n vertices, that uses positive differences from 1 to n , and only vertices labeled from 0 to $2n + 1$.

Suppose $n \equiv 2 \pmod{4}$. We construct our cycle iteratively. We start with two edges, $e_{0,1} = \{\frac{1}{2}n, \frac{3}{2}n + 1\}$ and $e_{0,2} = \{\frac{1}{2}n + 1, \frac{3}{2}n\}$. These edges have positive differences of n and $n - 1$. For $j = 1, \dots, \frac{n-6}{4}$ we add edges $e_{j,1}, e_{j,2}, e_{j,3}, e_{j,4}$ to form 2 paths of length $\frac{n-4}{2}$ as follows:

if j is odd then

$$e_{j,1} = \left\{ \frac{n}{2} - 2j + 2, \frac{3n}{2} + 2j + 3 \right\}$$

$$e_{j,2} = \left\{ \frac{n}{2} - 2j + 3, \frac{3n}{2} + 2j + 2 \right\}$$

$$e_{j,3} = \left\{ \frac{n}{2} + 2(j + 1), \frac{3n}{2} + 1 - 2(j - 1) \right\}$$

$$e_{j,4} = \left\{ \frac{n}{2} + 1 + 2(j + 1), \frac{3n}{2} - 2(j - 1) \right\}$$

if j is even, then

$$e_{j,1} = \left\{ \frac{n}{2} - 2j, \frac{3n}{2} + 2j + 1 \right\}$$

$$e_{j,2} = \left\{ \frac{n}{2} - 2j + 1, \frac{3n}{2} + 2j \right\}$$

$$e_{j,3} = \left\{ \frac{n}{2} + 2j, \frac{3n}{2} + 1 - 2j \right\}$$

$$e_{j,4} = \left\{ \frac{n}{2} + 1 + 2j, \frac{3n}{2} - 2j \right\}$$

For each j , $e_{j,1}$ has difference $n - 4j$, $e_{j,2}$ has difference $n - 4j + 2$, $e_{j,3}$ has difference $n - 4j + 1$, and $e_{j,4}$ has difference $n - 4j - 1$. Thus the $n - 4$ total edges cover each difference in the set $\{5, 6, 7, \dots, n - 1, n\}$ exactly once.

We now show that this forms two paths of length $\frac{n-4}{2}$. Clearly $e_{0,1}$ is adjacent to $e_{1,1}$ and $e_{1,3}$; and $e_{0,2}$ is adjacent to $e_{1,2}$ and $e_{1,4}$.

Now we show that for any j and $j + 1$, $e_{j,i}$ is adjacent to $e_{j+1,i}$.

Assume j odd. Then we have

$$e_{j,1} = \left\{ \frac{n}{2} - 2j + 2, \frac{3n}{2} + 2j + 3 \right\} \text{ and}$$

$$e_{j+1,1} = \left\{ \frac{n}{2} - 2(j+1), \frac{3n}{2} + 2(j+1) + 3 \right\} = \left\{ \frac{n}{2} - 2j - 2, \frac{3n}{2} + 2j + 3 \right\}$$

$$e_{j,2} = \left\{ \frac{n}{2} - 2j + 3, \frac{3n}{2} + 2j + 2 \right\} \text{ and}$$

$$e_{j+1,2} = \left\{ \frac{n}{2} - 2(j+1) + 1, \frac{3n}{2} + 2(j+1) \right\} = \left\{ \frac{n}{2} - 2j - 1, \frac{3n}{2} + 2j + 2 \right\}$$

$$e_{j,3} = \left\{ \frac{n}{2} + 2(j+1), \frac{3n}{2} + 1 - 2(j-1) \right\} \text{ and}$$

$$e_{j+1,3} = \left\{ \frac{n}{2} + 2(j+1), \frac{3n}{2} + 1 - 2(j+1) \right\}$$

$$e_{j,4} = \left\{ \frac{n}{2} + 1 + 2(j+1), \frac{3n}{2} - 2(j-1) \right\} \text{ and}$$

$$e_{j+1,4} = \left\{ \frac{n}{2} + 1 + 2(j+1), \frac{3n}{2} - 2(j+1) \right\}$$

Now assume j even. Then we have:

$$e_{j,1} = \left\{ \frac{n}{2} - 2j, \frac{3n}{2} + 2j + 1 \right\} \text{ and}$$

$$e_{j+1,1} = \left\{ \frac{n}{2} - 2(j+1) + 2, \frac{3n}{2} + 2(j+1) + 3 \right\} = \left\{ \frac{n}{2} - 2j, \frac{3n}{2} + 2j + 5 \right\}$$

$$e_{j,2} = \{\frac{n}{2} - 2j + 1, \frac{3n}{2} + 2j\} \text{ and}$$

$$e_{j+1,2} = \{\frac{n}{2} - 2(j+1) + 3, \frac{3n}{2} + 2(j+1) + 2\} = \{\frac{n}{2} - 2j + 1, \frac{3n}{2} + 2j + 4\}$$

$$e_{j,3} = \{\frac{n}{2} + 2j, \frac{3n}{2} + 1 - 2j\} \text{ and}$$

$$e_{j+1,3} = \{\frac{n}{2} + 2(j+2), \frac{3n}{2} + 1 - 2(j)\}$$

$$e_{j,4} = \{\frac{n}{2} + 1 + 2j, \frac{3n}{2} - 2j\} \text{ and}$$

$$e_{j+1,4} = \{\frac{n}{2} + 1 + 2(j+2), \frac{3n}{2} - 2(j)\}$$

Thus we have constructed two paths of length $\frac{n-4}{2}$

Now we count the number of vertices after $j = \frac{n-2}{4} - 1$ iterations.

Consider the vertices of degree 1 (which are at the ends of the two paths).

If $j = (\frac{n-6}{4})$ is odd, then we have end edges:

$$e_{j,1} = \left\{ \frac{n}{2} - 2 \left(\frac{n-6}{4} \right) + 2, \frac{3n}{2} + 2 \left(\frac{n-6}{4} \right) + 3 \right\} = \{5, 2n\}$$

$$e_{j,2} = \left\{ \frac{n}{2} - 2 \left(\frac{n-6}{4} \right) + 3, \frac{3n}{2} + 2 \left(\frac{n-6}{4} \right) + 2 \right\} = \{6, 2n-1\}$$

$$e_{j,3} = \left\{ \frac{n}{2} + 2 \left(\frac{n-6}{4} + 1 \right), \frac{3n}{2} + 1 - 2 \left(\frac{n-6}{4} - 1 \right) \right\} = \{n-1, n+6\}$$

$$e_{j,4} = \left\{ \frac{n}{2} + 1 + 2 \left(\frac{n-6}{4} + 1 \right), \frac{3n}{2} - 2 \left(\frac{n-6}{4} - 1 \right) \right\} = \{n, n+5\}$$

Thus the vertices which have not been used are 0, 1, 2, 3, 4, $n+1$, $n+2$, $n+3$, $n+4$. We connect the end vertices $2n$ and $2n-1$ to the unused vertex 0 to form the edges $\{2n, 0\}$, and $\{2n-1, 0\}$, which cover the positive differences of 1 and 2. We connect the end vertices $n-1$ and n with the unused vertex $n+3$ to form the edges $\{n-1, n+3\}$, and $\{n, n+3\}$ which cover the positive differences 3 and 4.

If $j = \frac{n-6}{4}$ is even, then we have end edges:

$$e_{j,1} = \left\{ \frac{n}{2} - 2 \left(\frac{n-6}{4} \right), \frac{3n}{2} + 2 \left(\frac{n-6}{4} \right) + 1 \right\} = \{3, 2n-2\}$$

$$e_{j,2} = \left\{ \frac{n}{2} - 2 \left(\frac{n-6}{4} \right) + 1, \frac{3n}{2} + 2 \left(\frac{n-6}{4} \right) \right\} = \{4, 2n-3\}$$

$$e_{j,3} = \left\{ \frac{n}{2} + 2 \left(\frac{n-6}{4} \right), \frac{3n}{2} + 1 - 2 \left(\frac{n-6}{4} \right) \right\} = \{n-3, n+4\}$$

$$e_{j,4} = \left\{ \frac{n}{2} + 1 + 2 \left(\frac{n-6}{4} \right), \frac{3n}{2} - 2 \left(\frac{n-6}{4} \right) \right\} = \{n-2, n+3\}$$

Thus, the vertices which have not been used are 0, 1, 2, 2n, 2n-1, n-1, n, n+1, n+2. We connect the end vertices 3 and 4 to the unused vertex 2 to form edges {2,3} and {2,4}, which cover the positive differences 1 and 2. We connect the end vertices n+4 and n+3 to the unused vertex n to form edges {n, n+4} and {n, n+3} which cover the positive differences 3 and 4. In each case, the 4 added edges complete the desired cycle of length n.

Suppose $n \equiv 1 \pmod{4}$. We will construct our cycle iteratively in the same manner. We start with two edges $e_{0,1} = \left\{ \frac{n-1}{2}, \frac{3n+3}{2} \right\}$ and $e_{0,2} = \left\{ \frac{n+1}{2}, \frac{3n+1}{2} \right\}$, edge $e_{0,1}$ with positive difference $n-1$ and n respectively. For $j = 1, 2, \dots, \frac{n-5}{4}$ we add edges $e_{j,1}, e_{j,2}, e_{j,3}, e_{j,4}$, to form 2 paths of length $\frac{n-3}{2}$ as follows:

if j is odd then

$$e_{j,1} = \left\{ \frac{n-1}{2} + 4 \left(\frac{j+1}{2} \right), \frac{3n+3}{2} - 4 \left(\frac{j-1}{2} \right) \right\}$$

$$e_{j,2} = \left\{ \frac{n+1}{2} + 4 \left(\frac{j+1}{2} \right), \frac{3n+1}{2} - 4 \left(\frac{j-1}{2} \right) \right\}$$

$$e_{j,3} = \left\{ \frac{3n+3}{2} + 4 \left(\frac{j+1}{2} \right), \frac{n-1}{2} - 4 \left(\frac{j-1}{2} \right) \right\}$$

$$e_{j,4} = \left\{ \frac{3n+1}{2} + 4 \left(\frac{j+1}{2} \right), \frac{n+1}{2} - 4 \left(\frac{j-1}{2} \right) \right\}$$

if j is even, then

$$e_{j,1} = \left\{ \frac{3n+3}{2} - 2j, \frac{n-1}{2} + 2j \right\}$$

$$e_{j,2} = \left\{ \frac{3n+1}{2} - 2j, \frac{n+1}{2} + 2j \right\}$$

$$e_{j,3} = \left\{ \frac{3n+3}{2} + 2j, \frac{n-1}{2} - 2j \right\}$$

$$e_{j,4} = \left\{ \frac{3n+1}{2} + 2j, \frac{n+1}{2} - 2j \right\}$$

We follow the same process described for $n \equiv 2 \pmod{4}$, to verify that we have two paths of length $\frac{n-3}{2}$ using all edge differences in the set $\{n, n-1, \dots, 4\}$. We now have to add three edges to cover differences in the set $\{1, 2, 3\}$. If $j = \frac{n-5}{2}$ is even, then we note that we necessarily have endpoints labeled 2 and 3 on our paths. We connect both of them to vertex 0, to get edges $\{0, 2\}$ and $\{0, 3\}$. We also connect our two remaining vertices with an edge, which is necessarily the edge $\{\frac{3n+3}{2} - 2j, \frac{3n+1}{2} - 2j\}$. If $j = \frac{n-5}{2}$ is odd, then we necessarily have endpoints labeled 0 and $2n$ on our two disjoint paths. We connect each of these vertices with a vertex labeled 2, thus giving us edges $\{0, 2\}$ and $\{2, 2n\}$ with differences 2 and 3 respectively. Connecting endpoints labeled $\frac{n-1}{2} + 4(\frac{j-1}{2})$ and $\frac{n+1}{2} + 4(\frac{j-1}{2})$, we have edge $\{\frac{n-1}{2} + 4(\frac{j-1}{2}), \frac{n+1}{2} + 4(\frac{j-1}{2})\}$ with difference 1. Thus we have connected our two paths, completing a cycle of length n , covering differences from $\{1, 2, \dots, n\}$, and have a closed \mathbb{Z}_{2n+1} -valuation on C_n .

□

As a supplement to this proof, we provide Figure 2.2, an example when $n = 10$.

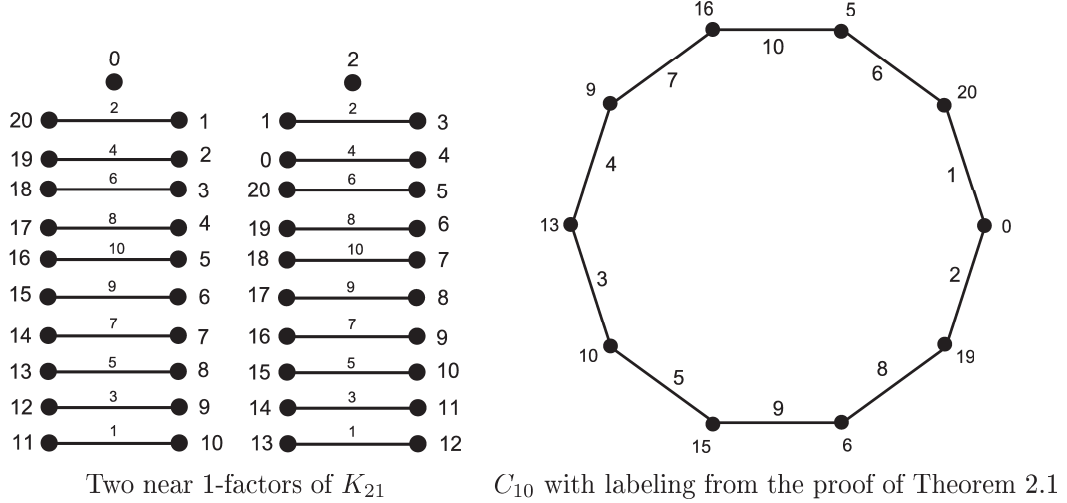


Figure 2.2: Finding a \mathbb{Z}_{21} -valuation of C_{10}

Figure 2.2 shows two near 1-factors of K_{21} . A near 1-factor G of a graph H is a subgraph of H in which every vertex except one is adjacent to exactly one other vertex. The near 1-factors in Figure 2.2 also provide the induced edge labels. Following our proof, we first select the edges $\{16, 5\}$ and $\{15, 6\}$, and continue the process described, eventually ending up with the cycle shown in Figure 2.2.

Chapter 3 ρ -labelings

There are in fact many different types of labelings that will yield an automorphic decomposition. We note that if we are to develop a labeling cyclically to achieve an automorphic decomposition, the labeling must also be a closed \mathbb{Z}_n -valuation [2]. However, since n can be very large, it is difficult to prove all graphs can have a closed \mathbb{Z}_n -valuation. We now look at a more restrictive labeling in this chapter, a ρ -labeling, which has quite a bit more structure, which allows for us to readily determine if certain classes of graphs are automorphic divisors.

What we have shown in Theorem 2.1 is that all cycles of length t have a ρ -labeling. A graph G with t edges has a ρ -labeling if there is an injection f from the vertices of G to the set $\{0, 1, \dots, 2t\}$, where if the edge labels induced by $|f(x) - f(y)|$ are a_1, a_2, \dots, a_t , then $a_i = i$ or $a_i = 2t + 1 - i$. It is noted, from Table 2 in Appendix A, that the cases C_t where $t \equiv 0, 3 \pmod{4}$ have graceful labelings [10], which are also ρ -labelings. This finding that all C_t have ρ -labelings was actually discussed in [10]. We now show that a ρ -labeling on a graph G with t edges is in fact a closed \mathbb{Z}_{2t+1} -valuation.

Theorem 3.1 A ρ -labeling on G is also a closed \mathbb{Z}_{2t+1} -valuation.

Proof A ρ -labeling on graph G uses vertex labels from the set $S = \{0, 1, \dots, 2t\}$ and must cover positive differences from 1 to t induced on the edges of G . Given any two distinct vertices with labels $f(x), f(y) \in S$, the positive difference $|f(x) - f(y)| \in \{1, \dots, t\}$. Thus we have $f^*(G) \subseteq f^*(E)$, and thus we have a closed \mathbb{Z}_{2t+1} -valuation.

□

We have shown above that all cycles C_m have ρ -labelings. Naturally, we look to find classes of graphs with ρ -labelings. Other ρ -labelings are seen in Table 1 which is a combination of results from [9] and [10].

Graph	ρ -labeling
Graphs G with at most 11 edges	ρ if $e(G) \leq 11$ (see [263] from Gallian's survey)
Graphs G with at most 8 vertices	ρ if $n(G) \leq 8$ and $G \notin \{K_6 - K_2, K_7 - K_{3,3}, K_7 - K_{1,5}, K_7 - K_2, K_7,$ $K_8 - K_{4,4}, K_8 - K_{3,4}, K_8 - K_{2,6}, K_8 - K_{1,6},$ $K_8 - K_{1,5}, K_8 - K_{2,2}, K_8 - (K_3 \cup$ $K_2), K_8 - 4K_2, K_8 - K_3, K_8 - 3K_2, K_8 - 2K_2, K_8 - K_2\}$ (El-Zanati, Vanden Eynden on Rosa-Type Labelings)
Lobsters	ρ (see [263] from Gallian's survey)
$C_r \cup C_s \cup C_t$	ρ (or more restrictive)
rC_m	ρ if $r \leq 4$
Stunted Trees T	ρ if $2 \cdot e(T) + 1$ is prime
Trees T	ρ if $n(G) \leq 55$ (El-Zanati, Vanden Eynden on Rosa-Type Labelings)
K_{p^t+1}	ρ if p is prime, $t \geq 1$ (El-Zanati, Vanden Eynden on Rosa-Type Labelings)
rC_3	ρ if $r \geq 1$ (El-Zanati, Vanden Eynden on Rosa-Type Labelings)
Graphs G with one component graceful	ρ if every other component has an α -labeling (El-Zanati, Vanden Eynden on Rosa-Type Labelings)
Graphs G with graceful bases	ρ (El-Zanati, Vanden Eynden on Rosa-Type Labelings) (Graceful base: graph resulting when all vertices of degree 1 are removed)

We now investigate 2-regular graphs and their ability to be automorphic divisors. More specifically, we look at disconnected graphs G with r components each isomorphic to C_m . G is more simply represented as rC_m . We note that a graph $G \cong rC_m$ is necessarily a 2-regular graph, since every vertex is part of a cycle, and every cycle is 2-regular.

The existence of a ρ -labeling on 2-regular graphs is a well researched topic. The following two major results about 2-regular graphs are known.

Theorem 3.2 (Aguado, et al. [1]) There exists a ρ -labeling (or more restrictive labeling) for all 2-regular graphs with at most 3 components (Aguado, El-Zanati, Hake, Stob, Yayla).

Theorem 3.3 (Donovan, et al.[8]) Every 2-regular graph consisting of at most four uniform components has a ρ -labeling (or a more restricted labeling).

We note the difference between Theorem 3.2 and Theorem 3.3, in the former different cycle lengths are allowed, and in the latter all cycle lengths must be the same. It is therefore natural to consider ρ -labelings for all 2-regular graphs. The following conjecture was made by El-Zanati and Vanden Eynden.

Conjecture 3.4 (El-Zanati and Vanden Eynden [9]) Every 2-regular graph G of size n has a ρ -labeling.

We attempt to solve the case of Conjecture 3.4 where G is the union of r copies of cycles of length m . We approach this problem using Skolem-type difference sets for cycle systems from [5]. A Skolem-type difference set of order n is a set of n triples that contain the numbers $\{1, \dots, 3n\}$ (or in the case of a hooked Skolem-type difference set, $\{1, \dots, 3n - 1, 3n + 1\}$) such that each number is not used more than once and $a + b = c$ is true for each triple (a, b, c) .

An m -cycle system of a graph G is a set \mathcal{C} of m -cycles in G whose edges partition the edge set of G . In other words, an m -cycle system of a graph G is a C_m -decomposition of G . Using these cycle systems, we want to see if we can find a ρ -labeling of G , thus proving that G can be an automorphic divisor.

Consider the induced edge labeling of a cycle C_5 given in Figure 3.1.

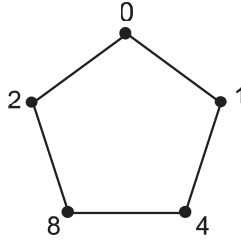


Figure 3.5: Cycle on 5 vertices with a closed \mathbb{Z}_{11} -valuation

The induced edge labels are $\{1, 3, 4, 6, 2\}$ and that $1 + 3 + 4 - 6 - 2 = 0$. This is not a special case and it is easily seen that a set of edge differences of a cycle C_m must sum to be zero, taking some differences to be negative. We choose a vertex on our cycle, and an orientation. Moving along our cycle in the direction of our orientation, we will have the edges $\{v_0, v_1\}, \{v_1, v_2\}, \dots, \{v_m, v_0\}$. We note that the edge labels given to $\{v_i, v_{i+1}\}$ by $f(v_{i+1}) - f(v_i)$ for $i = 0, 1, \dots, m$, tell us the number that must be added to the edge label $f(v_i)$ to get the next vertex label $f(v_{i+1})$. For example, given a clockwise orientation in Figure 3.1, the number -6 represents the number that must be added to the vertex labeled 8 to get the next vertex label 2. Thus, we note that if we start at vertex v_0 and finish at vertex v_0 , we must have edge differences given by $f(v_{i+1}) - f(v_i)$ that sum to 0.

We also note that these edge differences should not sum to 0 at any point before arriving back at our initial vertex v_0 . If they do sum to 0, our graph would not be two regular, since vertex v_0 would have degree greater than 2.

We state these properties, because they are in fact properties given in [5] that are necessary to have a Skolem-type cycle system. A Skolem-type cycle system is a cycle system which uses Skolem triples, or similar triples, to determine the cycles used to form the C_m -decomposition.

Suppose a graph $G = rC_m$ has a ρ -labeling. Then $f^*(E) = \{1, 2, \dots, rm\}$ and for each component r must have m labels which sum to 0. Thus, if a ρ -labeling exists for G , there must exist an r by m matrix with entries in $\{\pm 1, \pm 2, \dots, \pm rm\}$ where the sum of each row is 0, and $|x_{i,j}| \neq |y_{k,l}|$ unless $i = k$ and $j = l$. Each row represents the edge differences of a given m -cycle in rC_m . We note that this problem is in fact that of finding a cyclic m -cycle system of K_{2rm+1} , and a complete solution for this problem is given in [5]. It is important to note that this is a necessary condition, but not a sufficient one. This is because the $r \times m$ matrix consists of only edge differences, and not vertex labels. It could very well be the case that we have an $r \times m$ matrix satisfying the conditions above, but there is not a labeling on $2rm + 1$ integers that gives us r disjoint copies of C_m .

Suppose we have a permutation \mathcal{P} denoted $(0, 1, \dots, 2rm)$. Then we say that the m -cycle system \mathcal{C} is *cyclic* if for any cycle $C = (v_1, v_2, \dots, v_m)$ in \mathcal{C} , $\mathcal{P}(C) = (\mathcal{P}(v_1), \mathcal{P}(v_2), \dots, \mathcal{P}(v_m))$ is also in \mathcal{C} . In Figure 3.6 we give an example of a set of base cycles for a cyclic 3-cycle system of K_{13} . A *set of base cycles* for our m -cycle system \mathcal{C} is the minimum set of cycles which generate \mathcal{C} when the permutation \mathcal{P} is applied repeatedly to the set of base cycles.

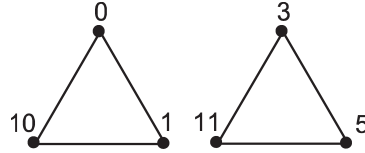


Figure 3.6: Base cycles for a cyclic 3-cycle system of K_{13} .

We want to find a ρ -labeling for $G = rC_m$. We define $t = m \cdot r$ as the number of edges in G . We look to cyclic m -cycle systems for our answer, since by definition, a cyclic m -cycle system of K_{2t+1} must use positive differences from 1 to t , and they must be used in our r copies of C_m . We now look to the results summarized in [5].

Theorem 3.7 (Bryant, Gavlas, and Ling [5]) Let m and r be integers with $m \geq 3$ and $r \geq 1$. There exists a cyclic m -cycle system of K_{2mr+1}

This theorem is proved in [5] using both normal and hooked Skolem and Langford sequences. A *Skolem sequence of order n* is a sequence $S = (s_1, s_2, \dots, s_{2n})$ of $2n$ integers such that for every $k \in \{1, 2, \dots, n\}$ there exist exactly two elements $s_i, s_j \in S$ such that $s_i = s_j = k$, and if $s_i = s_j = k$ with $i < j$, then $j - i = k$. An *extended Skolem sequence of order n* , is a Skolem sequence instead with $2n + 1$ integers, such that there is exactly one element $s_i \in ES$ such that $s_i = 0$. A *hooked Skolem sequence of order n* is an extended Skolem sequence of order n with $s_{2n} = 0$. The main part of [5] develops an r by m matrix X , in which the entries in each row sum to 0, and the absolute value of every entry, $|x_{i,j}|$, covers every number from $\{1, 2, \dots, mr\}$.

We make the following connection. Each row represents the edge differences of a copy of C_m , using the positive differences given by that row. This can create a cycle, since each row sums to 0. Since every number from $\{1, 2, \dots, mr\}$ is covered by $|x_{i,j}|$, every positive difference is covered. This appears to almost answer our question, giving us a ρ -labeling of rC_m , however each row of X only represents edge differences that are covered by the r copies of C_m . We must also find a vertex labeling that induces these edge differences, such that each of the r copies of C_m are disjoint (no vertex labels used more than once).

The next obvious step we took was to write a program, to generate X as described in [5] and use some sort of brute force method to generate disjoint cycles. Our program generates Skolem sequences, hooked Skolem sequences, Langford sequences, and hooked Langford sequences to find X . It is noted that the constructions from [5], [6], and [12] were used. We then attempt to generate the vertex labeling of cycles one at a time, attempting to see if we can find a vertex labeling of rC_m that has the given set of edge differences.

The program generates the first cycle of length m using the first row of X representing our edge differences, starting at vertex label 0. The rest of the cycle is determined by the differences given by X because the edge difference implies the next vertex label. Each subsequent cycle is generated by choosing the lowest vertex label that creates a cycle that is disjoint with all previously generated cycles. If the program fails to find a disjoint set of cycles, we change the starting vertex label of the first cycle, going all the way up to rm . If that still does not work, we also iterate which cycle we start with labeling first.

The code we used can be found in Appendix B. Using our code, we have found many ρ -labelings on rC_m . The following results have been found using our code.

Theorem 3.8 If $m \equiv 0 \pmod{4}$, $m \leq 96$ and $r \leq 99$, there exists a ρ -labeling of rC_m .

Theorem 3.9 If $m \equiv 3 \pmod{4}$, $m \leq 99$ and $r \leq 33$, there exists a ρ -labeling of rC_m .

Theorem 3.10 If $m \equiv 3 \pmod{4}$, $r \equiv 0$ or $1 \pmod{4}$, $m \leq 99$, and $r \leq 97$, there exists a ρ -labeling of rC_m .

The proofs to Theorems 3.8, 3.9, and 3.10 are proofs by existence, and can be generated by our code found in Appendix B. We also note that our code has generated many ρ -labelings of rC_m where $m \equiv 1$, or $2 \pmod{4}$, and these results are found in Table 3 and Table 4 in Appendix A.

Chapter 4 Conclusion

Automorphic decompositions tie several problems in graph theory together. Namely those of graph labeling and decomposition. Our results remind us that although computational power is strong, we still need a “graceful” way to solve our problems completely.

We conjecture from our results that a ρ -labeling of rC_m is likely to exist for all r , and m . This being said, we have shown for

- $m \equiv 0 \pmod{4}$, $m \leq 96$ and $r \leq 99$,
- $m \equiv 3 \pmod{4}$, $m \leq 99$ and $r \leq 33$, and
- $m \equiv 3 \pmod{4}$, $r \equiv 0$ or $1 \pmod{4}$, $m \leq 99$, and $r \leq 97$

that rC_m has a ρ -labeling and is also an automorphic divisor of the host K_{2rm+1} . This is just one class of graphs that are shown be automorphic divisors. It still needs to be shown that all 2-regular graphs have a ρ -labeling, and thus are automorphic divisors. We note that this a different method than Skolem-type difference sets for cycle systems will have to be used, since the graph may consist of cycles of varying lengths.

We refer the reader to [2] for more topics on automorphic decompositions, and more ideas for future research.

There appears to be much more to be done in research for automorphic decompositions. Questions that still need to be answered include:

- Is every graph an automorphic divisor for some suitable host?
- Do all 2-regular graphs have a ρ -labeling?
- How can we generate non-cyclic autmorphic hosts? Are there infinite families of them?

Work that will be done in the very near future involves investigating the cases where $m \equiv 0$ or $3 \pmod{4}$ to find a reason why the computer program can quickly find a ρ -labeling for these cases. Figuring this out would lead to another infinite family of graphs that can be automorphic divisors, along with furthering the work in [8].

Bibliography

- [1] Aguado, A., et al., *On ρ -labeling the union of three cycles*. Australasian Journal of Combinatorics, 2007. 37: p. 155-170.
- [2] Beeler, R.A. and R.E. Jamison, *Automorphic Decompositions of Graphs*. Graphs and Combinatorics, 2010. 27(2): p. 149-160.
- [3] Bliiek, C., B. Neveu, and G. Trombettoni, *Using Graph Decomposition for Solving Continuous CSPs*, in Principles and Practice of Constraint Programming — CP98, M. Maher and J.-F. Puget, Editors. 1998, Springer Berlin / Heidelberg. p. 102-116.
- [4] Bokhari, S.H. and J.R. Sauer, A parallel graph decomposition algorithm for DNA sequencing with nanopores. Bioinformatics, 2005. 21(7): p. 889-896.
- [5] Bryant, D., H. Gavlas, and A.C.H. Ling, *Skolem-type Difference Sets for Cycle Systems*. The Electronic Journal of Combinatorics, 2003. 10.
- [6] Colbourn, C.J. and J.H. Dinitz, *Handbook of Combinatorial Designs*. 2nd ed. Discrete Mathematics and its Applications, ed. K.H. Rosen. 2006: Chapman & Hall/CRC.
- [7] Davies, R.O. and C.J. Priday, *On Langford's Problem*. The Mathematical Gazette, 1959. 43(346): p. 250-255.
- [8] Donovan, D., et al., *Labelings of unions of up to four uniform cycles*. Australasian Journal of Combinatorics, 2004. 29: p. 323-336.
- [9] El-Zanati, S.I. and C. Vanden Eynden, *On Rosa-type labelings and cyclic graph decompositions*. Mathematica-Slovaca, 2009. 59: p. 1-18.
- [10] Gallian, J.A., *A Dynamic Survey of Graph Labeling*. The Electronic Journal of Combinatorics, 2010. 17.
- [11] Jin, Y., E. Sharafuddin, and Z.-L. Zhang, Unveiling core network-wide communication patterns through application traffic activity graph decomposition, in Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems. 2009, ACM: Seattle, WA, USA. p. 49-60.
- [12] Rosa, A., *On certain valuations of the vertices of a graph*, in Theory of graphs (International Symposium, Rome, 1966), Gordon and Breach, New York (1967). p. 349-355.
- [13] Shirinivas, S.G., S. Vetrivel, and D. N.M.Elango, *Applications of graph theory in computer science an overview*. International Journal of Engineering Science and Technology, 2010. 2(9): p. 4610-4621.

- [14] Simpson, J.E., *Langford sequences; perfect and hooked*. Discrete Mathematics, 1983. 44(1): p. 97-104.
- [15] West, D.B., *Introduction to Graph Theory*. 2nd ed. 2000: Prentice Hall.

Appendix A

Table 2: Summary of Graceful Results (From Gallian's survey)

Notes about Table 2: The letter G indicates that the graphs in that class are known to be graceful, a ? indicates that the gracefulness of the graphs in the class is an open problem, ? after a G indicates that the graphs have been conjectured to be graceful.

Graph	Graceful
trees	G if ≤ 35 vertices G if symmetrical G if at most 4 end-vertices G? Ringel-Kotzig G Caterpillars G? lobsters
cycles C_n	G iff $n \equiv 0, 3 \pmod{4}$
Wheels W_n	G
helms	G
webs	G
gears	G
cycles with P_k -chord	G
C_n with k consecutive chords	G if $k = 2, 3, n - 3$
unicyclic graphs	G? iff $G \neq C_n, n \equiv 1, 2 \pmod{4}$
P_n^k	G if $k = 2$
$C_n^{(t)}$	$n = 3$ G iff $t \equiv 0, 1 \pmod{4}$ G? if $nt \equiv 0, 3 \pmod{4}$ G if $n = 6, t$ even G if $n = 4, t > 1$ G if $n = 5, t > 1$ G if $n = 7$ and $t \equiv 0, 3 \pmod{4}$ G if $n = 11$
triangular snakes	G iff number of blocks $\equiv 0, 1$
K_4 -snakes	?
quadrilateral snakes	G
crowns $C_n \odot K_1$	G
$C_n \odot P_k$	G
grids $P_m \times P_n$	G
prisms $C_m \times P_n$	G if $n = 2$ G if m even G if m odd and $3 \leq n \leq 12$ G if $m = 3$ G if $m = 6$ G if $m \equiv 2 \pmod{4}$ and $n \equiv 3 \pmod{4}$

Table 2: Summary of Graceful Results continued

Graph	Graceful
$K_m \times P_n$	G if $(m, n) = (4, 2), (4, 3), (4, 4), (4, 5), (5, 2)$ not G if $(m, n) = (3, 3), (6, 2), (7, 2), (8, 2), (9, 2), (10, 2)$ not G? for $(m, 2)$ with $m > 5$
$K_{m,n} \odot K_1$	G
torus grids $C_m \times C_n$	G if $m \equiv 0 \pmod{4}$, n even not G if m, n odd (parity condition)
vertex-deleted $C_m \times P_n$	G if $n = 2$
edge-deleted $C_m \times P_n$	G if $n = 2$
Möbius ladders M_n	G
stacked books $S_m \times P_n$	$n = 2$, G iff $m \not\equiv 3 \pmod{4}$
n -cube $K_2 \times K_2 \cdots \times K_2$	G if m even
$K_4 \times P_n$	G
K_n	G if $n = 2, 3, 4, 5$
$K_{m,n}$	G iff $n \leq 4$
$K_{1,m,n}$	G
$K_{1,1,m,n}$	G
windmills $K_n^{(m)}$ ($n > 3$)	G if $n = 4, m \leq 100$ G? if $n = 4, m \geq 4$ G if $n = 4, 4 \leq m \leq 22$ not G if $n = 4, m = 2, 3$ not G if $(m, n) = (2, 5)$ not G if $n > 5$
$B(n, r, m)$ $r > 1$	G if $(n, r) = (3, 2), (4, 3)$ G $(n, 4, m) = (5, 2, 2)$ not G for $(n, 2, 2)$ for $n > 5$
mK_n	G iff $m = 1, n \leq 4$
$C_s \cup P_n$? G iff $s + n \geq 7$ G if $s = 3, s = 4, s = 5$ G if $s > 4, n = 2$ G if $s = 2n + 1$ G if $s = 2k, n \geq k + 1$
$C_p \cup C_q$	G iff $p + q \equiv 0, 3 \pmod{4}$

Table 2: Summary of Graceful Results continued

Graph	Graceful
$C_n \cup K_{p,q}$	for $n > 8$ G iff $n \equiv 0, 3 \pmod{4}$ G $C_6 \times K_{1,2n+1}$ G $C_3 \times K_{m,n}$ iff $m, n \geq 2$ G $C_4 \times K_{m,n}$ iff $(m, n) \neq (1, 1)$ G $C_7 \times K_{m,n}$ G $C_8 \times K_{m,n}$ G
$K_i \cup K_{m,n}$	G $2 \leq m_i < n_i$
$\bigcup_{i=1}^t K_{m_i, n_i}$	G $2 \leq m_i < n_i, m \equiv 0 \text{ or } 3 \pmod{4}, m \geq 11$
$C_m \cup \bigcup_{i=1}^t K_{m_i, n_i}$	G for connected G
$G + \overline{K_t}$	G for $n = 3, 4, 5, 7, 8, 9, 11, 12$
double cones $C_n + \overline{K_2}$	not G for $n \equiv 2 \pmod{4}$
t-point suspension $C_n + \overline{K_t}$	G if $n \equiv 0 \text{ or } 3 \pmod{12}$ not G if t is even and $n \equiv 2, 6, 10 \pmod{12}$ G if $n = 4, 7, 11$ or 19 G if $n = 5$ or 9 and $t = 2$
P_n^2	G
Petersen $P(n, k)$	G for $n = 5, 6, 7, 8, 9, 10, (n, k) = (8t, 3)$

Table 3: Results on rC_m , $m \equiv 1 \pmod{4}$

Values of r and m for which rC_m has a ρ -labeling.

r	m
5	5
6	5
7	5
8	5
9	5
10	5
11	5
12	5
13	5
14	5
15	5
16	5
17	5
18	5
19	5
6	9
8	9
9	9
11	9
12	9
14	9
15	9
17	9
19	9

r	m
5	13
8	13
9	13
12	13
13	13
15	13
17	13
18	13
5	17
8	17
11	17
12	17
15	17
19	17
5	21
8	21
11	21
5	25
8	25
11	25
5	29
8	29

Table 4: Results on rC_m , $m \equiv 2 \pmod{4}$

Values of r and m for which rC_m has a ρ -labeling.

r	m
5	6
6	6
7	6
8	6
9	6
10	6
11	6
12	6
13	6
14	6
15	6
16	6
17	6
18	6
19	6
5	10
6	10
7	10
8	10
9	10
10	10
11	10
12	10
13	10
14	10
15	10
16	10

r	m
17	10
18	10
19	10
5	14
6	14
7	14
8	14
9	14
10	14
11	14
12	14
13	14
14	14
15	14
16	14
17	14
18	14
19	14
5	18
6	18
7	18
8	18
9	18
10	18
11	18
13	18
14	18

Appendix B

Notes about Appendix B: The following code was created to implement results found from [5], [6], [7], and [14] to find the required r by m matrix for cyclic m -cycle systems. The code then attempts to generate a ρ -labeling of rC_m given the r by m matrix. The code is easily adjustable to find $m \equiv a \pmod{4}$, for $a = 0, 1, 2$ or 3 .

```

public class disjointDifference2 {

    private int[][] Y;
    /**
     * @param args
     *
     * makes disjoint difference sets to give a rC_m-decomposition
     */

    public static void main(String[] args) {
        disjointDifference2 d = new disjointDifference2();
        //initialize X, from [5]
        int[][] X;

        boolean disjoint = false;

        for(int i=2; i<20; i++){
            for(int j=3; j<30; j++){
                X=d.genX(i,j);
                disjoint = false;
                //try to find a disjoint set of cycles using
                //first by changing the way a cycle is labeled,
                //then by changing the order of the cycles that
                //are labeled.
                for(int k=0; k<=2*i*j && !disjoint;k++){
                    disjoint=false;
                    //now change the order here
                    for(int l=0; l<i*2 && !disjoint; l++){
                        if(d.isDisjoint(i, j, X, k)){
                            System.out.println("r="+i+"
\t m="+j+"\t"+"k="+k+"\t"+ "l="+l+"\t"+d.isDisjoint(i,j,X,k));
                            disjoint=true;
                        }
                        //shift the rows up one,
                        //effectively changing which row of X we start with
                        X=d.rearrange(i,j,X);
                    }
                }
            }
        }

    /**
     *
     * @param r number of copies of C_m
     * @param m length of the cycle
     * @return returns the r by m matrix X given in [5]
     */
    private int[][] genX(int r, int m){
        int mod = m%4;
        int[][] X = new int[r][m];
        int[][] Xprime = new int[r][m];
        switch (mod) {

```

```

case 0:
    //use lemma 2.1 to make matrix Y(m/4,0,r)
    X = genY1(m/4,0,r);
    //rearrange X as described in paper
    for(int i=0; i<r; i++){
        for(int j=0; j<m/2; j++){
            Xprime[i][j]=X[i][j*2];
        }
        for(int j=0; j<m/2-1; j++){
            Xprime[i][j+m/2]=X[i][m-2*(j+1)-1];
        }
        Xprime[i][m-1]=X[i][m-1];
    }
    break;
case 1:
    int modr = r%4;

    int[][] lang = this.genLangford(r-1);
    switch(modr) {
    case 0:
        X[0][0]=1; X[0][1]=-2; X[0][2]=3;
        X[0][3]=5*r-2;
        X[0][4]=-(5*r);

        for(int i=1; i<r; i++){
            X[i][0]=lang[i-1][0]+2;
            X[i][1]=lang[i-1][2]-2;
            X[i][2]=lang[i-1][1]+2;
            if(i<r/2){
                X[i][3]=5*r-2-4*i;
                X[i][4]=-(5*r-4*i);
            }
            else{
                X[i][3]=5*r-3-4*(i-r/2);
                X[i][4]=-(5*r-1-4*(i-r/2));
            }
        }
        //generate Y here
        if(m>5) {
            int[][] Y = this.genY1((m-5)/4, 5*r, r);
            for(int i=0; i<r; i++){
                for(int j=5; j<m; j++){
                    X[i][j]=Y[i][j-5];
                }
            }
        }
        break;

    case 1:
        if(r==1){
            X[0][0]=1;
            X[0][1]=-2;
            X[0][2]=3;
            X[0][3]=4;
            X[0][4]=-6;
            if(m>5) {

```

```

1);

int[][] Y = this.genY2((m-5)/4, 5,

for(int j=5; j<m; j++){
    X[0][j]=Y[0][j-5];
}

}

else{
    X[0][0]=1;
    X[0][1]=-2;
    X[0][2]=3;
    X[0][3]=5*r-1;
    X[0][4]=-(5*r+1);
    for(int i=1; i<r; i++){
        for(int j=0; j<m; j++){
            X[i][0]=lang[i-1][0]+2;
            X[i][1]=lang[i-1][2]-2;
            X[i][2]=lang[i-1][1]+2;

            if(i<r/2+1){
                X[i][3]=5*r-4*i;
                X[i][4]=-(5*r-2-4*(i-

1));

            }
            else{
                X[i][3]=5*r-5-4*(i-r/2-

1);

                X[i][4]=-(5*r-3-4*(i-

r/2-1));

            }

        }
    }
    if(m>5){
        int[][] Y = this.genY2((m-5)/4,

5*r, r);

        for(int i=0; i<r; i++){
            for(int j=5; j<m; j++){
                X[i][j]=Y[i][j-5];
            }
        }
    }
    break;

case 2:
    if(r==2){
        X[0][0]=1;
        X[0][1]=-5;
        X[0][2]=6;
        X[0][3]=7;
        X[0][4]=-9;

        X[1][0]=2;
        X[1][1]=-3;
        X[1][2]=4;

```



```

X[1][3]=8;
X[1][4]=-11;

if(m>5){
    int[][] Y = this.genY2((m-5)/4, 10,

2);

    for(int j=5; j<m; j++){
        X[0][j]=Y[0][j-5];
        X[1][j]=Y[1][j-5];
    }
}
else{
    X[r-1][0]=1;
    X[r-1][1]=-2;
    X[r-1][2]=3;
    X[r-1][3]=3*r;
    X[r-1][4]=-(3*r+2);
    for(int i=0; i<r-1; i++){
        for(int j=0; j<m; j++){
            X[i][0]=lang[i][0]+2;
            X[i][1]=lang[i][2]-2;
            X[i][2]=lang[i][1]+2;
            //
            if(i<r/2){
                X[i][3]=5*r-1-4*i;
                X[i][4]=-(5*r+1-4*i);
            }
            else{
                X[i][3]=5*r-4-4*(i-
r/2);
                X[i][4]=-(5*r-2-4*(i-
r/2));
            }
        }
    }
    if(m>5){
        int[][] Y = this.genY2((m-5)/4,
5*r, r);

        for(int i=0; i<r; i++){
            for(int j=5; j<m; j++){
                X[i][j]=Y[i][j-5];
            }
        }
    }
    break;
}
case 3:
    for(int i=0; i<r-1; i++){
        X[i][0]=lang[i][0]+2;
        X[i][1]=lang[i][2]-2;
        X[i][2]=lang[i][1]+2;
    }
    X[r-1][0]=1; X[r-1][1]=-2; X[r-1][2]=3;

```

```

        for(int i=0; i<r/2; i++){
            X[i][3]=5*r-3-4*i;
            X[i][4]=-(5*r-1-4*i);
        }
        for(int i=r/2; i<r; i++){
            X[i][3]=5*r-2-4*(i-r/2);
            X[i][4]=-(5*r-4*(i-r/2));
        }

        //generate Y here
        if(m>5){
            int[][] Y = this.genY1((m-5)/4, 5*r, r);
            for(int i=0; i<r; i++){
                for(int j=5; j<m; j++){
                    X[i][j]=Y[i][j-5];
                }
            }

            break;
        }

        //rearrange X, make into Xprime

        for(int i=0; i<r; i++){
            Xprime[i][0]=X[i][0];
            for(int j=1; j<(m+1)/2; j++){
                Xprime[i][j]=X[i][2*j-1];
            }
            for(int j=1; j<(m+1)/2-1; j++){
                Xprime[i][j+m/2]=X[i][m-j*2-1];
            }
            Xprime[i][m-1]=X[i][m-1];
        }
        break;
    }

    case 2:

        //generate first row for X
        for(int i=0; i<m && i<6; i++){
            X[0][i]=i+1;
            if(i==1 || i==3 || i==4){
                X[0][i]*=-1;
            }
            if(i==5){
                X[0][i]+=1;
            }
        }

        //if r is odd or even
        for(int j=0; j<m && j<6; j=j+1){
            for(int i=1; i<r-1; i=i+2){
                //generate columns 0 & 5
                if(j==0 || j==5){
                    X[i][j]=X[i-1][j]+5;

```

```

        X[i+1][j]=X[i][j]+7;
    }
    //generate columns 1 & 4
    if(j==1 || j==4){
        X[i][j]=X[i-1][j]-6;
        X[i+1][j]=X[i][j]-6;
    }
    //generate column 2
    if(j==2){
        X[i][j]=X[i-1][j]+7;
        X[i+1][j]=X[i][j]+5;
    }
    //generate column 3
    if(j==3){
        X[i][j]=X[i-1][j]-5;
        X[i+1][j]=X[i][j]-7;
    }
}
}
if(r%2==0){ //generate last row if it is even
    for(int j=0; j<m && j<6; j++){
        //generate columns 0 & 5
        if(j==0 || j==5){
            X[r-1][j]=X[r-2][j]+5;
        }
        //generate columns 1 & 4
        if(j==1 || j==4){
            X[r-1][j]=X[r-2][j]-6;
        }
        //generate column 2
        if(j==2){
            X[r-1][j]=X[r-2][j]+7;
        }
        //generate column 3
        if(j==3){
            X[r-1][j]=X[r-2][j]-5;
        }
    }
}
}
//if m > 6, we need to use Y
if(m>6){
    int[][] Y = new int[r][(m-6)/4];
    if(r%2==1){ //if r is odd, get Y1 (from
        //retrieve expected vector from Lemma 2.1
        Y= this.genY1((m-6)/4, 6*r, r);
    }
    else if(r%2==0){
        //retrieve expected vector from Lemma 2.2
        Y= this.genY2((m-6)/4, 6*r, r);
    }
    //put X and Y together
    for(int i=0; i<r; i++){
        for(int j=6; j<m; j++){
            X[i][j]=Y[i][j-6];
        }
    }
}

```

lemma 2.1)

```

    }
}

//rearrange X as described in paper
for(int i=0; i<r; i++){
    for(int j=0; j<m/2+1; j++){
        if(j<=1) Xprime[i][j]=X[i][j];
        else Xprime[i][j]=X[i][(j-1)*2];
    }
    for(int j=1; j<m/2-1; j++){
        Xprime[i][j+m/2]=X[i][m-2*j-1];
    }
    Xprime[i][m-1]=X[i][m-1];
}
break;
case 3:
    //Generate Skolem sequence triples
    int[][] firstRows = new int[r][3];
    firstRows = Skolem(r); //done!

    //get Y if necessary
    if(m>3){
        int[][] Y = new int[r][m-3];
        if(r%4==0 || r%4==1){
            Y = this.genY1((m-3)/4, 3*r, r);
        }
        if(r%4==2 || r%4==3){
            Y = this.genY2((m-3)/4, 3*r, r);
        }
    }

    for(int j=0; j<m; j++){
        for(int i=0; i<r; i++){
            if(j<3) X[i][j] = firstRows[i][j];
            else X[i][j] = Y[i][j-3];
        }
    }

    //rearrange X, make into Xprime
    for(int i=0; i<r; i++){
        Xprime[i][0]=X[i][0];
        for(int j=1; j<(m+1)/2; j++){
            Xprime[i][j]=X[i][2*j-1];
        }
        for(int j=1; j<(m+1)/2-1; j++){
            Xprime[i][j+m/2]=X[i][m-j*2-1];
        }
        Xprime[i][m-1]=X[i][m-1];
    }
    break;
}

return Xprime;
}

```

```

/**
 * Generates matrix Y as given in Lemma 2.1 in [5]
 * @param r
 * @param n
 * @param m
 * @return
 */
private int[][] genY1(int r, int n, int m){
    //make Y
    Y = new int[m][4*r];
    for(int i=0; i<m; i++){
        for(int j=0; j<4*r; j=j+2){
            Y[i][j]=2*m*(j/2+1)-1*(2*i+1)+n;
            if(j%4==1 || j%4==2) Y[i][j]=Y[i][j]*-1;
            Y[i][j+1]=2*m*(j/2+1)-2*i+n;
            if((j+1)%4==1 || (j+1)%4==2)
Y[i][j+1]=Y[i][j+1]*-1;
        }
    }
    return Y;
}

/**
 * Generates matrix Y as given in Lemma 2.2 in [5]
 * @param r
 * @param n
 * @param m
 * @return
 */
private int[][] genY2(int r, int n, int m){
    Y = new int[m][4*r];
    Y = genY1(r,n,m);
    //Y2 is the same as Y1, except the first two columns are
changed
    //change the first two columns
    //adjust the first two entries
    Y[0][0]=0+n;
    Y[0][1]=2+n;
    Y[0][1]=Y[0][1]*-1;
    for(int i=1; i<m; i++){
        Y[i][0]=2*m-(2*(i-1)+1)+ n ;
        Y[i][1]=2*m-(2*(i-1))+ n;
        Y[i][1]=Y[i][1]*-1;
    }
    //also change top right corner
    Y[0][4*r-1]=4*r*m+1+n;
    //if this happens to be one of the columns that needed to
be changed... change it!
    if((4*r-1)%4==1 || (4*r-1)%4==2) Y[0][4*r-1]=Y[0][4*r-1]*-
1;

    return Y;
}

/**

```

```

    * Generates a Skolem sequence and returns difference triples
    from it.
    * Note that this generates both hooked and normal Skolem
    sequences.
    * @param r order of skolem sequence
    * @return returns an r by 3 matrix of difference triples
    */
private int[][] Skolem(int r){
    int[][] triples = new int[r][3];
    int mod = r%4;
    switch(mod) {
        case 0:
            if(r==4){
                triples[0][0]=5;
                triples[0][1]=6;
                triples[0][2]=1;
                triples[1][0]=9;
                triples[1][1]=11;
                triples[1][2]=2;
                triples[2][0]=7;
                triples[2][1]=10;
                triples[2][2]=3;
                triples[3][0]=8;
                triples[3][1]=12;
                triples[3][2]=4;
            }
            //use the following construction if r>5
            if(r>5){
                //note: s=(r-1)/4; i=r (from CRC handbook);
                int rowCount = 0;
                int s=r/4;
                for(int i=1; i<=2*s; i++){
                    //row 1 in book
                    triples[rowCount][0]=4*s+i-1+r;
                    triples[rowCount][1]=8*s-i+1+r;

                    triples[rowCount][2]=Math.abs(triples[rowCount][0]-
                    triples[rowCount][1]);

                    rowCount++;

                    if(i<=s-2){
                        //row 2 in book
                        triples[rowCount][0]=i+r;
                        triples[rowCount][1]=4*s-i-1+r;

                        triples[rowCount][2]=Math.abs(triples[rowCount][0]-
                        triples[rowCount][1]);

                        rowCount++;

                        //row 3 in book
                        triples[rowCount][0]=s+i+1+r;
                        triples[rowCount][1]=3*s-i+r;

                        triples[rowCount][2]=Math.abs(triples[rowCount][0]-
                        triples[rowCount][1]);

                        rowCount++;
                    }
                }
            }
    }
}

```

```

        }
    }
    //row 4 in book
    triples[rowCount][0]=s-1+r;
    triples[rowCount][1]=3*s+r;

    triples[rowCount][2]=Math.abs(triples[rowCount][0]-
triples[rowCount][1]);
    rowCount++;

    triples[rowCount][0]=s+r;
    triples[rowCount][1]=s+1+r;

    triples[rowCount][2]=Math.abs(triples[rowCount][0]-
triples[rowCount][1]);
    rowCount++;

    triples[rowCount][0]=2*s+r;
    triples[rowCount][1]=4*s-1+r;

    triples[rowCount][2]=Math.abs(triples[rowCount][0]-
triples[rowCount][1]);
    rowCount++;

    triples[rowCount][0]=2*s+1+r;
    triples[rowCount][1]=6*s+r;

    triples[rowCount][2]=Math.abs(triples[rowCount][0]-
triples[rowCount][1]);
    rowCount++;
}

break;
case 1:
    if(r==1){
        triples[0][0]=1;
        triples[0][1]=2;
        triples[0][2]=3;
    }
    else if(r==5){
        triples[0][0]=13;
        triples[0][1]=14;
        triples[0][2]=1;
        triples[1][0]=6;
        triples[1][1]=8;
        triples[1][2]=2;
        triples[2][0]=9;
        triples[2][1]=12;
        triples[2][2]=3;
        triples[3][0]=7;
        triples[3][1]=11;
        triples[3][2]=4;
        triples[4][0]=10;
        triples[4][1]=15;
        triples[4][2]=5;
    }
}

```

```

        if(r>5){
            //if r>5, we use the following construction
            //note: s=(r-1)/4; i=r (from CRC handbook);
            int rowCount = 0;
            int s=(r-1)/4;
            for(int i=1; i<=2*s; i++){
                //row 1 in book
                triples[rowCount][0]=4*s+i+1+r;
                triples[rowCount][1]=8*s-i+3+r;

                triples[rowCount][2]=Math.abs(triples[rowCount][0]-
                triples[rowCount][1]);

                rowCount++;
                if(i<=s){
                    //row2 in book
                    triples[rowCount][0]=i+r;
                    triples[rowCount][1]=4*s-i+1+r;

                    triples[rowCount][2]=Math.abs(triples[rowCount][0]-
                    triples[rowCount][1]);

                    rowCount++;
                }
                if(i<=s-2){
                    //row 3 in book
                    triples[rowCount][0]=s+i+2+r;
                    triples[rowCount][1]=3*s-i+1+r;

                    triples[rowCount][2]=Math.abs(triples[rowCount][0]-
                    triples[rowCount][1]);

                    rowCount++;
                }
            }
            //row 4 in book
            triples[rowCount][0]=s+1+r;
            triples[rowCount][1]=s+2+r;

            triples[rowCount][2]=Math.abs(triples[rowCount][0]-
            triples[rowCount][1]);

            rowCount++;

            triples[rowCount][0]=2*s+1+r;
            triples[rowCount][1]=6*s+2+r;

            triples[rowCount][2]=Math.abs(triples[rowCount][0]-
            triples[rowCount][1]);

            rowCount++;

            triples[rowCount][0]=2*s+2+r;
            triples[rowCount][1]=4*s+1+r;

            triples[rowCount][2]=Math.abs(triples[rowCount][0]-
            triples[rowCount][1]);

            rowCount++;
        }
        break;

```



```

    case 2: //must make hooked skolem sequence
        if (r==2) {
            triples[0][0]=1+r;
            triples[0][1]=2+r;
            triples[0][2]=Math.abs(triples[0][0]-
triples[0][1]);

            triples[1][0]=3+r;
            triples[1][1]=5+r;
            triples[1][2]=Math.abs(triples[1][0]-
triples[1][1]);

        }
        if (r>=6) {
            int s=(r-2)/4;
            int rowCount=0;
            for(int i=1; i<=2*s; i++){
                triples[rowCount][0]=i+r;
                triples[rowCount][1]=4*s-i+2+r;

                triples[rowCount][2]=Math.abs(triples[rowCount][0]-
triples[rowCount][1]);

                rowCount++;
            }
            for(int i=1; i<=s-1; i++){
                triples[rowCount][0]=4*s+i+3+r;
                triples[rowCount][1]=8*s-i+4+r;

                triples[rowCount][2]=Math.abs(triples[rowCount][0]-
triples[rowCount][1]);

                rowCount++;
            }
            for(int i=1; i<=s-1; i++){
                triples[rowCount][0]=5*s+i+2+r;
                triples[rowCount][1]=7*s-i+3+r;

                triples[rowCount][2]=Math.abs(triples[rowCount][0]-
triples[rowCount][1]);

                rowCount++;
            }

            triples[rowCount][0]=2*s+1+r;
            triples[rowCount][1]=6*s+2+r;

            triples[rowCount][2]=Math.abs(triples[rowCount][0]-
triples[rowCount][1]);

            rowCount++;

            triples[rowCount][0]=4*s+2+r;
            triples[rowCount][1]=6*s+3+r;

            triples[rowCount][2]=Math.abs(triples[rowCount][0]-
triples[rowCount][1]);

            rowCount++;

            triples[rowCount][0]=4*s+3+r;

```

```

        triples[rowCount][1]=8*s+5+r;

        triples[rowCount][2]=Math.abs(triples[rowCount][0]-
triples[rowCount][1]);
        rowCount++;

        triples[rowCount][0]=7*s+3+r;
        triples[rowCount][1]=7*s+4+r;

        triples[rowCount][2]=Math.abs(triples[rowCount][0]-
triples[rowCount][1]);
        rowCount++;
    }

    break;
case 3: //must make hooked skolem sequence
    if(r==3) {
        triples[2][0]=2+r;
        triples[2][1]=3+r;
        triples[2][2]=Math.abs(triples[0][0]-
triples[0][1]);

        triples[1][0]=5+r;
        triples[1][1]=7+r;
        triples[1][2]=Math.abs(triples[1][0]-
triples[1][1]);

        triples[0][0]=1+r;
        triples[0][1]=4+r;
        triples[0][2]=Math.abs(triples[2][0]-
triples[2][1]);

    }
    if(r>=6) {
        int s=(r+1)/4;
        int rowCount=0;

        for(int i=1; i<=2*s-2; i++){
            triples[rowCount][0]=4*s+i+r;
            triples[rowCount][1]=8*s-i-2+r;

            triples[rowCount][2]=Math.abs(triples[rowCount][0]-
triples[rowCount][1]);
            rowCount++;
        }
        for(int i=1; i<=s-2; i++){
            triples[rowCount][0]=i+r;
            triples[rowCount][1]=4*s-i-1+r;

            triples[rowCount][2]=Math.abs(triples[rowCount][0]-
triples[rowCount][1]);
            rowCount++;
        }
        for(int i=1; i<=s-2; i++){
            triples[rowCount][0]=s+i+1+r;
            triples[rowCount][1]=3*s-i+r;

```

```

        triples[rowCount][2]=Math.abs(triples[rowCount][0]-
triples[rowCount][1]);
                                rowCount++;
        }

        triples[rowCount][0]=s-1+r;
        triples[rowCount][1]=3*s+r;

        triples[rowCount][2]=Math.abs(triples[rowCount][0]-
triples[rowCount][1]);
                                rowCount++;

        triples[rowCount][0]=s+r;
        triples[rowCount][1]=s+1+r;

        triples[rowCount][2]=Math.abs(triples[rowCount][0]-
triples[rowCount][1]);
                                rowCount++;

        triples[rowCount][0]=2*s+r;
        triples[rowCount][1]=4*s-1+r;

        triples[rowCount][2]=Math.abs(triples[rowCount][0]-
triples[rowCount][1]);
                                rowCount++;

        triples[rowCount][0]=2*s+1+r;
        triples[rowCount][1]=6*s-1+r;

        triples[rowCount][2]=Math.abs(triples[rowCount][0]-
triples[rowCount][1]);
                                rowCount++;

        triples[rowCount][0]=4*s+r;
        triples[rowCount][1]=8*s-1+r;

        triples[rowCount][2]=Math.abs(triples[rowCount][0]-
triples[rowCount][1]);
                                rowCount++;
    }

    break;
}

//make the second column (largest values) negative
for(int i=0; i<r; i++)
    triples[i][1]*=-1;

return triples;
}

/**
 * Generates a (hooked) Langford sequence of defect 2 and returns
the difference triples
 * @param r order of Langford sequence

```

```

    * @return returns an r by 3 matrix of difference triples
    */
    private int[][] genLangford(int r){
        int[][] langford = new int[r][3];

        //create a langford sequence of order r and defect 2 (using
        Roy Davies paper: On Langford's problem II)

        //generate the sequence
        int[] sequence = new int[r*2+1];

        int mod = r%4;
        int m;
        int p; //position in sequence vector
        switch(mod){
            case 0:
                /* if r=4m
                4m - 4, ..., 2m, 4m - 2, 2m - 3, ..., 1, 4m - 1, 1, ...,
                2m - 3,
                2m, ..., 4m - 4, 4m, 4m - 3, ..., 2m + 1, 4m - 2, 2m -
                2,
                ..., 2, 2m - 1, 4m - 1, 2, ..., 2m - 2, 2m + 1, ..., 4m
                - 3,
                2m - 1, 4m. */
                m = r/4;
                p = 0; //index of sequence (position)

                if(r>0){
                    for(int i = 4*m-4; i>=2*m; i-=2){
                        sequence[p]=i;
                        p++;
                    }
                    sequence[p]=4*m-2; p++;
                    for(int i = 2*m-3; i>=1; i-=2){
                        sequence[p]=i; p++;
                    }
                    sequence[p]=4*m-1; p++;
                    for(int i = 1; i<= 2*m-3; i+=2){
                        sequence[p]=i; p++;
                    }
                    for(int i = 2*m; i <= 4*m-4; i+=2){
                        sequence[p]=i; p++;
                    }
                    sequence[p]=4*m; p++;
                    for(int i = 4*m-3; i >= 2*m+1; i-=2){
                        sequence[p]=i; p++;
                    }
                    sequence[p]=4*m-2; p++;
                    for(int i = 2*m-2; i >= 2; i-=2){
                        sequence[p]=i; p++;
                    }
                    sequence[p]=2*m-1; p++;
                    sequence[p]=4*m-1; p++;
                    for(int i = 2; i <= 2*m-2; i+=2){
                        sequence[p]=i; p++;
                    }
                }
            }

```

```

        for(int i = 2*m+1; i <= 4*m-3; i+=2){
            sequence[p]=i; p++;
        }
        sequence[p]=2*m-1; p++;
        sequence[p]=4*m; p++;
    }
    else if(r==0){
        sequence[0]=0;
    }
    break;

case 3:
    m=(r+1)/4;
    p=0;
    /* if r=4m-1
    4m - 4, ..., 2m, 4m - 2, 2m - 3, ..., 1, 4m - 1,
1, ..., 2m - 3,
2m, ..., 4m - 4, 2m - 1, 4m - 3, ..., 2m + 1, 4m - 2,
2m - 2, ..., 2, 2m - 1, 4m - 1, 2, ..., 2m - 2, 2m +
1, ...,
4m - 3. */
    for(int i = 4*m-4; i >= 2*m; i-=2){
        sequence[p]=i; p++;
    }
    sequence[p]=4*m-2; p++;
    for(int i = 2*m-3; i >= 1; i-=2){
        sequence[p]=i; p++;
    }
    sequence[p]=4*m-1; p++;
    for(int i = 1; i <= 2*m-3; i+=2){
        sequence[p]=i; p++;
    }
    for(int i = 2*m; i <= 4*m-4; i+=2){
        sequence[p]=i; p++;
    }
    sequence[p]=2*m-1; p++;
    for(int i = 4*m-3; i >= 2*m+1; i-=2){
        sequence[p]=i; p++;
    }
    sequence[p]=4*m-2; p++;
    for(int i = 2*m-2; i >= 2; i-=2){
        sequence[p]=i; p++;
    }
    sequence[p]=2*m-1; p++;
    sequence[p]=4*m-1; p++;
    for(int i = 2; i <= 2*m-2; i+=2){
        sequence[p]=i; p++;
    }
    for(int i = 2*m+1; i <= 4*m-3; i+=2){
        sequence[p]=i; p++;
    }

    break;

```

```

case 2:
    /* if r=4m-2
    1, 2m - 3, 1, 4m - 8, ..., 2m - 2, 2m - 5, ..., 3, 4m
- 3,
    2m - 3, 4m - 6, 3,..., 2m - 5, 4m - 4, 2m - 2,..., 4m
- 8,
    4m - 2, 4m - 5, ..., 2m - 1, 2m - 4, ..., 2, 4m - 6, 4m -
3,
    2, ..., 2m - 4, 4m - 4, 2m - 1, ..., 4m - 5, *, 4m - 2.
*/

m=(r+2)/4;
p=0;
if(r>6){
    sequence[p]=1; p++;
    sequence[p]=2*m-3; p++;
    sequence[p]=1; p++;
    for(int i = 4*m-8; i >= 2*m-2; i-=2){
        sequence[p]=i; p++;
    }
    for(int i = 2*m-5; i >= 3; i-=2){
        sequence[p]=i; p++;
    }
    sequence[p]=4*m-3; p++;
    sequence[p]=2*m-3; p++;
    sequence[p]=4*m-6; p++;
    for(int i = 3; i <= 2*m-5; i+=2){
        sequence[p]=i; p++;
    }
    sequence[p]=4*m-4; p++;
    for(int i = 2*m-2; i <= 4*m-8; i+=2){
        sequence[p]=i; p++;
    }

    sequence[p]=4*m-2; p++;
    for(int i = 4*m-5; i >= 2*m-1; i-=2){
        sequence[p]=i; p++;
    }
    for(int i = 2*m-4; i >= 2; i-=2){
        sequence[p]=i; p++;
    }
    sequence[p]=4*m-6; p++;
    sequence[p]=4*m-3; p++;
    for(int i = 2; i <= 2*m-4; i+=2){
        sequence[p]=i; p++;
    }
    sequence[p]=4*m-4; p++;
    for(int i = 2*m-1; i <= 4*m-5; i+=2){
        sequence[p]=i; p++;
    }

    sequence[p]=0; p++;
    sequence[p]=4*m-2; p++;
}
if(r==6){

```

defect 2

```
//make hooked langford sequence of order 6,

sequence[0]=2;
sequence[1]=5;
sequence[2]=3;
sequence[3]=2;
sequence[4]=4;
sequence[5]=6;
sequence[6]=3;
sequence[7]=5;
sequence[8]=1;
sequence[9]=4;
sequence[10]=1;
sequence[11]=0;
sequence[12]=6;
}
else if (r==2) {
    sequence[0]=1;
    sequence[1]=2;
    sequence[2]=1;
    sequence[3]=0;
    sequence[4]=2;
}

break;

case 1:
    /* if r=4m-3
    4m - 6, ..., 2m - 2, 4m--5, 2m - 5, ..., 1, 4m - 4,
1,...
    2m - 5, 2m - 2, ..., 4m - 6, 4m - 3, 4m - 7, ..., 2m
- 1,
    4m--5, 2m-4, ..., 2, 2m--3, 4m-4, 2, ..., 2m-4,
    2m - 1, ..., 4m- 7, 2m - 3, *, 4m - 3. */

m=(r+3)/4;
p=0;
if (r>1) {
    for(int i = 4*m-6; i >= 2*m-2; i-=2){
        sequence[p]=i; p++;
    }
    sequence[p]=4*m-5; p++;
    for(int i = 2*m-5; i >= 1; i-=2){
        sequence[p]=i; p++;
    }
    sequence[p]=4*m-4; p++;
    for(int i = 1; i <= 2*m-5; i+=2){
        sequence[p]=i; p++;
    }
    for(int i = 2*m-2; i <= 4*m-6; i+=2){
        sequence[p]=i; p++;
    }
    sequence[p]=4*m-3; p++;
    for(int i = 4*m-7; i >= 2*m-1; i-=2){
        sequence[p]=i; p++;
    }
}
```

```

    }
    sequence[p]=4*m-5; p++;
    for(int i = 2*m-4; i >= 2; i-=2){
        sequence[p]=i; p++;
    }
    sequence[p]=2*m-3; p++;
    sequence[p]=4*m-4; p++;
    for(int i = 2; i <= 2*m-4; i+=2){
        sequence[p]=i; p++;
    }
    for(int i = 2*m-1; i <= 4*m-7; i+=2){
        sequence[p]=i; p++;
    }
    sequence[p]=2*m-3; p++;
    sequence[p]=0; p++;
    sequence[p]=4*m-3; p++;
}
else if(r==1){
    sequence[0]=1;
    sequence[1]=0;
    sequence[2]=1;
}

break;
}

//increase every number by one, to make it proper
for(int i=0; i<2*r+1; i++){
    if(sequence[i]!=0) sequence[i]+=1;
}

langford = ltriples(sequence, r);

return langford;
}

/**
 * Helper method that turns a (hooked) Langford sequence into
difference triples
 * @param sequence (hooked) Langford sequence
 * @param r order of the (hooked) Langford sequence
 * @return returns an r by 3 matrix of difference triples
 */
private int[][] ltriples(int[] sequence, int r){
    int[][] ltrip = new int[r][3];
    boolean notFound = true;

    for(int k=2; k<2+r; k++){
        ltrip[k-2][0]=k;
        notFound=true;
        for(int i=0; i<2*r+1 && notFound; i++){
            if(sequence[i]==k){
                ltrip[k-2][1]=i+1+2+r-1;
                ltrip[k-2][2]=i+1+k+2+r-1;
                ltrip[k-2][2]*=-1;
                notFound = false;
            }
        }
    }
}

```



```

        }
    }

    return ltrip;
}

/**
 * Helper method to return a matrix X with rows swap1 and swap2
switched
 * @param swap1    row to be switched with row swap2
 * @param swap2    row to be switched with row swap1
 * @param r number of rows in X
 * @param m number of columns in X
 * @param X an r by m matrix of integers
 * @return returns X with entries in rows swap1 and swap2
switched
 */
private int[][] swap(int swap1, int swap2, int r, int m, int[][]
X) {
    int[] temp = new int[m];
    for(int i=0; i<m; i++){
        temp[i] = X[swap1][i];
        X[swap1][i]=X[swap2][i];
        X[swap2][i]=temp[i];
    }

    return X;
}

/**
 * Method used to check to see if we can generate a vertex
labeling
 * using the differences given.
 * @param r number of rows
 * @param m number of columns
 * @param differences    an r by m matrix consisting of
differences
 * @param startValue    value at which we start labeling the
first cycle
 * @return returns true if we can generate a disjoint vertex
labeling, false otherwise
 */
private boolean isDisjoint(int r, int m, int[][] differences, int
startValue) {
    //initialize vector that represents used vertex labels
    boolean[] used = new boolean[r*m*2+1];
    //set used to equal false
    for(int i=0; i<r*m*2+1; i++){
        used[i]=false;
    }

    //vertex labels
    int[][] labels = new int[r][m];

    //set up first row of vertex labels

```

```

        labels[0][0]=startValue;
        used[labels[0][0]]=true;
        for(int i=1; i<m; i++){
            //check to see if number is already used, if not
            assign it!
            if(!used[mod(labels[0][i-1]+differences[0][i-1],r*m*2+1)]){
                labels[0][i]=mod(labels[0][i-1]+differences[0][i-1],(r*m*2+1));
                used[labels[0][i]]=true;
            }
        }

        //go row by row (let the first label be the lowest possible
number,
        // incrementing by 1 each time if it is already used)
        //check to make sure vertex labels aren't repeated
        //if they are, set vertex labels used this row to false
        //start the row over again, incrementing by 1.

        boolean isValid=false;
        boolean rowWorks;
        for(int i=1; i<r; i++){
            rowWorks=false;
            for(int n=0; n<r*m*2+1 && !rowWorks; n++){ //n
represents vertex labels we attempt to use
                //assign vertex labels
                if(!used[n]){
                    isValid=true;
                    labels[i][0]=n;           //assign first
value of cycle (determining the rest below)
                    used[labels[i][0]]=true;
                    for(int j=1; j<m && isValid; j++){
                        if(!used[mod(labels[i][j-1]+differences[i][j-1],(r*m*2+1))]){
                            labels[i][j]=mod(labels[i][j-1]+differences[i][j-1],(r*m*2+1));
                            used[labels[i][j]]=true;
                        }
                    }
                    //row works if we were able
                    if(j==m-1) rowWorks = true;
                }
            }
            else{           //if not,try an new
starting value for our cycle (but first... undo labels assigned)
                //undo "using" all labels
                for(j=j-1 ; j>=0; j--){
                    used[labels[i][j]]=false;
                }
                //trigger ending this loop
                isValid=false;
            }
        }
    }
}

```

```

        }
        if(!isValid && (n==r*m*2)){
            return false;
        }
    }
}

return true;
}

/**
 * Evaluates i (mod m)
 * @param i integer to be modulo'd
 * @param m modulo
 * @return returns i (mod m)
 */
private int mod(int i, int m){
    if(i%m>=0) return i%m;
    else return i%m+m;
}

/**
 * Rearranges X and returns it. Used so we can attempt to find
different
 * and more rho labelings
 * @param r number of rows in X
 * @param m number of columns in X
 * @param X an r by m matrix
 * @return returns X after it has been rearranged (rows shifted
and swapped)
 */
private int[][] rearrange(int r, int m, int[][] X){
    int[] temp = new int[m];
    for(int j=0; j<m; j++) temp[j]=X[0][j];

    for(int i=0; i<r-1; i++){
        for(int j=0; j<m; j++){
            X[i][j]=X[i+1][j];
        }
    }
    for(int j=0; j<m; j++) X[r-1][j]=temp[j];

    //try swapping rows
    this.swap(0, r-2, r, m, X);

    return X;
}

/**
 * Debug method to print out an r by m matrix X
 * @param r number of rows of X
 * @param m number of columns of X
 * @param X the r by m matrix to be printed
 */
private void debug(int r, int m, int[][] X){
    int sum = 0;
    for(int i=0; i<r; i++){

```

```

sum=0;
for(int j=0; j<m; j++){
    System.out.print(""+X[i][j]+" ");
    sum+=X[i][j];
}
System.out.println(": Sum=" + mod(sum,2*r*m+1));
}
}
}

```