CYCLE SYSTEMS: AN INVESTIGATION OF COLOURING AND INVARIANTS

ANDREA BURGESS

CYCLE SYSTEMS:
AN INVESTIGATION OF COLOURING AND INVARIANTS

by

© Andrea Burgess

A thesis submitted to the
School of Graduate Studies
in partial fulfilment of the
requirements for the degree of
Master of Science

Department of Mathematics and Statistics
Memorial University of Newfoundland

October 2005

St. John's Newfoundland

Abstract

An m-cycle system of order n is a partition of the edges of the complete graph K_n into m-cycles. This thesis explores two aspects of cycle systems: colouring of cycle systems and invariants for cycle systems.

A weak k-colouring of an m-cycle system of order n is a partition of the vertices of K_n into k colour classes such that the vertices of no m-cycle are all of the same colour. The smallest value of k for which a cycle system S admits a weak k-colouring is called the chromatic number of S. We study weak colourings of even cycle systems, and show that for any integers $k \geq 2$ and $r \geq 2$, there is a k-chromatic $(2r)$-cycle system.

An invariant for an m-cycle system is a function I such that $I(S) = I(S')$ for any m-cycle systems S and S' which are isomorphic. We examine the utility of various invariants in distinguishing nonisomorphic cycle systems and enumerate all pairwise nonisomorphic 11-cycle systems of order 11.

Acknowledgements

First, I would like to thank my supervisor, David Pike, without whose support and ideas I could not have completed this work. I would also like to thank my family for their continued encouragement. Thanks also to my fellow students and friends Jiajia Zhang and Yubo Zou. Finally, I would like to acknowledge financial support from NSERC.

Contents

List of Tables

List of Figures

List of Appendices

Chapter 1

Introduction

An *m-cycle system of order n* is a partition of the edges of the complete graph K_n into m-cycles. Where it is convenient to specify the vertex set and cycle set, we will denote by $(X, \mathcal{C})$ a cycle system with vertex set X and cycle set $\mathcal{C}$. Necessary and sufficient conditions for the existence of an m-cycle system of order $n > 1$ are that n is odd, m divides $\frac{n(n-1)}{2}$, and $n \geq m$. For a given m, any integer n for which there exists an m-cycle system of order n will be said to be *m-admissible* (or simply *admissible* if the value of m is unambiguous). In the case $m = 3$, 3-cycle systems are also called *Steiner triple systems*, and have been widely studied as a type of block design. In this thesis, we will investigate two aspects of cycle systems, namely colouring of cycle systems and invariants for cycle systems. For basic definitions of concepts in graph theory and combinatorial design theory, the reader is referred to [3, 51]; other definitions will appear in later chapters as needed.

1.1 History

Interest in cycle systems dates back to the mid-19th century, with the proof that a 3-cycle system of order n exists if and only if $n \equiv 1$ or 3 (mod 6) [33]. Since that time, one of the most significant problems regarding cycle systems was to prove that the obvious necessary conditions for the existence of an m-cycle system of order n are sufficient for $m > 3$. The final solution to the existence problem would not appear until approximately a century and a half after the publication of [33]. We will review some of the major results leading to this solution; for a complete survey, the reader is referred to [35], followed by [1, 46] for the final solution.

By the end of the 19th century, it had been proven by Walecki [50] that an n-cycle system of order n, i.e. a Hamilton decomposition of K_n, exists for any odd $n > 3$. Approximately 70 years later, it was shown that for m even and $n \equiv 1 \pmod{2m}$, there is an m-cycle system of order n [34, 44]. In the case that m is odd, it was proven by Jackson [31] that there is an m-cycle system of order n for any $n \equiv 1$ or $m \pmod{2m}$, implying that the obvious necessary conditions are sufficent when $m = p^\alpha$ for any prime p and positive integer α. If $m = 2p^\alpha$ for some prime p and positive integer α, the obvious necessary conditions for the existence of an m-cycle system of order n were shown to be sufficient by Alspach and Varma [2]. That the obvious necessary conditions are sufficient for all m was finally shown by Alspach and Gavlas [1] in the case that m is odd and by Šajna [46] in the case that m is even.

1.2 Outline of thesis

In this thesis, we will discuss two aspects of cycle systems. Chapter 2 deals with weak colourings of cycle systems. The main result proven in this chapter is that for

any integers $k \geq 2$ and $r \geq 2$, there is a k-chromatic $(2r)$-cycle system. We also discuss the spectra of k-chromatic 4-cycle systems, and show that for any $k \geq 2$, there is a k-chromatic 4-cycle system of any admissible order $n \geq n_4(k)$, where $n_4(k)$ is the smallest order for which there is a k-chromatic 4-cycle system. The results of Sections 2.2.1, 2.2.2 and 2.2.3, dealing with colouring 4-cycle systems, have been accepted for publication; see [8].

Chapter 3 discusses invariants for cycle systems and their uses in the enumeration of cycle systems of small order. Our study of invariants and enumeration arose from our study of colouring cycle systems, as we wished to generate cycle systems of small order to test for properties that could potentially be exploited in the construction of cycle systems with desired colouring properties. Nevertheless, this topic is of interest on its own. In Chapter 3, we begin by discussing some known invariants for cycle systems. A deterministic search algorithm to enumerate pairwise nonisomorphic cycle systems of small order is described, whereby cycle system invariants are employed in the rejection of isomorphic copies of cycle systems. Finally, we present the results of our application of this algorithm, including a complete enumeration of pairwise nonisomorphic 11-cycle systems of order 11, and we comment on the effectiveness of cycle system invariants in distinguishing nonisomorphic systems.

In Chapter 4, we summarize the results presented in the thesis and pose some open questions which arise from our work.

Chapter 2

Colouring of Cycle Systems

In this chapter, some results regarding weak colouring of cycle systems are presented. We begin by defining the relevant terms. We then review the history of the study of cycle system colouring, and follow with a discussion of weak colouring of even cycle systems.

Let k be a positive integer. A cycle system is said to be *weakly k-colourable* if its vertex set may be partitioned into k sets, called *colour classes*, such that no cycle is monochromatic, i.e. no cycle has all of its vertices the same colour; such a partition is referred to as a *weak k-colouring*. (Other types of cycle system colourings which have been studied include colourings in which every cycle has each of its vertices a different colour (called *strong colourings*), colourings in which each cycle must have at least two vertices of the same colour, and colourings in which each cycle must have at least two vertices coloured differently and at least two vertices of the same colour. A survey of various types of colourings for triple systems may be found in [11, Chapter 18]; for further information on these and other types of colourings, see also [10, 21, 27, 38, 43].) In this thesis, all of the colourings that we consider are

weak colourings. Therefore, we hereafter omit the descriptors "weak" and "weakly"; weak colourings will simply be called colourings, and a weakly k-colourable system will be said to be k-colourable. For a given cycle system, it is a natural question to ask what is the smallest value of k for which the system has a k-colouring. This value is called the system's *chromatic number*; a cycle system has chromatic number k, or is *k-chromatic*, if it is k-colourable but not $(k-1)$-colourable.

2.1 History of cycle system colouring

Most of the focus on colouring cycle systems has been in the colouring of 3-cycle systems, or Steiner triple systems. It is trivial that any STS(3) is 2-chromatic, but it was proven by Rosa [45] and independently by Pelikán [41] that any Steiner triple system of order $n \geq 7$ has chromatic number at least 3. Indeed, any Steiner triple system of order n, where $7 \leq n \leq 15$, is 3-chromatic [36]. Furthermore, for any admissible $n \geq 7$, there is a 3-chromatic STS(n) [45].

Although the results just mentioned all involve 3-chromatic Steiner triple systems, higher chromatic numbers are indeed attained. This observation was first made by Rosa [45], who noted that for any positive integer k, there is a Steiner triple system whose chromatic number is at least k; this result stems from the facts that for any k, there is a partial Steiner triple system with chromatic number at least k [22], and that any partial Steiner triple system can be embedded in a Steiner triple system [49]. In fact, de Brandes, Phelps and Rödl [17] proved in 1982 that any integer $k \geq 3$ is the chromatic number of some Steiner triple system, and moreover, for any integer $k \geq 3$, there is an integer v_k such that for any admissible $v \geq v_k$, there is a k-chromatic STS(v). Let $n_3(k)$ denote the smallest such v_k which is an admissible order of a

Steiner triple system. It is known that $n_3(3) = 7$ [45], while $19 \leq n_3(4) \leq 21$ [28], $27 \leq n_3(5) \leq 127$ [23, 24, 25, 30], and $27 \leq n_3(6) \leq 487$ [5, 23, 24, 30]. The question of whether, for a given order v, the set $C(v)$ of integers k which are the chromatic number of some STS(v) is an interval was posed by de Brandes, Phelps and Rödl [17]; this question has been answered in the affirmative for $v \leq 25$ [23, 24, 30, 36], but is unsolved in general. Another natural question is whether $n_3(k)$ is the smallest order for which there exists a k-chromatic 3-cycle system. For $k = 3$ and $k = 4$, this question has also been answered in the affirmative. (See [45] and [28].) However, for $k \geq 5$, the question is unsettled [17, 25].

Weak colourings of m-cycle systems for $m > 3$ have been less widely studied than triple system colourings. It is evident from the definition that any m-cycle system of order $n > 1$ must have chromatic number at least 2. Milici and Tuza [40] showed that every m-cycle system of order $2m + 1$ is 2-colourable for any $m > 3$, while each m-cycle system of order $4m + 1$ is 2-colourable if $m \geq 10$, thus proving the existence of 2-chromatic m-cycle systems for any $m > 3$. It was also proven in [40] that no m-cycle system of order $n \leq \sqrt{m2^m} + \frac{1}{2}$ or $n < \frac{2^m}{em}$ has chromatic number greater than 2. Nevertheless, in a previous paper, Milici and Tuza [39] showed that for any $m > 3$, there exists an m-cycle system which is not 2-colourable; they did not, however, determine the chromatic number of the non-2-colourable cycle systems that they constructed.

2.2 Colouring 4-cycle systems

In this section, we give some results about colouring of 4-cycle systems. The main result of this section is an analogue for 4-cycle systems of a theorem of

de Brandes, Phelps and Rödl [17] regarding 3-cycle systems. In particular, we prove, by constructive methods, that for any integer $k \geq 2$, there is an admissible integer $n_4(k)$ such that for every admissible $n \geq n_4(k)$, there is a k-chromatic 4-cycle system of order n. As many of our constructions require the following theorem of Sotteau, we state it here.

Theorem 2.2.1. [47] *Necessary and sufficient conditions for the decomposition of the complete bipartite graph $K_{m,n}$ into $(2r)$-cycles are that m and n are even, $m \geq r$, $n \geq r$, and mn is divisible by $2r$.*

Our preliminary investigation into the colouring of 4-cycle systems led us to consider systems with special properties. A cycle system is called *uniquely k-colourable* if every k-colouring represents the same partition of the vertex set into colour classes, that is, the system has only one k-colouring up to permutation of colour classes. We begin by briefly discussing some implications of the existence of a uniquely k-colourable cycle system.

Lemma 2.2.2. *Let $(X, \mathcal{C})$ be a k-chromatic m-cycle system of order n. If this system is uniquely k-colourable, then in a k-colouring of $(X, \mathcal{C})$, each colour class must have size at least $m - 1$.*

Proof. Give $(X, \mathcal{C})$ a k-colouring with colour classes $C_1, \ldots, C_k$, where for each $i \in \{1, \ldots, k\}$, the elements of C_i have colour i. Suppose there is some colour class C_i which has size at most $m - 2$. Let C_j be another colour class, and let $x \in C_j$. Re-colour x with colour i. Then no cycle can be monochromatically coloured with colour i since at most $m - 1$ vertices have this colour. Clearly, no cycle can be monochromatically coloured with colour j or with any other colour, since the original colouring had no monochromatic cycle. So we have created a k-colouring of $(X, \mathcal{C})$

which yields a different partition into colour classes than the original colouring, thus contradicting that $(X, \mathcal{C})$ is uniquely k-colourable. $\square$

While the next lemma does not deal directly with unique colourings, we will subsequently employ it in a construction requiring a uniquely colourable 4-cycle system as an ingredient.

Lemma 2.2.3. *Let $(X, \mathcal{C})$ be a k-chromatic 4-cycle system of order n, and consider a fixed k-colouring of $(X, \mathcal{C})$. Let $X = \{\infty\} \cup \{x_1, \ldots, x_{n-1}\}$ where ∞ is in a colour class of smallest size. Then $\{x_1, x_2, \ldots, x_{n-1}\}$ can be partitioned into pairs, such that at least one pair is contained in a largest colour class, C_k, and no pair not contained C_k is monochromatic.*

Proof. Let the colour classes, with the element ∞ removed, be C_1, C_2, $\ldots$, C_k, where $|C_1| < |C_2|$ and $|C_2| \leq |C_3| \leq \cdots \leq |C_k|$.

Note that we may assume, without loss of generality, that each C_i is of even cardinality. Otherwise, noting that there must be an even number of sets C_i of odd cardinality, choose an element from each such set, and partition these elements into pairs. None of these pairs is monochromatic, and we are left with k sets of even cardinality to partition into pairs in the required way. As well, if $|C_k|$ is odd, then this pairing removes an element y of C_k, and an element of each other class of size $|C_k|$ (if such classes exist), so $C_k - \{y\}$ is a set of largest size among the remaining sets.

For each element $v \in C_1$, choose a distinct element $u \in C_2$ and form the pair $\{u, v\}$. Then, if $k > 2$ and any elements $v \in C_2$ remain, for each such element, choose a distinct element $u \in C_3$ and form the pair $\{u, v\}$. We proceed inductively. For $i = 3, \ldots, k-1$, having previously paired any remaining elements of C_{i-1} with

elements of C_i, for any unpaired elements $u \in C_i$, choose a distinct element $v \in C_{i+1}$ and form the pair $\{u, v\}$. Such pairing is always possible, since for each $i \in \{1, \ldots, k-1\}$, $|C_i| \leq |C_{i+1}|$.

Now, if at least two elements of C_k remain, we can partition the remaining elements of C_k into pairs, and we are then finished. Otherwise, no elements of C_k remain; we note in this case that $|C_{k-1}| = |C_k|$ and $k \geq 3$ (since $|C_1| < |C_2|$). Choose two elements $u_1, u_2 \in C_{k-1}$ which have been placed in pairs $\{u_1, v_1\}$, $\{u_2, v_2\}$, where v_1, $v_2 \in C_k$. Choose a pair $\{w, z\}$ which has already been formed, where $w \in C_i$ for some $i \leq k-2$. Note that $z \notin C_{k-1}$, since all elements of C_{k-1} have been used in forming the pairs among elements of C_{k-1} and elements of C_k. Replace the pairs $\{u_1, v_1\}$, $\{u_2, v_2\}$ and $\{w, z\}$ with $\{u_1, w\}$, $\{u_2, z\}$ and $\{v_1, v_2\}$. We have now partitioned the elements $\{x_1, \ldots, x_{n-1}\}$ into pairs as required, in which exactly one pair, namely $\{v_1, v_2\}$, is contained in the colour class C_k and no other pair is monochromatic. □

The following theorem constructs a $(k+1)$-chromatic 4-cycle system of order $2n-1$, given a uniquely k-colourable, k-chromatic 4-cycle system of order n.

Theorem 2.2.4. *If there exists a k-chromatic, uniquely k-colourable 4-cycle system of order n, then there exists a $(k+1)$-chromatic 4-cycle system of order $2n-1$.*

Proof. Let $(X_1, \mathcal{C}_1)$ and $(X_2, \mathcal{C}_2)$ be uniquely k-colourable 4-cycle systems of order n (possibly $X_1 = X_2$ and $\mathcal{C}_1 = \mathcal{C}_2$), where $X_1 = \{\infty, x_{1,1}, x_{2,1}, \ldots, x_{n-1,1}\}$, $X_2 = \{\infty, x_{1,2}, x_{2,2}, \ldots, x_{n-1,2}\}$, and ∞ is in a colour class of smallest size in each of $(X_1, \mathcal{C}_1)$ and $(X_2, \mathcal{C}_2)$. We will form a decomposition of the $K_{n-1,n-1}$ between $\{x_{1,1}, \ldots, x_{n-1,1}\}$ and $\{x_{1,2}, \ldots, x_{n-1,2}\}$ into 4-cycles, which, combined with the cycles of $\mathcal{C}_1$ and $\mathcal{C}_2$, will form a 4-cycle system of order $2n-1$ on vertex set $X_1 \cup X_2$.

By Lemma 2.2.3, $\{x_{1,1}, \ldots, x_{n-1,1}\}$ can be partitioned into pairs such that at least

one pair is contained in a colour class of $(X_1, \mathcal{C}_1)$ of largest size, and no pair not contained in this colour class is monochromatic. Let $\{v_1, v_2\}$ be a pair contained in this largest colour class. For each colour class C_i ($i \in \{1, \ldots, k\}$) of $(X_2, \mathcal{C}_2)$, choose two elements $u_{i,1}$, $u_{i,2} \in C_i - \{\infty\}$ and form the 4-cycle $(v_1, u_{i,1}, v_2, u_{i,2})$. (The vertices $u_{i,1}, u_{i,2} \in C_i - \{\infty\}$ exist by Lemma 2.2.2.) Form the remaining cycles between $\{v_1, v_2\}$ and $\left(\left(X_2 - \cup_{i=1}^{k}\{u_{i,1}, u_{i,2}\}\right) - \{\infty\}\right)$ using any decomposition of $K_{2,n-1-2k}$ into 4-cycles. (Such a decomposition exists by Theorem 2.2.1.)

For each of the remaining pairs $\{w_1, w_2\} \neq \{v_1, v_2\}$ in the pairing of $X_1 - \{\infty\}$, form all cycles containing w_1 and w_2 by using any decomposition of $K_{2,n-1}$ into 4-cycles. (Such a decomposition exists by Theorem 2.2.1.) This step completes the decomposition of the $K_{n-1,n-1}$ between $\{x_{1,1}, \ldots, x_{n-1,1}\}$ and $\{x_{1,2}, \ldots, x_{n-1,2}\}$ into 4-cycles, and so we have constructed a 4-cycle system S of order $2n-1$.

Suppose S is k-colourable, and assign a k-colouring. Then $(X_1, \mathcal{C}_1)$ and $(X_2, \mathcal{C}_2)$ have both been given k-colourings. Since $(X_1, \mathcal{C}_1)$ and $(X_2, \mathcal{C}_2)$ are uniquely k-colourable, then for some $j \in \{1, \ldots, k\}$, v_1 and v_2 both have colour j. Also, for each $i \in \{1, \ldots, k\}$, there exists $j_i \in \{1, \ldots, k\}$ such that $u_{i,1}$ and $u_{i,2}$ must have the same colour j_i, and if $i_1 \neq i_2$, then $j_{i_1} \neq j_{i_2}$. Choose $i \in \{1, \ldots, k\}$ such that $j_i = j$. The cycle $(v_1, u_{j_i,1}, v_2, u_{j_i,2})$ is monochromatic, which is a contradiction.

However, S can be $(k+1)$-coloured, as follows. In the pairing of the elements of $X_1 - \{\infty\}$ formed by Lemma 2.2.3, let the monochromatic pairs (including $\{v_1, v_2\}$) be $\{w_{1,1}, w_{1,2}\}$, $\{w_{2,1}, w_{2,2}\}$, $\ldots$, $\{w_{q,1}, w_{q,2}\}$. Give each of $(X_1, \mathcal{C}_1)$ and $(X_2, \mathcal{C}_2)$ a k-colouring, and then re-colour each of $w_{1,2}$, $w_{2,2}$, $\ldots$, $w_{q,2}$ with a $(k+1)^{\text{st}}$ colour. Clearly no 4-cycle in S which does not contain any of the vertices in $\{w_{1,2}, w_{2,2}, \ldots, w_{q,2}\}$ is monochromatic. Also, no 4-cycle in $\mathcal{C}_1$ or $\mathcal{C}_2$ is monochromatic. Any other 4-cycle of S has the form $(w_{i,1}, y_1, w_{i,2}, y_2)$ where $i \in \{1, \ldots, q\}$ and $y_1, y_2 \in X_2 - \{\infty\}$, and is

not monochromatic since $w_{i,1}$ and $w_{i,2}$ have different colours.

Thus no 4-cycle in S is monochromatic, and so we have produced a $(k+1)$-chromatic 4-cycle system. □

Theorem 2.2.4 provides a nice construction of a $(k+1)$-chromatic 4-cycle system, in essence of double the order of the k-chromatic, uniquely k-colourable system used to construct it. Nevertheless, we are unaware of whether a uniquely k-colourable, k-chromatic 4-cycle system actually exists for any $k \geq 2$. Thus, to prove that k-chromatic 4-cycle systems do indeed exist, we turn our attention to a different approach, which is presented in subsequent sections.

2.2.1 3-chromatic 4-cycle systems

In this section, we construct a 3-chromatic 4-cycle system S of order 49.

We begin our construction by taking six sets $S_i = \{x_{1,i}, x_{2,i}, \ldots, x_{8,i}\}$, $i \in \{1, 2, \ldots, 6\}$, and adding a vertex ∞. For each $i \in \{1, 2, \ldots, 6\}$, we create the following 4-cycles: $(x_{1,i}, x_{2,i}, x_{3,i}, x_{4,i})$, $(\infty, x_{1,i}, x_{3,i}, x_{5,i})$, $(x_{1,i}, x_{5,i}, x_{2,i}, x_{6,i})$, $(x_{1,i}, x_{7,i}, x_{2,i}, x_{8,i})$, $(\infty, x_{2,i}, x_{4,i}, x_{6,i})$, $(\infty, x_{3,i}, x_{6,i}, x_{7,i})$, $(\infty, x_{4,i}, x_{5,i}, x_{8,i})$, $(x_{5,i}, x_{6,i}, x_{8,i}, x_{7,i})$, $(x_{3,i}, x_{7,i}, x_{4,i}, x_{8,i})$. Note that for each $i \in \{1, \ldots, 6\}$, we have formed a 4-cycle system of order 9 with vertex set $S_i \cup \{\infty\}$, with ∞ being a vertex shared by each system.

To complete the 4-cycle system S, we will define a 4-cycle decomposition of the copy of $K_{8,8}$ between S_i and S_j for each pair $\{i, j\} \subset \{1, 2, \ldots, 6\}$. We will use the following two decompositions, D and D'. The decomposition D, which is illustrated in Figure 2.1, has the following 4-cycles:

$(x_{1,i}, x_{1,j}, x_{3,i}, x_{3,j})$, $(x_{2,i}, x_{2,j}, x_{4,i}, x_{4,j})$, $(x_{5,i}, x_{5,j}, x_{7,i}, x_{7,j})$, $(x_{6,i}, x_{6,j}, x_{8,i}, x_{8,j})$,

$(x_{1,i}, x_{2,j}, x_{3,i}, x_{4,j})$, $(x_{2,i}, x_{1,j}, x_{4,i}, x_{3,j})$, $(x_{5,i}, x_{6,j}, x_{7,i}, x_{8,j})$, $(x_{6,i}, x_{5,j}, x_{8,i}, x_{7,j})$,

$(x_{1,i}, x_{5,j}, x_{2,i}, x_{6,j})$, $(x_{3,i}, x_{7,j}, x_{4,i}, x_{8,j})$, $(x_{5,i}, x_{1,j}, x_{6,i}, x_{2,j})$, $(x_{7,i}, x_{3,j}, x_{8,i}, x_{4,j})$,

$(x_{1,i}, x_{7,j}, x_{2,i}, x_{8,j})$, $(x_{3,i}, x_{5,j}, x_{4,i}, x_{6,j})$, $(x_{5,i}, x_{3,j}, x_{6,i}, x_{4,j})$, $(x_{7,i}, x_{1,j}, x_{8,i}, x_{2,j})$.

The decomposition D', which is illustrated in Figure 2.2, has the following 4-cycles:

$(x_{1,i}, x_{1,j}, x_{4,i}, x_{4,j})$, $(x_{2,i}, x_{2,j}, x_{3,i}, x_{3,j})$, $(x_{5,i}, x_{5,j}, x_{8,i}, x_{8,j})$, $(x_{6,i}, x_{6,j}, x_{7,i}, x_{7,j})$,

$(x_{1,i}, x_{2,j}, x_{4,i}, x_{3,j})$, $(x_{2,i}, x_{1,j}, x_{3,i}, x_{4,j})$, $(x_{5,i}, x_{6,j}, x_{8,i}, x_{7,j})$, $(x_{6,i}, x_{5,j}, x_{7,i}, x_{8,j})$,

$(x_{1,i}, x_{5,j}, x_{4,i}, x_{8,j})$, $(x_{2,i}, x_{6,j}, x_{3,i}, x_{7,j})$, $(x_{5,i}, x_{1,j}, x_{8,i}, x_{4,j})$, $(x_{6,i}, x_{2,j}, x_{7,i}, x_{3,j})$,

$(x_{1,i}, x_{6,j}, x_{4,i}, x_{7,j})$, $(x_{2,i}, x_{5,j}, x_{3,i}, x_{8,j})$, $(x_{5,i}, x_{2,j}, x_{8,i}, x_{3,j})$, $(x_{6,i}, x_{1,j}, x_{7,i}, x_{4,j})$.

Let $i, j \in \{1, 2, \ldots, 6\}$, $i \neq j$. If $\{i, j\} \subseteq \{1, 2, 3\}$ or $\{i, j\} \subseteq \{4, 5, 6\}$, we put decomposition D between S_i and S_j. Otherwise, we put decomposition D' between S_i and S_j. We have now constructed a 4-cycle system S of order 49.

Lemma 2.2.5. *The 4-cycle system S which we have constructed is not 2-colourable.*

Proof. Assume that S is 2-colourable, and assign an arbitrary 2-colouring. For each $i \in \{1, 2, \ldots, 6\}$, let $X_i = (x_{1,i}, x_{2,i}, x_{3,i}, x_{4,i})$ and let $Y_i = (x_{5,i}, x_{6,i}, x_{7,i}, x_{8,i})$. Let $A_1 = \{X_1, X_2, X_3\}$, $A_2 = \{Y_1, Y_2, Y_3\}$, $B_1 = \{X_4, X_5, X_6\}$ and $B_2 = \{Y_4, Y_5, Y_6\}$.

Note that each of A_1, A_2, B_1 and B_2 can have at most one 4-tuple coloured in either pattern `1*1*` or `*1*1` (where `*` is a placeholder for an arbitrary colour); otherwise, there would be a monochromatic cycle in D. Also, each can have at most one 4-tuple coloured in either pattern `2*2*` or `*2*2`. Since each of A_1, A_2, B_1 and B_2 contains three 4-tuples, it follows that each must contain a 4-tuple coloured with none of these patterns. Say $X_{i_1} \in A_1$, $Y_{i_2} \in A_2$, $X_{j_1} \in B_1$ and $Y_{j_2} \in B_2$ are each coloured with patterns different from those given above. The possible patterns for these four 4-tuples are as follows: `1122`, `2211`, `1221` and `2112`.

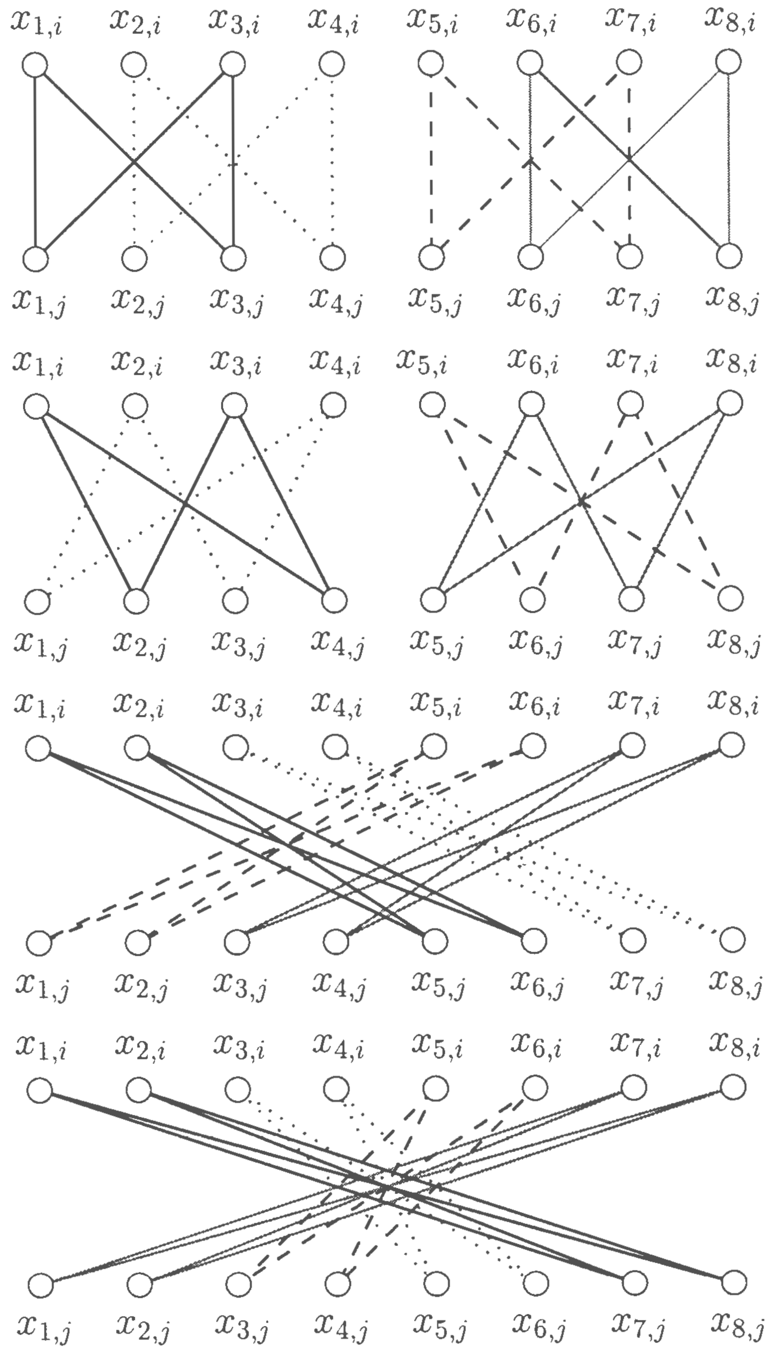

Figure 2.1: The decomposition D.

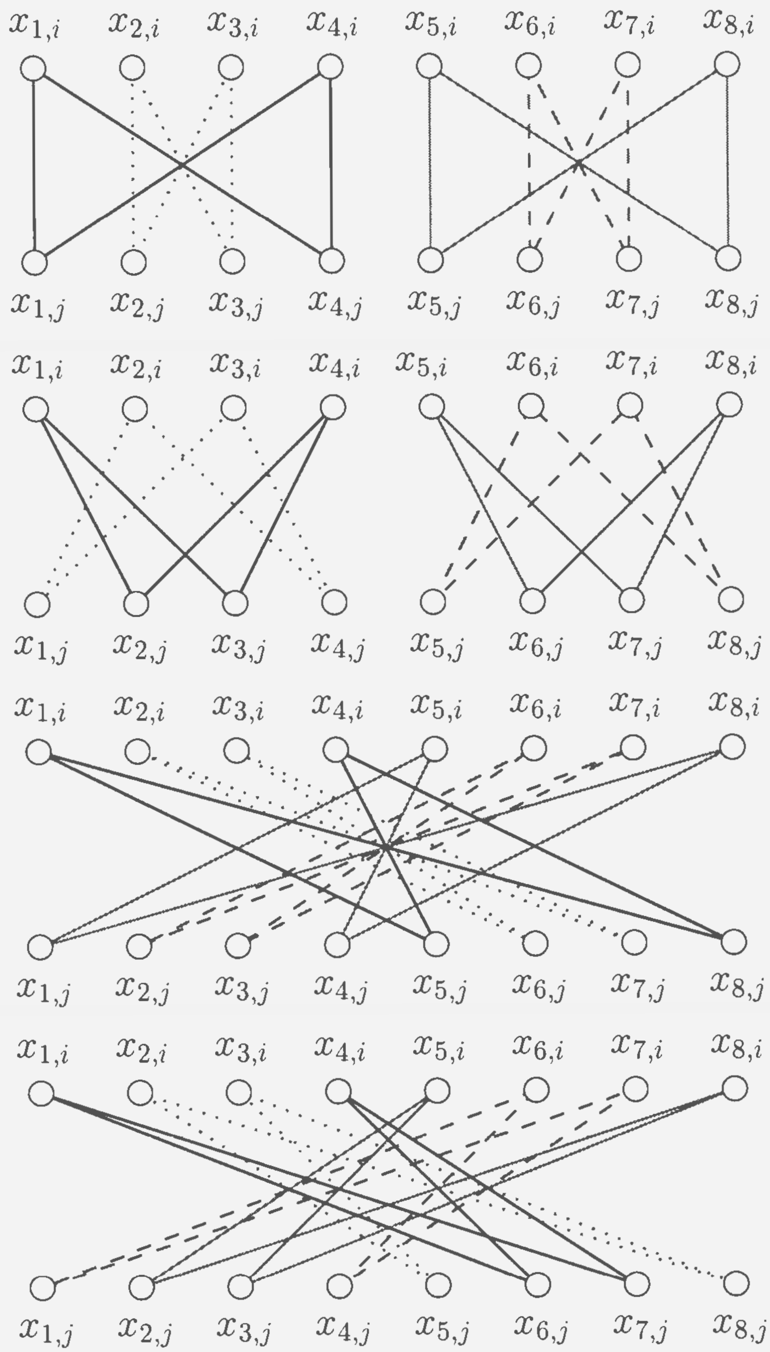

Figure 2.2: The decomposition D'.

Consider the effects of colouring X_{i_1} with either 1122 or 2211. If $i_1 \neq i_2$, then S_{i_1} and S_{i_2} have decomposition D between them, which means that having Y_{i_2} coloured with either 1122 or 2211 would guarantee a monochromatic cycle. If $i_1 = i_2$, then S contains the 4-cycles $(x_{1,i_1}, x_{5,i_1}, x_{2,i_1}, x_{6,i_1})$ and $(x_{1,i_1}, x_{7,i_1}, x_{2,i_1}, x_{8,i_1})$, which again would guarantee a monchromatic cycle were Y_{i_2} coloured by either 1122 or 2211. It follows that if X_{i_1} is not coloured in either of patterns 1221 or 2112, then Y_{i_2} must be coloured in one of these patterns.

Hence, at least one of X_{i_1} and Y_{i_2} is coloured in one of the patterns 1221 and 2112. Let $Z_1 \in \{X_{i_1}, Y_{i_2}\}$ be one such 4-tuple. Similarly, at least one of X_{j_1} and Y_{j_2} is coloured in one of the patterns 1221 and 2112. Let $Z_2 \in \{X_{j_1}, Y_{j_2}\}$ be one such 4-tuple. Now, the set S_i which contains Z_1 and the set S_j which contains Z_2 have decomposition D' between them, which implies that there is a monochromatic cycle between Z_1 and Z_2. □

Lemma 2.2.6. *The 4-cycle system S which we have constructed is 3-colourable.*

Proof. We exhibit a specific 3-colouring. Give ∞ colour 1. For each $i \in \{1, 2, \ldots, 6\}$, we colour the vertices in the set S_i as follows.

Vertex	$x_{1,i}$	$x_{2,i}$	$x_{3,i}$	$x_{4,i}$	$x_{5,i}$	$x_{6,i}$	$x_{7,i}$	$x_{8,i}$
Colour	1	1	2	3	1	2	3	3

It is easy to verify that no 4-cycle in any of the 4-cycle systems that were created with vertex set $S_i \cup \{\infty\}$ has a monochromatic cycle. The remaining 4-cycles of S occur in the decompositions D and D' placed between sets of the form S_i and S_j. It is also readily verified that neither decomposition between any two sets S_i and S_j yields a monochromatic cycle. □

2.2.2 k-chromatic 4-cycle systems

In the preceding section, we proved the existence of a 3-chromatic 4-cycle system. We now extend this result with the following theorem.

Theorem 2.2.7. *For any integer $k \geq 2$, there exists a 4-cycle system which is k-chromatic.*

For $k = 2$, it is easily verified that any of the eight pairwise nonisomorphic 4-cycle systems of order 9 is 2-chromatic. (For a list of these systems, refer to [19].) The results of Section 2.2.1 guarantee the existence of a 3-chromatic 4-cycle system. We now consider the case where $k \geq 4$.

Given $k \geq 4$, we wish to construct a 4-cycle system which is not $(k-1)$-colourable, but is k-colourable. Let ℓ be the least even integer greater than or equal to k, and let h be the least multiple of 8 greater than or equal to ℓ. To construct the required system, we will take t sets of vertices $S_i = \{x_{1,i}, x_{2,i}, \ldots, x_{h,i}\}$, $i \in \{1, 2, \ldots, t\}$, where the value of t will be determined later. Each set S_i will be joined to a common vertex ∞ to form a 4-cycle system of order $h + 1$. By placing 4-cycle decompositions of $K_{h,h}$ between any two distinct sets S_i and S_j, we will create a 4-cycle system of order $ht + 1$.

For each $i \in \{1, 2, \ldots, t\}$, let $X_i = \{x_{1,i}, x_{2,i}, \ldots, x_{\ell,i}\}$. We will now define 4-cycle decompositions of $K_{\ell,\ell}$ to be placed between distinct sets X_i and X_j. We begin by forming a decomposition D_1 of $K_{\ell,\ell}$ consisting of the 4-cycles $(x_{2u-1,i}, x_{2v-1,j}, x_{2u,i}, x_{2v,j})$, $1 \leq u, v \leq \ell/2$. Observe that if $(x_{\beta,i}, x_{\gamma,j}, x_{\delta,i}, x_{\varepsilon,j})$ is a cycle in D_1, then so is $(x_{\beta,j}, x_{\gamma,i}, x_{\delta,j}, x_{\varepsilon,i})$. Hence, in placing decomposition D_1 between X_i and X_j, the order of i and j is irrelevant.

We will say that a set X_i is coloured with a pattern $c_1 c_2 \ldots c_\ell$ if, for each

$\beta \in \{1, 2, \ldots, \ell\}$, $x_{\beta,i}$ has colour c_β. Given a colour c, no two sets X_i and X_j with D_1 between them can both have one of the following colour patterns: cc**** $\cdots$ **, **cc**** $\cdots$ **, ****cc** $\cdots$ **, $\ldots$, **** $\cdots$ **cc. If $X_{i_1}, X_{i_2}, \ldots, X_{i_n}$ pairwise have D_1 between them, then at most one of them can have one of the $\ell/2$ colouring patterns listed above. Let $P_{c,1}$ denote this set of $\ell/2$ colouring patterns for the colour c. In a $(k-1)$-colouring, there are $k-1$ such sets: $P_{1,1}, P_{2,1}, \ldots, P_{k-1,1}$.

We now form additional 4-cycle decompositions of $K_{\ell,\ell}$ as follows. Let σ be a permutation of $\{1, 2, \ldots, \ell\}$. Consider the decomposition

$$\{(x_{\sigma(2u-1),i}, x_{\sigma(2v-1),j}, x_{\sigma(2u),i}, x_{\sigma(2v),j}) : 1 \leq u, v \leq \ell/2\}.$$

If this decomposition does not contain the 4-cycle $(x_{1,i}, x_{1,j}, x_{2,i}, x_{2,j})$, then we will call σ an *acceptable* permutation. The number of permutations of $\{1, 2, \ldots, \ell\}$ which are not acceptable is $2\left(\frac{\ell}{2}\right)(\ell-2)!$, and so there are $\ell! - 2\left(\frac{\ell}{2}\right)(\ell-2)! = \ell(\ell-2)(\ell-2)!$ acceptable permutations. Letting $p = \ell(\ell-2)(\ell-2)! + 1$, denote the acceptable permutations by $\sigma_2, \sigma_3, \ldots, \sigma_p$, and let the corresponding decompositions be $D_2, D_3, \ldots, D_p$, respectively. Note that, for any $\alpha \in \{2, \ldots, p\}$, D_α has the property that, if $(x_{\beta,i}, x_{\gamma,j}, x_{\delta,i}, x_{\varepsilon,j})$ is a 4-cycle in D_α, then so is $(x_{\beta,j}, x_{\gamma,i}, x_{\delta,j}, x_{\varepsilon,i})$, since D_1 also has this property.

For each $\alpha \in \{2, \ldots, p\}$, and each colour c, let $P_{c,\alpha}$ be the set of colour patterns obtained by applying the permutation σ_α to the colour positions of each pattern in $P_{c,1}$. So $P_{c,\alpha}$ consists of $\ell/2$ patterns, each of which has two positions that require colour c and $\ell - 2$ positions that are unrestricted. Note that whenever X_i and X_j have the decomposition D_α between them, at most one of X_i and X_j can be coloured with a pattern from $P_{c,\alpha}$. Also, in a $(k-1)$-colouring, there are $k-1$ such sets $P_{1,\alpha}, P_{2,\alpha}, \ldots, P_{k-1,\alpha}$.

Lemma 2.2.8. *For any given $i \in \{1, 2, \ldots, t\}$, and any $(k-1)$-colouring of X_i, this colouring matches a pattern in $P_{c,\alpha}$ for some $c \in \{1, \ldots, k-1\}$, $\alpha \in \{1, \ldots, p\}$.*

Proof. Since X_i has $\ell > k-1$ elements, in any $(k-1)$-colouring of X_i, two elements must have the same colour. Say $x_{v_1,i}$ and $x_{v_2,i}$ both have colour c, where $v_1 < v_2$. If $\{v_1, v_2\} = \{2r-1, 2r\}$ for some $r \in \{1, \ldots, \ell/2\}$, then the colouring matches a pattern in $P_{c,1}$. Otherwise, if $v_1 \neq 2$, consider the permutation σ_{α_1} of $\{1, \ldots, \ell\}$, where $\sigma_{\alpha_1}(1) = v_1$, $\sigma_{\alpha_1}(2) = v_2$, $\sigma_{\alpha_1}(v_1) = 1$, $\sigma_{\alpha_1}(v_2) = 2$ and $\sigma_{\alpha_1}(y) = y$ if $y \in \{1, \ldots, \ell\} \setminus \{1, 2, v_1, v_2\}$. Then the colouring is in P_{c,α_1}. Next, if $v_1 = 2$ and v_2 is odd, consider the permutation σ_{α_2} defined by $\sigma_{\alpha_2}(2) = v_2 + 1$, $\sigma_{\alpha_2}(v_2 + 1) = 2$ and $\sigma_{\alpha_2}(y) = y$ if $y \in \{1, \ldots, \ell\} \setminus \{2, v_2+1\}$; the colouring is in P_{c,α_2}. Finally, if $v_1 = 2$ and v_2 is even, consider the permutation σ_{α_3} defined by $\sigma_{\alpha_3}(2) = v_2 - 1$, $\sigma_{\alpha_3}(v_2 - 1) = 2$ and $\sigma_{\alpha_3}(y) = y$ if $y \in \{1, \ldots, \ell\} \setminus \{2, v_2 - 1\}$. Then the colouring is in P_{c,α_3}. □

Now, let $t = k^p$. We will take t sets $S_i = \{x_{1,i}, \ldots, x_{h,i}\}$, $1 \leq i \leq t$, and use the p decompositions D_1, D_2, ..., D_p to form a 4-cycle decomposition of the copy of $K_{h,h}$ between each pair of sets S_i and S_j. We proceed in p steps to form the required 4-cycle decompositions.

Step 1. Take k sets S_1, S_2, ..., S_k of vertices, each of size h, and place D_1 between X_i and X_j, $1 \leq i, j \leq k$. Using Sotteau's result (Theorem 2.2.1), form the remaining 4-cycles in the 4-cycle decomposition of the $K_{h,h}$ between S_i and S_j using any 4-cycle decompositions of $K_{h-\ell,\ell}$ between $S_i - X_i$ and X_j and between $S_j - X_j$ and X_i, and any 4-cycle decomposition of $K_{h-\ell,h-\ell}$ between $S_i - X_i$ and $S_j - X_j$. Let R_1 denote the resulting configuration.

Step n ($2 \leq n \leq p$). Having previously completed Step $(n-1)$ to obtain R_{n-1}, we complete Step n as follows. Take k copies of R_{n-1}, so that we now have k^n

sets of vertices, with sets $S_{(i-1)k^{n-1}+1}$, $S_{(i-1)k^{n-1}+2}$, ..., $S_{(i-1)k^{n-1}+k^{n-1}} = S_{ik^{n-1}}$ being in the i^{th} copy of R_{n-1} for $i = 1, 2, \ldots, k$. The 4-cycle decomposition of $K_{h,h}$ between $S_{(i-1)k^{n-1}+d_1}$ and $S_{(i-1)k^{n-1}+d_2}$, where $i \in \{1, \ldots, k\}$, $d_1, d_2 \in \{1, \ldots, k^{n-1}\}$ and $d_1 \neq d_2$, is inherited from R_{n-1} and therefore makes use of one of the decompositions $D_1, \ldots, D_{n-1}$. Place D_n between any pair of sets $X_{(i-1)k^{n-1}+d_1}$ and $X_{(j-1)k^{n-1}+d_2}$, where $i, j \in \{1, \ldots, k\}$ such that $i \neq j$, and $d_1, d_2 \in \{1, \ldots, k^{n-1}\}$. For any pair $S_{(i-1)k^{n-1}+d_1}$ and $S_{(j-1)k^{n-1}+d_2}$, where $i, j \in \{1, \ldots, k\}$ such that $i \neq j$, and $d_1, d_2 \in \{1, \ldots, k^{n-1}\}$, complete the 4-cycle decomposition of $K_{h,h}$ between $S_{(i-1)k^{n-1}+d_1}$ and $S_{(j-1)k^{n-1}+d_2}$ using any 4-cycle decomposition of $K_{h-\ell,\ell}$ between $S_{(i-1)k^{n-1}+d_1} - X_{(i-1)k^{n-1}+d_1}$ and $X_{(j-1)k^{n-1}+d_2}$ and between $S_{(j-1)k^{n-1}+d_2} - X_{(j-1)k^{n-1}+d_2}$ and $X_{(i-1)k^{n-1}+d_1}$, and also by using any 4-cycle decomposition of $K_{h-\ell,h-\ell}$ between $S_{(i-1)k^{n-1}+d_1} - X_{(i-1)k^{n-1}+d_1}$ and $S_{(j-1)k^{n-1}+d_2} - X_{(j-1)k^{n-1}+d_2}$. Let R_n denote the configuration obtained as the result of Step n.

Lastly, we add a vertex ∞, and for each $i \in \{1, \ldots, t\}$, form a 4-cycle decomposition of the copy of K_{h+1} with vertex set $S_i \cup \{\infty\}$. We have now constructed a 4-cycle system S of order $hk^p + 1$.

Lemma 2.2.9. *The 4-cycle system S which we have constructed is not $(k-1)$-colourable.*

Proof. Assume that S is indeed $(k-1)$-colourable, and assign an arbitrary $(k-1)$-colouring.

For each $i, j \in \{1, \ldots, k\}$ such that $i \neq j$, and for each $d_1, d_2 \in \{1, \ldots, k^{p-1}\}$, $X_{(i-1)k^{p-1}+d_1}$ and $X_{(j-1)k^{p-1}+d_2}$ have decomposition D_p between them. Thus, for any given $c \in \{1, \ldots, k-1\}$, there can be at most one $i \in \{1, \ldots, k\}$ such that $X_{(i-1)k^{p-1}+d}$ is coloured in a pattern contained in $P_{c,p}$ for some $d \in \{1, \ldots, k^{p-1}\}$. It follows that

there exists an $i_p \in \{1, \ldots, k\}$ such that for any $d \in \{1, \ldots, k^{p-1}\}$, the colouring of $X_{(i_p-1)k^{p-1}+d}$ matches no pattern in $P_{c,p}$ for any $c \in \{1, 2, \ldots, k-1\}$. Without loss of generality, it may be assumed that $i_p = 1$.

Continue in an inductive fashion. Suppose that, for some $n \in \{3, 4, \ldots, p\}$, for any $d \in \{1, 2, \ldots, k^{n-1}\}$, the colouring of X_d matches no pattern in $P_{c,q}$ for any $c \in \{1, \ldots, k-1\}$, $q \in \{n, n+1, \ldots, p\}$. Note that for each $i, j \in \{1, \ldots, k\}$ such that $i \neq j$, and for each $d_1, d_2 \in \{1, \ldots, k^{n-2}\}$, $X_{(i-1)k^{n-2}+d_1}$ and $X_{(j-1)k^{n-2}+d_2}$ have decomposition D_{n-1} between them. Thus, for any given $c \in \{1, \ldots, k-1\}$, there can be at most one $i \in \{1, \ldots, k\}$ such that $X_{(i-1)k^{n-2}+d}$ is coloured in a pattern in $P_{c,n-1}$ for some $d \in \{1, \ldots, k^{n-2}\}$. It follows that there exists an $i_{n-1} \in \{1, \ldots, k\}$ such that for each $d \in \{1, \ldots, k^{n-2}\}$, the colouring of $X_{(i_{n-1}-1)k^{n-2}+d}$ matches no pattern in $P_{c,n-1}$ for any $c \in \{1, \ldots, k-1\}$. Without loss of generality, it may be assumed that $i_{n-1} = 1$. Thus, for $d \in \{1, 2, \ldots, k^{n-2}\}$, the colouring of X_d matches no pattern in $P_{c,q}$ for any $c \in \{1, \ldots, k-1\}$, $q \in \{n-1, n, \ldots, p\}$.

We continue in this manner, until we have determined that for any $d \in \{1, \ldots, k\}$, the colouring of X_d matches no pattern in $P_{c,q}$ for any $c \in \{1, \ldots, k-1\}$, $q \in \{2, 3, \ldots, p\}$. Now, for any $i, j \in \{1, \ldots, k\}$ such that $i \neq j$, X_i and X_j have decomposition D_1 between them. Given $c \in \{1, \ldots, k-1\}$, there can be at most one $i \in \{1, 2, \ldots, k\}$ such that X_i is coloured in a pattern in $P_{c,1}$. So there must be some $i_1 \in \{1, \ldots, k\}$ such that the colouring of X_{i_1} matches no pattern in $P_{c,1}$ for any $c \in \{1, \ldots, k-1\}$. Hence the $(k-1)$-colouring of X_{i_1} is contained in none of the pattern sets $P_{c,q}$ where $c \in \{1, \ldots, k-1\}$, $q \in \{1, \ldots, p\}$, which contradicts Lemma 2.2.8. □

Lemma 2.2.10. *The 4-cycle system S which we have constructed is k-colourable.*

Proof. We consider the following cases.

Case 1. k is even.

Colour the vertex ∞ with colour 1. Given any constituent set S_i, colour the $\ell = k$ vertices of its X_i with a distinct colour so that each of the k colours is used exactly once, and colour the vertices of $S_i - X_i$ with colours 1, 2, ..., $h - \ell$, so that no two elements of $S_i - X_i$ have the same colour. Note that for any even $k \geq 4$, $h - \ell = h - k \leq k$, so this colouring can be done using k colours.

Then no 4-cycle contained within one of the 4-cycle systems with vertex set $S_i \cup \{\infty\}$ has a monochromatic cycle for any $i \in \{1, \ldots, t\}$, since no four vertices of $S_i \cup \{\infty\}$ all have the same colour. Also, no cycle contained in a copy of D_α between X_i and X_j $(1 \leq i, j \leq t)$ is monochromatic, since no two elements of X_i have the same colour. Finally, no other 4-cycle in S is monochromatic, since any such cycle must contain two elements of $S_i - X_i$ for some $i \in \{1, \ldots, t\}$; these two elements do not have the same colour.

So we have given S a k-colouring in which no 4-cycle of S is monochromatic.

Case 2. k is odd.

We will revisit the construction of S to k-colour the sets S_i. At the end of Step 1, we have k sets S_i, any two of which have decomposition D_1 between them. We k-colour the corresponding sets X_i such that all k colours are used in X_i, $x_{1,i}$ and $x_{2,i}$ have the same colour, and if $i \neq j$, then $x_{1,i}$ and $x_{1,j}$ have different colours.

At this point, there are no monochromatic cycles involving the cycles in decomposition D_1, since given X_i and X_j, no four vertices in $X_i \cup X_j$ have the same colour.

Colour the $h - \ell$ vertices in each set $S_i - X_i$ with colours 2, 3, ..., $h - \ell + 1$ so that no two have the same colour. Note that for any odd $k \geq 5$, $h - \ell + 1 = h - k \leq k$, so this colouring is possible using k colours.

In the remainder of the construction, when we take a copy of a configuration, we give the copy the same colouring as the original.

All that remains to be coloured is the vertex ∞. Give ∞ colour 1. Note that no cycle in a constituent 4-cycle system with vertex set $S_i \cup \{\infty\}$ for some i is monochromatic, since no four vertices of $S_i \cup \{\infty\}$ have the same colour.

Now, consider the cycles between sets X_i and X_j (i.e. the cycles of the decompositions D_α, $\alpha \in \{1, \ldots, p\}$). Recall that, after Step 1, no such cycle was monochromatic. In the completed cycle system S, any two sets S_i and S_j with D_1 between them occur in a single copy of the results of Step 1, which has the same colouring as the original, and hence no cycle in a copy of D_1 is monochromatic. If two sets X_i and X_j have decomposition D_α between them for some $\alpha \neq 1$, then there can be no monochromatic cycle between X_i and X_j, as follows. Recalling that σ_α is an acceptable permutation, D_α cannot contain the 4-cycle $(x_{1,i}, x_{1,j}, x_{2,i}, x_{2,j})$. Choose any 4-cycle in the copy of D_α between X_i and X_j. If $x_{1,i}$ and $x_{2,i}$ are both vertices in this cycle, then $x_{1,j}$ and $x_{2,j}$ cannot both be vertices in this cycle. So the two vertices in this cycle contained in X_j do not have the same colour. Otherwise, $x_{1,i}$ and $x_{2,i}$ are not both in the cycle, so the two vertices of the cycle contained in X_i do not have the same colour. In either case, the cycle is not monochromatic.

Finally, any remaining cycle in S contains two vertices in $S_i - X_i$ for some $i \in \{1, \ldots, t\}$; these two vertices do not have the same colour.

Hence, we have given S a k-colouring in which no 4-cycle of S is monochromatic.

□

2.2.3 Spectra of k-chromatic 4-cycle systems

In 1982, de Brandes, Phelps and Rödl [17] showed that for any $k \geq 3$, there is an integer v_k such that for any $v \geq v_k$ such that $v \equiv 1$ or $3 \pmod 6$, there is a k-chromatic STS(v). Let $n_3(k)$ denote the smallest such v_k which is an admissible order of a Steiner triple system. Recall from Section 2.1 that $n_3(3) = 7$ [45], $19 \leq n_3(4) \leq 21$ [28], $27 \leq n_3(5) \leq 127$ [23, 24, 25, 30] and $27 \leq n_3(6) \leq 487$ [5, 23, 24, 30].

We now consider a similar concept for 4-cycle systems. Let $n_4(k)$ denote the least value w_k for which there exists a k-chromatic 4-cycle system of order w_k. That there exists a k-chromatic 4-cycle system of order n for every admissible $n \geq n_4(k)$ easily follows from Theorem 2.2.7 and repeated application of the following lemma:

Lemma 2.2.11. *If there exists a k-chromatic 4-cycle system of order n, then there exists a k-chromatic 4-cycle system of order $n + 8$.*

Proof. We present an iterative construction similar to one found in [35], accompanied by information about colouring. Let S denote a fixed k-chromatic 4-cycle system of order n, and let S have vertex set $\{y_1, y_2, \ldots, y_n\}$. We add eight vertices $x_1, x_2, \ldots, x_8$. To create a 4-cycle system of order $n + 8$, we will decompose the copy of K_9 with vertex set $\{y_1, x_1, x_2, \ldots, x_8\}$ and the copy of $K_{8,n-1}$ with bipartition (A, B), where $A = \{x_1, x_2, \ldots, x_8\}$, $B = \{y_2, y_3, \ldots, y_n\}$, into 4-cycles.

First, we form a 4-cycle decomposition of K_9 with vertex set $\{y_1, x_1, x_2, \ldots, x_8\}$ by taking the following nine 4-cycles: (y_1, x_1, x_2, x_3), (y_1, x_2, x_5, x_4), (y_1, x_5, x_1, x_6), (y_1, x_7, x_1, x_8), (x_1, x_3, x_6, x_4), (x_2, x_4, x_7, x_6), (x_2, x_7, x_3, x_8), (x_3, x_4, x_8, x_5), (x_5, x_6, x_8, x_7). These nine 4-cycles form a 4-cycle system of order 9, which we will denote by T.

Next, we form the required 4-cycle decomposition of $K_{8,n-1}$ as follows. For each $i \in \{1,2,3,4\}$, using Theorem 2.2.1, create a 4-cycle decomposition of $K_{2,n-1}$ between $\{x_{2i-1}, x_{2i}\}$ and $\{y_1, y_2, \ldots, y_n\}$. Combined, these decompositions form a 4-cycle decomposition of $K_{8,n-1}$.

We have now formed a 4-cycle system S' of order $n+8$. Note that S' contains the original system S. Hence, since S is k-chromatic, clearly the chromatic number of S' is at least k. To prove that S' is indeed k-chromatic, we will exhibit a specific k-colouring of S'.

First, colour the vertices y_1, y_2, $\ldots$, y_n, using k colours, such that y_1 has colour 1 and no 4-cycle in S is monochromatic. Next, for $i \in \{1,2,\ldots,8\}$, colour x_i with colour 1 if i is even, and with colour 2 otherwise. Note that none of the nine 4-cycles of T is monochromatic, since each contains a pair of vertices x_i and x_j where i and j have different parity. Any remaining 4-cycle of S' must contain two vertices x_{2i-1} and x_{2i} for some $i \in \{1,2,3,4\}$; these two vertices are coloured differently.

So we have given S' a k-colouring in which no 4-cycle of S' is monochromatic, and hence S' is k-chromatic. □

As previously mentioned, each 4-cycle system of order 9 is 2-chromatic, so that $n_4(2) = 9$. Furthermore, based on the k-chromatic cycle systems for $k \geq 3$ that we have constructed, we have the following bounds: $17 \leq n_4(3) \leq 49$, and for $k \geq 4$, $17 \leq n_4(k) \leq hk^p + 1$, where h and p are as defined in the proof of Theorem 2.2.7. We note that the value of p that we have provided could likely be lowered, thereby reducing the order of the system constructed, as some of the decompositions $D_2, \ldots, D_p$ may have the same set of cycles. The problem of finding the exact value of $n_4(k)$ for $k \geq 3$ remains open.

2.3 Colouring even cycle systems

We now consider colourings of even cycle systems in general. In this section, the results of Section 2.2.2 will be extended, to prove that for any $r \geq 3$ and $k \geq 2$, there exists a k-chromatic $(2r)$-cycle system. We will begin, however, by considering $(2r)$-cycle decompositions of the complete bipartite graph $K_{2r,2r}$, which will be used in later constructions of $(2r)$-cycle systems.

2.3.1 Decomposition of $K_{2r,2r}$ into $(2r)$-cycles

That $K_{2r,2r}$ admits a $(2r)$-cycle decomposition for any integer $r \geq 2$ can be readily seen from Theorem 2.2.1. In Section 2.3.2, however, we will require a $(2r)$-cycle decomposition which satisfies an additional property, which we will discuss in Lemma 2.3.4. We begin by stating some results regarding Hamilton decompositions, or decompositions into Hamiltonian cycles, of the complete bipartite graph $K_{r,r}$ and related graphs.

Theorem 2.3.1. [50] *If r is a positive even integer, then $K_{r,r}$ has a Hamilton decomposition.*

Theorem 2.3.2. [20] *Let r be a positive odd integer. For any 1-factor I of $K_{r,r}$, $K_{r,r} - I$ has a Hamilton decomposition.*

In [7], it is proved in two ways that if r is odd, then for any 2-factor U of $K_{r,r}$, there is a 1-factor F of $K_{r,r} - E(U)$ such that $(K_{r,r} - E(U)) - F$ has a Hamilton decomposition. The following theorem follows naturally from the second proof.

Theorem 2.3.3. [7] *Let r be a positive odd integer and let $G \cong K_{r,r}$ be the complete bipartite graph with bipartition $(\{w_1, w_2, \ldots, w_r\}, \{y_1, y_2, \ldots, y_r\})$. For each*

edge $\{w_i, y_j\}$, *consider the* difference value $(j - i) \pmod{r}$, *and let* H *denote the set of edges of difference* 0, 1 *or* $r - 1$. *Then* $G - H$ *has a Hamilton decomposition.*

We are now able to find the desired $(2r)$-cycle decomposition of $K_{2r,2r}$ for any integer $r \geq 2$.

Lemma 2.3.4. *Let* $r \geq 2$ *be an integer, and let* $G \cong K_{2r,2r}$ *be the complete bipartite graph with bipartition* (W, Y), *where* $W = \{w_1, w_2, \ldots, w_{2r}\}$ *and* $Y = \{y_1, y_2, \ldots, y_{2r}\}$. *Then* G *has a decomposition into* $(2r)$*-cycles with the property that for any cycle* C *in this decomposition, if* $V(C) \cap W = \{w_{i_1}, w_{i_2}, \ldots, w_{i_r}\}$ *where* $i_1 < i_2 < \cdots < i_r$, $i_1 \leq r$ *and* $i_r > r$, *then* $V(C) \cap Y \neq \{y_{i_1}, y_{i_2}, \ldots, y_{i_r}\}$. *Furthermore, this decomposition contains at least one cycle on vertex set* $\{w_1, \ldots, w_r, y_1, \ldots, y_r\}$ *and at least one cycle on vertex set* $\{w_{r+1}, \ldots, w_{2r}, y_{r+1}, \ldots, y_{2r}\}$.

Proof. We begin by noting that the edge set of G consists exactly of the edges of the following four copies of $K_{r,r}$: the $K_{r,r}$ between $\{w_1, \ldots, w_r\}$ and $\{y_1, \ldots, y_r\}$ (which we will denote by G_1), the $K_{r,r}$ between $\{w_{r+1}, \ldots, w_{2r}\}$ and $\{y_{r+1}, \ldots, y_{2r}\}$ (which we will denote by G_2), the $K_{r,r}$ between $\{w_{r+1}, \ldots, w_{2r}\}$ and $\{y_1, \ldots, y_r\}$ (which we will denote by G_3), and the $K_{r,r}$ between $\{w_1, \ldots, w_r\}$ and $\{y_{r+1}, \ldots, y_{2r}\}$ (which we will denote by G_4).

If r is even, then by Theorem 2.3.1, each of G_1, G_2, G_3 and G_4 has a $(2r)$-cycle decomposition. Combined, these four $(2r)$-cycle decompositions form a $(2r)$-cycle decomposition of the $K_{2r,2r}$ with bipartition (W, Y), which is easily seen to have the desired properties.

We now suppose r is odd. Let I denote an arbitrary 1-factor of $K_{r,r}$. By Theorem 2.3.2, $G_1 - I$, $G_2 - I$ and $G_3 - I$ each have $(2r)$-cycle decompositions, and by Lemma 2.3.3, $G_4 - H$ has a decomposition into $(2r)$-cycles, where H

consists of the edges $\{w_i, y_j\} \in E(G_4)$ such that $j - i \equiv 0$, 1, or $r-1 \pmod{r}$. Thus, it need only be proved that the graph G' induced by edges $I_1 \cup I_2 \cup I_3 \cup H$ can be decomposed into $(2r)$-cycles, where I_1, I_2 and I_3 are 1-factors (to be determined later) of G_1, G_2 and G_3, respectively, such that for any cycle C in the decomposition of G', if $W \cap V(C) = \{w_{i_1}, w_{i_2}, \ldots, w_{i_r}\}$ where $i_1 < i_2 < \cdots < i_r$, $i_1 \leq r$ and $i_r > r$, then $Y \cap V(C) \neq \{y_{i_1}, y_{i_2}, \ldots, y_{i_r}\}$.

We will consider separately the four cases $r \equiv 1$, 3, 5 or 7 (mod 8); in each case, the three cycles which make up the decomposition of G' will be defined. As an explanation of the notation used, '$w_i : y_j$' (similarly, '$y_j : w_i$') indicates that between w_i and y_j we will use a path consisting of three edges, one each from I_1, I_2 and I_3. For example, $(y_4, w_2 : y_5, w_3)$ denotes a 6-cycle containing the edges $\{y_4, w_2\}$, $\{y_5, w_3\}$, $\{w_3, y_4\}$ and three edges (one from each of I_1, I_2 and I_3) which form a 3-path between w_2 and y_5; such a 6-cycle is illustrated in Figure 2.3. Since the three edges indicated by '$w_i : y_j$' or '$y_j : w_i$' constitute the unique 3-path between w_i and y_j in $G[I_1 \cup I_2 \cup I_3]$, they do not need to be explicitly stated. As $K_{r,r} - I$ has a Hamilton decomposition for any 1-factor I, it will follow that the required 1-factors I_1, I_2 and I_3 exist provided each vertex w_i $(1 \leq i \leq r)$ and y_j $(r+1 \leq j \leq 2r)$ is incident with such an edge exactly once.

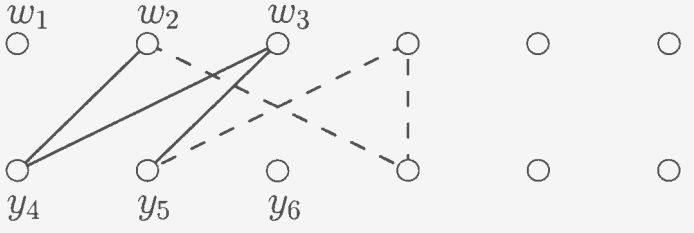

Figure 2.3: A 6-cycle represented by $(y_4, w_2 : y_5, w_3)$.

Case 1. $r \equiv 1 \pmod 8$

In the case $r = 9$, the three 18-cycles $(w_1, y_{10} : w_2, y_{11} : w_3, y_{12} : w_4, y_{13} : w_9, y_{18})$, $(w_1, y_{11}, w_3, y_{13}, w_5, y_{14} : w_6, y_{15} : w_7, y_{16}, w_8, y_{17} :)$ and $(y_{10}, w_2, y_{12}, w_4, y_{14}, w_6, y_{16} : w_8, y_{18} : w_5, y_{15}, w_7, y_{17}, w_9)$ form the required decomposition, so we will hereafter assume $r \geq 17$. Write $r = 8s + 1$, and consider the following three $(2r)$-cycles, where the use of underbracing in a cycle is intended to clarify the patterns which form the cycle.

$$
\begin{aligned}
C_1 &= (\underbrace{w_1, y_{8s+2} : w_2, y_{8s+3} : \ldots : w_{4s}, y_{12s+1}}_{} : w_{8s+1}, y_{16s+2}) \\
C_2 &= (\underbrace{w_1, y_{8s+3}, w_3, y_{8s+5}, \ldots, w_{4s-1}, y_{12s+1}}_{}, \\
&\qquad \underbrace{w_{4s+1}, y_{12s+2} : w_{4s+2}, y_{12s+3} : \ldots : w_{6s}, y_{14s+1}}_{} : \\
&\qquad \underbrace{w_{6s+1}, y_{14s+2}, w_{6s+2}, y_{14s+3}, \ldots, w_{8s}, y_{16s+1}}_{} :) \\
C_3 &= (\underbrace{y_{8s+2}, w_2, y_{8s+4}, w_4, \ldots, y_{14s}, w_{6s}}_{}, y_{14s+2} : \\
&\qquad \underbrace{w_{4s+1}, y_{12s+3}, w_{4s+3}, y_{12s+5}, \ldots, w_{6s+1}, y_{14s+3}}_{} : \\
&\qquad \underbrace{w_{6s+2}, y_{14s+4} : w_{6s+3}, y_{14s+5} : \ldots : w_{8s-2}, y_{16s}}_{} : \\
&\qquad w_{8s}, y_{16s+2} : w_{8s-1}, y_{16s+1}, w_{8s+1})
\end{aligned}
$$

Note that C_1 has $4(4s) + 2 = 8s + 2 = 2r$ edges, C_2 has $(4s) + 4(2s) + 2(2s) + 2 = 8s + 2 = 2r$ edges, and C_3 has $(6s + 1) + 2 + (2s + 2) + 2 + 4(2s - 3) + 7 = 16s + 2 = 2r$ edges, as required.

To see that this decomposition of H has the desired property, note that the $(2r)$-cycle C_1 contains $4s + 1$ vertices of $\{w_1, w_2, \ldots, w_r\}$ and $4s + 1$ vertices of $\{y_{r+1}, y_{r+2}, \ldots, y_{2r}\}$, hence $4s$ vertices of $\{y_1, y_2, \ldots, y_r\}$. The $(2r)$-cycle C_2 contains $6s$ vertices of $\{w_1, w_2, \ldots, w_r\}$ and $2s + 1$ of $\{y_1, \ldots, y_r\}$. The $(2r)$-cycle C_3 contains $6s + 1$ vertices of $\{w_1, w_2, \ldots, w_r\}$ and $2s$ of $\{y_1, \ldots, y_r\}$. Thus, choosing $C \in \{C_1, C_2, C_3\}$, and letting $V(C) \cap \{w_1, \ldots, w_{2r}\} = \{w_{i_1}, w_{i_2}, \ldots, w_{i_r}\}$

where $i_1 < i_2 < \cdots < i_r$, it follows that $V(C) \cap \{y_1, \ldots, y_{2r}\} \neq \{y_{i_1}, y_{i_2}, \ldots, y_{i_r}\}$, since $|V(C) \cap \{w_1, \ldots, w_r\}| \neq |V(C) \cap \{y_1, \ldots, y_r\}|$.

Case 2. $r \equiv 5 \pmod 8$

In the case $r = 5$, the three 10-cycles $(w_1, y_6 : w_2, y_7 : w_5, y_{10})$, $(w_1, y_7, w_3, y_8 : w_4, y_9 :)$, $(y_6, w_2, y_8, w_4, y_{10} : w_3, y_9, w_5)$ form the required decomposition. Now suppose that $r \geq 13$, and write $r = 8s + 5$. Consider the following three $(2r)$-cycles.

$$
\begin{aligned}
C_1 &= (\underbrace{w_1, y_{8s+6} : w_2, y_{8s+7} : \ldots : w_{4s+2}, y_{12s+7}}\, : w_{8s+5}, y_{16s+10}) \\
C_2 &= (\underbrace{w_1, y_{8s+7}, w_3, y_{8s+9}, \ldots, w_{4s+1}, y_{12s+7}}, \\
&\qquad \underbrace{w_{4s+3}, y_{12s+8} : w_{4s+4}, y_{12s+9} : \ldots : w_{6s+3}, y_{14s+8}}\, : \\
&\qquad \underbrace{w_{6s+4}, y_{14s+9}, w_{6s+5}, y_{14s+10}, \ldots, w_{8s+4}, y_{16s+9}}\, :) \\
C_3 &= (\underbrace{y_{8s+6}, w_2, y_{8s+8}, w_4, \ldots, y_{14s+8}, w_{6s+4}}, y_{14s+10} : \\
&\qquad \underbrace{w_{4s+3}, y_{12s+9}, w_{4s+5}, y_{12s+11}, \ldots, w_{6s+3}, y_{14s+9}}\, : \\
&\qquad \underbrace{w_{6s+5}, y_{14s+11} : w_{6s+6}, y_{14s+12} : \ldots : w_{8s+2}, y_{16s+8}}\, : \\
&\qquad w_{8s+4}, y_{16s+10} : w_{8s+3}, y_{16s+9}, w_{8s+5})
\end{aligned}
$$

Note that C_1 has $4(4s+2)+2 = 16s+10 = 2r$ edges, C_2 has $(4s+2)+4(2s+1)+2(2s+1)+2 = 16s+10 = 2r$ edges, and C_3 has $(6s+5)+2+(2s+2)+2+4(2s-2)+7 = 16s + 10 = 2r$ edges, as required.

To see that this decomposition of H has the desired property, note that C_1 contains $4s + 3$ vertices of $\{w_1, w_2, \ldots, w_r\}$ but $4s + 2$ vertices of $\{y_1, y_2, \ldots, y_r\}$, C_2 contains $6s + 3$ vertices of $\{w_1, w_2, \ldots, w_r\}$ but $2s + 2$ vertices of $\{y_1, y_2, \ldots, y_r\}$ and C_3 contains $6s+4$ vertices of $\{w_1, w_2, \ldots, w_r\}$ but $2s+1$ vertices of $\{y_1, y_2, \ldots, y_r\}$. Thus, for any $C \in \{C_1, C_2, C_3\}$, $|V(C) \cap \{w_1, \ldots, w_r\}| \neq |V(C) \cap \{y_1, \ldots, y_r\}|$, and so

if $V(C)\cap\{w_1,\ldots,w_{2r}\}=\{w_{i_1},\ldots,w_{i_r}\}$, it cannot be true that $V(C)\cap\{y_1,\ldots,y_{2r}\}=\{y_{i_1},\ldots,y_{i_r}\}$.

Case 3. $r\equiv 3 \pmod 8$

If $r=3$, then the three 6-cycles $(w_1,y_6,w_3:y_4)$, $(w_1:y_6,w_2,y_5)$ and $(y_4,w_2:y_5,w_3)$ give the required decomposition. We now suppose $r\geq 11$, and write $r=8s+3$. Consider the following three $(2r)$-cycles.

$$\begin{aligned}
C_1 &= (\underbrace{w_1,y_{8s+4}:w_2,y_{8s+5}:\ldots:w_{4s+1},y_{12s+4}:}w_{8s+3},y_{16s+6})\\
C_2 &= (\underbrace{y_{8s+4},w_2,y_{8s+6},w_4,\ldots,y_{12s+2},w_{4s}},y_{12s+4},\\
&\qquad \underbrace{w_{4s+2},y_{12s+5}:w_{4s+3},y_{12s+6}:\ldots:w_{6s+2},y_{14s+5}:}\\
&\qquad \underbrace{w_{6s+3},y_{14s+6},w_{6s+4},y_{14s+7},\ldots,w_{8s+2},y_{16s+5}},w_{8s+3})\\
C_3 &= (\underbrace{w_1,y_{8s+5},w_3,y_{8s+7},\ldots,w_{6s+3},y_{14s+7}}:\\
&\qquad \underbrace{w_{4s+2},y_{12s+6},w_{4s+4},y_{12s+8},\ldots,w_{6s+2},y_{14s+6}}:\\
&\qquad \underbrace{w_{6s+4},y_{14s+8}:w_{6s+5},y_{14s+9}:\ldots:w_{8s+2},y_{16s+6}:})
\end{aligned}$$

Note that C_1 has $4(4s+1)+2=16s+6=2r$ edges, C_2 has $(4s+1)+4(2s+1)+2(2s)+1=16s+6=2r$ edges and C_3 has $(6s+4)+2+(2s+2)+2+4(2s-1)=16s+6=2r$ edges, as required.

To see that this decomposition of H has the desired property, note that C_1 contains $4s+2$ vertices of $\{w_1,w_2,\ldots,w_r\}$ but $4s+1$ vertices of $\{y_1,y_2,\ldots,y_r\}$, C_2 contains $6s+2$ vertices of $\{w_1,w_2,\ldots,w_r\}$ but $2s+1$ vertices of $\{y_1,y_2,\ldots,y_r\}$ and C_3 contains $6s+2$ vertices of $\{w_1,w_2,\ldots,w_r\}$ but $2s+1$ vertices of $\{y_1,y_2,\ldots,y_r\}$. Thus, for any $C\in\{C_1,C_2,C_3\}$, $|V(C)\cap\{w_1,\ldots,w_r\}|\neq|V(C)\cap\{y_1,\ldots,y_r\}|$, and so if $V(C)\cap\{w_1,\ldots,w_{2r}\}=\{w_{i_1},\ldots,w_{i_r}\}$, it cannot be true that $V(C)\cap\{y_1,\ldots,y_{2r}\}=\{y_{i_1},\ldots,y_{i_r}\}$.

Case 4. $r \equiv 7 \pmod 8$

If $r = 7$, then the three 14-cycles $(w_1, y_8 : w_2, y_9 : w_3, y_{10} : w_7, y_{14})$, $(y_8, w_2, y_{10}, w_4, y_{11} : w_5, y_{12} : w_6, y_{13}, w_7)$ and $(w_1, y_9, w_3, y_{11}, w_5, y_{13} : w_4, y_{12}, w_6, y_{14} :)$ form the required decomposition. We now suppose $r \geq 15$, and write $r = 8s + 7$. Consider the following three $(2r)$-cycles.

$$\begin{aligned}
C_1 &= (\underbrace{w_1, y_{8s+8} : w_2, y_{8s+9} : \ldots : w_{4s+3}, y_{12s+10}}_{} : w_{8s+7}, y_{16s+14}) \\
C_2 &= (\underbrace{y_{8s+8}, w_2, y_{8s+10}, w_4, \ldots, y_{12s+8}, w_{4s+2}}_{}, y_{12s+10}, \\
&\qquad \underbrace{w_{4s+4}, y_{12s+11} : w_{4s+5}, y_{12s+12} : \ldots : w_{6s+5}, y_{14s+12}}_{} : \\
&\qquad \underbrace{w_{6s+6}, y_{14s+13}, w_{6s+7}, y_{14s+14}, \ldots, w_{8s+6}, y_{16s+13}}_{}, w_{8s+7}) \\
C_3 &= (\underbrace{w_1, y_{8s+9}, w_3, y_{8s+11}, \ldots, w_{6s+5}, y_{14s+13}}_{} : \\
&\qquad \underbrace{w_{4s+4}, y_{12s+12}, w_{4s+6}, y_{12s+14}, \ldots, w_{6s+6}, y_{14s+14}}_{} : \\
&\qquad \underbrace{w_{6s+7}, y_{14s+15} : w_{6s+8}, y_{14s+16} : \ldots : w_{8s+6}, y_{16s+14}}_{} :)
\end{aligned}$$

Note that C_1 has $4(4s+3)+2 = 16s+14 = 2r$ edges, C_2 has $(4s+3)+4(2s+2)+2(2s+1)+1 = 16s+14 = 2r$ edges and C_3 has $(6s+6)+2+(2s+4)+2+4(2s) = 16s+14 = 2r$ edges, as required.

To see that this decomposition of H has the desired property, note that C_1 contains $4s+4$ vertices of $\{w_1, w_2, \ldots, w_r\}$ but $4s+3$ vertices of $\{y_1, y_2, \ldots, y_r\}$, C_2 contains $6s+5$ vertices of $\{w_1, w_2, \ldots, w_r\}$ but $2s+2$ vertices of $\{y_1, y_2, \ldots, y_r\}$ and C_3 contains $6s+5$ vertices of $\{w_1, w_2, \ldots, w_r\}$ but $2s+2$ vertices of $\{y_1, y_2, \ldots, y_r\}$. Thus, for any $C \in \{C_1, C_2, C_3\}$, $|V(C) \cap \{w_1, \ldots, w_r\}| \neq |V(C) \cap \{y_1, \ldots, y_r\}|$, and so if $V(C) \cap \{w_1, \ldots, w_{2r}\} = \{w_{i_1}, \ldots, w_{i_r}\}$, it cannot be true that $V(C) \cap \{y_1, \ldots, y_{2r}\} = \{y_{i_1}, \ldots, y_{i_r}\}$. $\square$

2.3.2 Colouring $(2r)$-cycle systems

In this section, we will prove the following generalization of Theorem 2.2.7 to even cycle systems.

Theorem 2.3.5. *For any integers $k \geq 2$ and $r \geq 2$, there exists a k-chromatic $(2r)$-cycle system.*

In the case $k = 2$, Milici and Tuza have shown that any m-cycle system of order $2m + 1$ is 2-colourable for any integer $m \geq 4$ [40]. Furthermore, in the case $r = 2$, it is proven in Section 2.2 that for any $k \geq 2$, there is a k-chromatic 4-cycle system. We therefore need only consider the case $r \geq 3$ and $k \geq 3$.

Let $k \geq 3$ and $r \geq 3$ be integers. We wish to construct a $(2r)$-cycle system which is k-chromatic. Let ℓ be the least multiple of $2r$ greater than $(r-1)(k-1)$. Write $\ell = (r-1)k + s$. Note that $2 - r \leq s \leq r + 1$. Let $\varsigma = 2\left\lceil \frac{s}{2} \right\rceil$ be the least even integer greater than or equal to s, and let h denote the least multiple of $4r$ greater than or equal to ℓ (so $h - \ell \in \{0, 2r\}$).

To construct a k-chromatic $(2r)$-cycle system, we will take t sets of vertices $S_i = \{x_{1,i}, \ldots, x_{h,i}\}$, $i \in \{1, \ldots, t\}$, where the value of t will be determined later. Each set S_i will be joined to a common vertex ∞ to form a $(2r)$-cycle system of order $h + 1$. By also placing $(2r)$-cycle decompositions of $K_{h,h}$ between any two distinct sets S_i and S_j, we will create a $(2r)$-cycle system S of order $ht + 1$.

For each $q \in \{0, 1, \ldots, \frac{\ell}{2r} - 1\}$, observe that between the vertices in $W_q = \{x_{2rq+1,i}, x_{2rq+2,i}, \ldots, x_{2rq+2r,i}\} \subseteq S_i$ and $Y_q = \{x_{2rq+1,j}, x_{2rq+2,j}, \ldots, x_{2rq+2r,j}\} \subseteq S_j$ is an instance Γ_q of $K_{2r,2r}$. Γ_q admits a $(2r)$-cycle decomposition D_q that satisfies the properties given in Lemma 2.3.4 (with W_q and Y_q substituted for W and Y, respectively, so that $w_\beta = x_{2rq+\beta,i}$ and $y_\beta = x_{2rq+\beta,j}$ for each $\beta \in \{1, 2, \ldots, 2r\}$). By

interchanging vertices $x_{2rq+r,i}$ and $x_{2rq+(r+1),i}$ (i.e. w_r and w_{r+1}) and vertices $x_{2rq+r,j}$ and $x_{2rq+(r+1),j}$ (i.e. y_r and y_{r+1}) in Γ_q, the decomposition D_q is transformed into another $(2r)$-cycle decomposition D'_q of Γ_q. Observe that D'_q has no $(2r)$-cycle on vertex set $\{x_{2rq+1,i}, x_{2rq+1,j}, x_{2rq+2,i}, x_{2rq+2,j}, \ldots, x_{2rq+r,i}, x_{2rq+r,j}\}$ or on vertex set $\{x_{2rq+(r+1),i}, x_{2rq+(r+1),j}, x_{2rq+(r+2),i}, x_{2rq+(r+2),j}, \ldots, x_{2rq+2r,i}, x_{2rq+2r,j}\}$. The decompositions D_q and D'_q will be used when we construct decompositions of $K_{\ell,\ell}$, which will subsequently be used in the construction of S.

For each $i \in \{1, \ldots, t\}$, let $X_i = \{x_{1,i}, \ldots, x_{\ell,i}\}$. For each $i, j \in \{1, 2, \ldots, t\}$ such that $i < j$, between X_i and X_j is an instance $\Gamma_{i,j}$ of $K_{\ell,\ell}$. Consider the $(2r)$-cycle decomposition $\mathcal{D}_{i,j}$ admitted by $\Gamma_{i,j}$ which is formed in the following manner. For any $q \in \{0, 1, \ldots, \frac{\varsigma}{2}-1\}$, place D'_q between $\{x_{2rq+1,i}, x_{2rq+2,i}, \ldots, x_{2rq+2r,i}\}$ and $\{x_{2rq+1,j}, x_{2rq+2,j}, \ldots, x_{2rq+2r,j}\}$. (If $\varsigma \leq 0$, then $\{0, 1, \ldots, \frac{\varsigma}{2} - 1\} = \emptyset$, so no $(2r)$-cycles are placed in this step.) For any $q \in \{\max\{0, \frac{\varsigma}{2}\}, \ldots, \frac{\ell}{2r} - 1\}$, place D_q between $\{x_{2rq+1,i}, x_{2rq+2,i}, \ldots, x_{2rq+2r,i}\}$ and $\{x_{2rq+1,j}, x_{2rq+2,j}, \ldots, x_{2rq+2r,j}\}$. If $q_1, q_2 \in \{0, 1, \ldots, \frac{\ell}{2r} - 1\}$ where $q_1 \neq q_2$, place an arbitrary $(2r)$-cycle decomposition of $K_{2r,2r}$ between $\{x_{2rq_1+1,i}, \ldots, x_{2rq_1+2r,i}\}$ and $\{x_{2rq_2+1,j}, \ldots, x_{2rq_2+2r,j}\}$. Note that for each r-subset $A = \{a_1, \ldots, a_r\} \subseteq \{1, \ldots, \ell\}$, there exists a $(2r)$-cycle in $\mathcal{D}_{i,j}$ on vertex set $\{x_{a_1,i}, x_{a_1,j}, x_{a_2,i}, x_{a_2,j}, \ldots, x_{a_r,i}, x_{a_r,j}\}$ if and only if A is one of the following sets: $\{2rq+1, \ldots, 2rq+(r+1)\} - \{2rq+r\}$ where $0 \leq q \leq \frac{\varsigma}{2} - 1$, $\{2rq+r, \ldots, 2rq+2r\} - \{2rq+(r+1)\}$ where $0 \leq q \leq \frac{\varsigma}{2} - 1$, or $\{rq+1, \ldots, rq+r\}$ where $\max\{0, \varsigma\} \leq q \leq \frac{\ell}{r} - 1$.

Since the subscripts used on $\mathcal{D}_{i,j}$ correspond naturally to those of X_i and X_j, we hereafter omit the subscripts on $\mathcal{D}_{i,j}$. However, instead of writing $\mathcal{D}$ for $\mathcal{D}_{i,j}$, we instead write $\mathcal{D}_1$ (because additional $(2r)$-cycle decompositions (to be named $\mathcal{D}_2, \ldots, \mathcal{D}_p$) between X_i and X_j will also be introduced).

For an integer $q \geq 1$ and non-negative integers $\theta_1, \theta_2, \ldots, \theta_q$ such that $\theta_1 + \theta_2 + \cdots + \theta_q = \ell$, let ${c_1}^{\theta_1}{c_2}^{\theta_2} \cdots {c_q}^{\theta_q}$ denote the colouring of X_i in which vertices $x_{1,i}, x_{2,i}, \ldots, x_{\theta_1,i}$ have colour c_1, and, for each $j \in \{2, \ldots, q\}$, vertices $x_{\theta_1+\theta_2+\cdots+\theta_{j-1}+1,i}$, $x_{\theta_1+\theta_2+\cdots+\theta_{j-1}+2,i}$, $\ldots$, $x_{\theta_1+\theta_2+\cdots+\theta_{j-1}+\theta_j,i}$ have colour c_j. The colours $c_1, \ldots, c_q$ in the colouring ${c_1}^{\theta_1}{c_2}^{\theta_2} \cdots {c_q}^{\theta_q}$ are not necessarily all distinct. We will use the symbol $*$ to denote that the colour choice for the corresponding vertex is unrestricted.

Let $c \in \{1, 2, \ldots, k-1\}$. For a given $q \in \{0, 1, \ldots, \frac{\varsigma}{2} - 1\}$, consider the patterns $P_{c,q}^{(1)} = *^{2rq}C^{r-1}*C*^{\ell-(2rq+r+1)}$ and $P_{c,q}^{(2)} = *^{2rq+r-1}C*C^{r-1}*^{\ell-(2rq+2r)}$, and for a given $q \in \{\max\{0, \varsigma\}, \ldots, \frac{\ell}{r} - 1\}$, consider the pattern $P_{c,q}^{(3)} = *^{rq}C^{r}*^{\ell-(rq+r)}$. If X_i and X_j have decomposition $\mathcal{D}_1$ between them, at most one of X_i and X_j can be coloured in a pattern matching $P_{c,q}^{(1)}$ for a fixed $q \in \{0, 1, \ldots, \frac{\varsigma}{2} - 1\}$, for otherwise there would be a monochromatic $(2r)$-cycle in the decomposition $\mathcal{D}_1$ between X_i and X_j. Also, at most one of X_i and X_j can be coloured in a pattern matching $P_{c,q}^{(2)}$ for a fixed $q \in \{0, 1, \ldots, \frac{\varsigma}{2} - 1\}$, and at most one of X_i and X_j can be coloured in a pattern matching $P_{c,q}^{(3)}$ for a fixed $q \in \{\max\{0, \varsigma\}, \ldots, \frac{\ell}{r} - 1\}$, as colouring both X_i and X_j both with a colouring matching $P_{c,q}^{(\lambda)}$ for some $\lambda \in \{1, 2\}$ and $q \in \{0, 1, \ldots, \frac{\varsigma}{2} - 1\}$, or both with a colouring matching $P_{c,q}^{(3)}$ for some $q \in \{\max\{0, \varsigma\}, \ldots, \frac{\ell}{r} - 1\}$ would result in a monochromatic cycle. Let

$$\mathcal{P}_{c,1} = \{P_{c,q}^{(1)}, P_{c,q}^{(2)} : 0 \leq q \leq \frac{\varsigma}{2} - 1\} \cup \{P_{c,q}^{(3)} : \max\{0, \varsigma\} \leq q \leq \frac{\ell}{r} - 1\}.$$

Note that $|\mathcal{P}_{c,1}| = \frac{\ell}{r}$.

We now form additional $(2r)$-cycle decompositions admitted by $\Gamma_{i,j} \cong K_{\ell,\ell}$ as follows. Let σ be a permutation of $\{1, 2, \ldots, \ell\}$, and consider the decomposition

$$\begin{aligned}\sigma(\mathcal{D}_1) \;=\; & \{\sigma(C) = (x_{\sigma(u_1),i}, x_{\sigma(u_2),j}, x_{\sigma(u_3),i}, x_{\sigma(u_4),j}, \ldots, x_{\sigma(u_{2r-1}),i}, x_{\sigma(u_{2r}),j}) : \\ & C = (x_{u_1,i}, x_{u_2,j}, x_{u_3,i}, x_{u_4,j}, \ldots, x_{u_{2r-1},i}, x_{u_{2r},j}) \text{ is a } (2r)\text{-cycle in } \mathcal{D}_1\}.\end{aligned}$$

We will say that σ is *acceptable* if its corresponding decomposition $\sigma(\mathcal{D}_1)$ contains no $(2r)$-cycle on vertex set $\{x_{rq+1,i}, x_{rq+1,j}, x_{rq+2,i}, x_{rq+2,j}, \ldots, x_{rq+r,i}, x_{rq+r,j}\}$ for any $q \in \{0, 1, \ldots, \varsigma - 1\}$. Supposing there are p acceptable permutations, denote them by σ_1, σ_2, ..., σ_p (where σ_1 is the identity permutation), and let the decompositions corresponding to σ_2, ..., σ_p be $\mathcal{D}_2$, ..., $\mathcal{D}_p$, respectively, so that $\mathcal{D}_\delta = \sigma_\delta(\mathcal{D}_1)$ for each $\delta \in \{2, \ldots, p\}$.

For each $\alpha \in \{2, \ldots, p\}$ and each colour $c \in \{1, 2, \ldots, k-1\}$, let $\mathcal{P}_{c,\alpha}$ denote the set of colour patterns obtained by applying the permutation σ_α to the colour positions in each pattern in $\mathcal{P}_{c,1}$. Note that, for any given pattern P in $\mathcal{P}_{c,\alpha}$, if X_i and X_j have decomposition $\mathcal{D}_\alpha$ between them, then at most one of X_i and X_j can be coloured in a pattern that satisfies P; otherwise there would be a monochromatic cycle between X_i and X_j. Note that X_i and X_j might each be coloured in a pattern matching some pattern in $\mathcal{P}_{c,\alpha}$, but the colouring of X_i and that of X_j do not match the same pattern in $\mathcal{P}_{c,\alpha}$.

Lemma 2.3.6. *Let $\{a_1, a_2, \ldots, a_r\} \subseteq \{1, 2, \ldots, \ell\}$, where $a_1 < a_2 < \cdots < a_r$, such that $\{a_1, a_2, \ldots, a_r\} \neq \{rq+1, rq+2, \ldots, rq+r\}$ for any $q \in \{0, 1, \ldots, \varsigma - 1\}$. Then for any colour $c \in \{1, 2, \ldots, k-1\}$, the colouring pattern*

$$P = *^{a_1-1}C*^{a_2-a_1-1}C*^{a_3-a_2-1}C \cdots *^{a_r-a_{r-1}-1}C*^{\ell-a_r}$$

is in $\mathcal{P}_{c,\alpha}$ for some $\alpha \in \{1, 2, \ldots, p\}$.

Proof. Note that if $\varsigma \leq 0$, then any permutation of $\{1, 2, \ldots, \ell\}$ is acceptable. In this case, let σ_α be a permutation of $\{1, 2, \ldots, \ell\}$ such that $\sigma_\alpha(\{1, 2, \ldots, r\}) = \{a_1, a_2, \ldots, a_r\}$. The colouring P is in $\mathcal{P}_{c,\alpha}$.

We now assume that $\varsigma > 0$. For each $q \in \{0, 1, \ldots, \frac{\varsigma}{2} - 1\}$, let $A_q = \{2rq+1, \ldots, 2rq+(r+1)\} - \{2rq+r\}$ and let $B_q = \{2rq+r, \ldots, 2rq+2r\} - \{2rq+(r+1)\}$.

If $\{a_1, a_2, \ldots, a_r\} = A_q$ or $\{a_1, a_2, \ldots, a_r\} = B_q$ for some $q \in \{0, 1, \ldots, \frac{\varsigma}{2} - 1\}$, then the colouring P is in $\mathcal{P}_{c,1}$.

Otherwise, let $M = \max_{0 \leq q \leq \varsigma/2-1} \{|A_q \cap \{a_1, \ldots, a_r\}|, |B_q \cap \{a_1, \ldots, a_r\}|\}$. Let $R \in \{A_q, B_q : 0 \leq q \leq \frac{\varsigma}{2} - 1\}$ such that $|R \cap \{a_1, \ldots, a_r\}| = M$. Suppose $a_{i_1}, a_{i_2}, \ldots, a_{i_M} \in \{a_1, a_2, \ldots, a_r\} \cap R$ such that $i_1 < i_2 < \cdots < i_M$. Let $a_{j_1}, a_{j_2}, \ldots, a_{j_{r-M}} \in \{a_1, \ldots, a_r\} - \{a_{i_1}, \ldots, a_{i_M}\}$ such that $j_1 < j_2 < \cdots < j_{r-M}$, and let $b_1, b_2, \ldots, b_{r-M} \in R - \{a_{i_1}, \ldots, a_{i_M}\}$ such that $b_1 < b_2 < \cdots < b_{r-M}$.

We now define three permutations σ_{α_1}, σ_{α_2} and σ_{α_3} of $\{1, 2, \ldots, \ell\}$. It will be shown that for some $\alpha \in \{\alpha_1, \alpha_2, \alpha_3\}$, σ_α is acceptable and the colouring P is in $\mathcal{P}_{c,\alpha}$. Define σ_{α_1} by $\sigma_{\alpha_1}(a_{j_\delta}) = b_\delta$ and $\sigma_{\alpha_1}(b_\delta) = a_{j_\delta}$ for each $\delta \in \{1, 2, \ldots, r - M\}$, and $\sigma_{\alpha_1}(y) = y$ for any $y \in \{1, 2, \ldots, \ell\} - \{a_{i_1}, a_{i_2}, \ldots, a_{i_{r-M}}, b_1, b_2, \ldots, b_{r-M}\}$. Also, if $M < r - 1$, define σ_{α_2} by $\sigma_{\alpha_2}(a_{j_1}) = b_{r-M}$, $\sigma_{\alpha_2}(b_1) = a_{j_{r-M}}$, $\sigma_{\alpha_2}(a_{j_{r-M}}) = b_1$, $\sigma_{\alpha_2}(b_{r-M}) = a_{j_1}$, $\sigma_{\alpha_2}(a_{j_\delta}) = b_\delta$ and $\sigma_{\alpha_2}(b_\delta) = a_{j_\delta}$ if $\delta \in \{2, \ldots, r - M - 1\}$, and $\sigma_{\alpha_2}(y) = y$ for any $y \in \{1, \ldots, \ell\} - \{a_{j_1}, a_{j_2}, \ldots, a_{j_{r-M}}, b_1, b_2, \ldots, b_{r-M}\}$. Finally, if $M = 0$, define σ_{α_3} by $\sigma_{\alpha_3}(\delta) = a_\delta$ and $\sigma_{\alpha_3}(a_\delta) = \delta$ if $\delta \in \{1, 2, \ldots, r - 1\}$, $\sigma_{\alpha_3}(r + 1) = a_r$, $\sigma_{\alpha_3}(a_r) = r + 1$, and $\sigma_{\alpha_3}(y) = y$ for any $y \in \{1, 2, \ldots, \ell\} - (\{1, 2, \ldots, r - 1, r + 1\} \cup \{a_1, a_2, \ldots, a_r\})$. Note that since $\varsigma > 0$, if $M = 0$, then $a_\delta \geq a_1 > 2r \geq r + 1$ for any $\delta \in \{1, \ldots, r\}$, and it follows that σ_{α_3} is well-defined.

Now, suppose $M = r - 1$. If σ_{α_1} is not acceptable, then for some $q \in \{0, 1, \ldots, \frac{\varsigma}{2} - 1\}$, either $a_{j_1} = 2rq + r$ and $b_1 = 2rq + (r+1)$ (so that $\{a_{i_1}, \ldots, a_{i_{r-1}}\} = \{2rq + 1, 2rq + 2, \ldots, 2rq + (r - 1)\}$), or else $a_{j_1} = 2rq + (r + 1)$ and $b_1 = 2rq + r$ (so that $\{a_{i_1}, \ldots, a_{i_{r-1}}\} = \{2rq + (r + 2), 2rq + (r + 3), \ldots, 2rq + 2r\}$). In either case, $\{a_1, a_2, \ldots, a_r\} = \{rq + 1, rq + 2, \ldots, rq + r\}$ for some $q \in \{0, 1, \ldots, \varsigma - 1\}$, which is a contradiction to the hypothesis of the lemma. So σ_{α_1} is acceptable. Furthermore, there exists a colouring pattern $Q \in \mathcal{P}_{c,1}$ such that position z of Q has symbol c if

and only if $z \in R$. Since $\sigma_{\alpha_1}(R) = \{a_1, \ldots, a_r\}$, then σ_{α_1} maps those positions of Q that contain c onto the positions of P that contain c, and hence $P \in \mathcal{P}_{c,\alpha_1}$.

Next, suppose $1 < M < r-1$ and that, for some $q \in \{0, 1, \ldots, \frac{\varsigma}{2} - 1\}$, both $b_{r-M} = 2rq + (r+1)$ and $a_{j_{r-M}} = 2rq + r$, or both $b_1 = 2rq + r$ and $a_{j_1} = 2rq + (r+1)$. Consider the permutation σ_{α_2}. The definition of this permutation ensures that for this q, $\sigma_{\alpha_2}(2rq+r) \neq 2rq+(r+1)$ and $\sigma_{\alpha_2}(2rq+(r+1)) \neq 2rq + r$. We first suppose $b_{r-M} = 2rq + (r+1)$ and $a_{j_{r-M}} = 2rq + r$ for some $q \in \{0, 1, \ldots, \frac{\varsigma}{2} - 1\}$, and assume that σ_{α_2} is not acceptable. We note that $R = A_q$ and $a_{j_\delta} \leq 2rq$ whenever $1 \leq \delta \leq r-M-1$. If the $(2r)$-cycle decomposition $\sigma_{\alpha_2}(\mathcal{D}_1)$ has a $(2r)$-cycle on vertex set $\{x_{2rq+1,i}, x_{2rq+1,j}, x_{2rq+2,i}, x_{2rq+2,j}, \ldots, x_{2rq+r,i}, x_{2rq+r,j}\}$, then since σ_{α_2} fixes some element of $\{2rq+1, \ldots, 2rq+(r-1)\}$ (because $M \geq 2$), it must be that σ_{α_2} maps $2rq + (r+1)$ into $\{2rq+1, \ldots, 2rq+r\}$, necessitating $\sigma_{\alpha_2}(2rq + (r+1)) = 2rq + r$, which is a contradiction. If $\sigma_{\alpha_2}(\mathcal{D}_1)$ has a $(2r)$-cycle on $\{x_{2rq+(r+1),i}, x_{2rq+(r+1),j}, x_{2rq+(r+2),i}, x_{2rq+(r+2),j}, \ldots, x_{2rq+2r,i}, x_{2rq+2r,j}\}$, then since σ_{α_2} fixes each element of $\{2rq + (r+2), \ldots, 2rq + 2r\}$, it must be that σ_{α_2} maps $2rq + r$ to $2rq + (r+1)$, which is a contradiction. The permutation σ_{α_2} fixes δ whenever $\delta > 2rq + 2r$, and so $\sigma_{\alpha_2}(\mathcal{D}_1)$, like $\mathcal{D}_1$, has no $(2r)$-cycle on any of the sets $\{x_{r\bar{q}+1,i}, x_{r\bar{q}+1,j}, x_{r\bar{q}+2,i}, x_{r\bar{q}+2,j} \ldots, x_{r\bar{q}+r,i}, x_{r\bar{q}+r,j}\}$ for any $\bar{q} \in \{2q+2, \ldots, \varsigma - 1\}$. So, for some $\bar{q} \in \{0, \ldots, 2q-1\}$, $\sigma_{\alpha_2}(\mathcal{D}_1)$ must contain a $(2r)$-cycle C on vertex set $\{x_{r\bar{q}+1,i}, x_{r\bar{q}+1,j}, x_{r\bar{q}+2,i}, x_{r\bar{q}+2,j}, \ldots, x_{r\bar{q}+r,i}, x_{r\bar{q}+r,j}\}$. Necessarily, there exists an element $\phi \in \{1, \ldots, r-M\}$ such that $a_{j_\phi} \in \{r\bar{q}+1, \ldots, r\bar{q}+r\}$. Moreover, $\{r\bar{q}+1, \ldots, r\bar{q}+r\}$ contains at least one element, f, which is fixed by σ_{α_2}. Since $\sigma_{\alpha_2}^{-1} = \sigma_{\alpha_2}$, $\sigma_{\alpha_2}(C)$ is a $(2r)$-cycle of $\mathcal{D}_1$ such that $\sigma_{\alpha_2}(C)$ contains both $x_{f,i}$ and $x_{\sigma_{\alpha_2}(a_{j_\phi}),i} = x_{b_\psi,i}$ for some $\psi \in \{1, \ldots, r-M\}$, which is a contradiction because $\mathcal{D}_1$ has no $(2r)$-cycle containing both vertices $x_{f,i}$ and $x_{b_\psi,i}$

(since $f \in \{r\bar{q}+1, \ldots, r\bar{q}+r\}$, and either $b_\psi \in \{r(2q)+1, \ldots, r(2q)+r\}$ or else $b_\psi = 2rq+(r+1) \in \{r(2q+1)+1, \ldots, r(2q+1)+r\}$). Hence σ_{α_2} is acceptable.

In the case that $b_1 = 2rq+r$ and $a_{j_1} = 2rq+(r+1)$ for some $q \in \{0, 1, \ldots, \frac{\varsigma}{2}-1\}$, we note that $R = B_q$ and $a_{j_\delta} \geq 2rq+2r$ whenever $2 \leq \delta \leq r-M$; that σ_{α_2} is acceptable may be shown in a similar manner to the case that $b_{r-M} = 2rq+(r+1)$ and $a_{j_{r-M}} = 2rq+r$. Now, in either of these cases, there exists a colouring pattern $Q \in \mathcal{P}_{c,1}$ such that position z of Q has symbol c if and only if $z \in R$. Since $\sigma_{\alpha_2}(R) = \{a_1, \ldots, a_r\}$, then σ_{α_2} maps those positions of Q that contain c onto the positions of P that contain c, and hence $P \in \mathcal{P}_{c,\alpha_2}$.

We next suppose that $1 < M < r-1$, and that for all $q \in \{0, 1, \ldots, \frac{\varsigma}{2}-1\}$, it is not true that both $b_{r-M} = 2rq+(r+1)$ and $a_{j_{r-M}} = 2rq+r$, and it is not true that both $b_1 = 2rq+r$ and $a_{j_1} = 2rq+(r+1)$. Let $q \in \{0, 1, \ldots, \frac{\varsigma}{2}-1\}$ such that $R \in \{A_q, B_q\}$. We note that $\sigma_{\alpha_1}(2rq+r) \neq 2rq+(r+1)$ and $\sigma_{\alpha_1}(2rq+(r+1)) \neq 2rq+r$ (as this would imply that both $a_{j_1} = 2qr+r$ and $b_1 = 2qr+(r+1)$ or both $a_{j_{r-M}} = 2rq+(r+1)$ and $b_{r-M} = 2rq+r$, either of which yields that $M = 1$). Assume that the permutation σ_{α_1} is not acceptable. We first consider the case that $R = A_q$. If the $(2r)$-cycle decomposition $\sigma_{\alpha_1}(\mathcal{D}_1)$ has a $(2r)$-cycle on vertex set $\{x_{2rq+1,i}, x_{2rq+1,j}, x_{2rq+2,i}, x_{2rq+2,j}, \ldots, x_{2rq+r,i}, x_{2rq+r,j}\}$, then since σ_{α_1} fixes some element of $\{2rq+1, \ldots, 2rq+(r-1)\}$ (because $M \geq 2$), it must be that σ_{α_1} maps $2rq+(r+1)$ into $\{2rq+1, \ldots, 2rq+r\}$, necessitating $\sigma_{\alpha_1}(2rq+(r+1)) = 2rq+r$, which is a contradiction. If $\sigma_{\alpha_1}(\mathcal{D}_1)$ has a $(2r)$-cycle on $\{x_{2rq+(r+1),i}, x_{2rq+(r+1),j}, x_{2rq+(r+2),i}, x_{2rq+(r+2),j}, \ldots, x_{2rq+2r,i}, x_{2rq+2r,j}\}$, then since σ_{α_1} fixes some element of $\{2rq+(r+2), \ldots, 2rq+2r\}$, it must be that σ_{α_1} maps $2rq+r$ into $\{2rq+(r+1), \ldots, 2rq+2r\}$, necessitating $\sigma_{\alpha_1}(2rq+r) = 2rq+(r+1)$, which is a contradiction. So, for some $\bar{q} \in \{0, 1, \ldots, 2q-1\} \cup \{2q+2, \ldots, \varsigma-1\}$,

$\sigma_{\alpha_1}(\mathcal{D}_1)$ contains a $(2r)$-cycle C on $\{x_{r\bar{q}+1,i}, x_{r\bar{q}+1,j}, x_{r\bar{q}+2,i}, x_{r\bar{q}+2,j}, \ldots, x_{r\bar{q}+r,i}, x_{r\bar{q}+r,j}\}$. Necessarily, there exists an element $\phi \in \{1, \ldots, r-M\}$ such that $a_{j_\phi} \in \{r\bar{q}+1, \ldots, r\bar{q}+r\}$. Moreover, $\{r\bar{q}+1, \ldots, r\bar{q}+r\}$ contains at least one element, f, fixed by σ_{α_1}. Since $\sigma_{\alpha_1}^{-1} = \sigma_{\alpha_1}$, $\sigma_{\alpha_1}(C)$ is a $(2r)$-cycle of $\mathcal{D}_1$ which contains $x_{f,i}$ and $x_{\sigma_{\alpha_1}(a_{j_\phi}),i} = x_{b_\psi,i}$ for some $\psi \in \{1, \ldots, r-M\}$, which is a contradiction, because $\mathcal{D}_1$ has no $(2r)$-cycle containing both vertices $x_{f,i}$ and $x_{b_\psi,i}$ (since $f \in \{r\bar{q}+1, \ldots, r\bar{q}+r\}$, and either $b_\psi \in \{r(2q)+1, \ldots, r(2q)+r\}$ or $b_\psi = 2rq+(r+1) \in \{r(2q+1)+1, \ldots, r(2q+1)+r\}$). Hence σ_{α_1} is acceptable. In the case that $R = B_q$, that σ_{α_1} is acceptable may be shown in a similar manner. Now, in either case, there exists a colouring pattern $Q \in \mathcal{P}_{c,1}$ such that position z of Q has symbol c if and only if $z \in R$. Since $\sigma_{\alpha_1}(R) = \{a_1, \ldots, a_r\}$, then σ_{α_1} maps those positions of Q which contain symbol c onto the positions of P that contain c, and hence $P \in \mathcal{P}_{c,\alpha_1}$.

Now, suppose that $M = 1$ and that, for some $q \in \{0, 1, \ldots, \frac{s}{2} - 1\}$, both $b_{r-M} = 2rq + (r+1)$ and $a_{j_{r-M}} = 2rq + r$, or both $b_1 = 2rq + r$ and $a_{j_1} = 2rq + (r+1)$. Consider the permutation σ_{α_2}. The definition of this permutation ensures that for this q, $\sigma_{\alpha_2}(2rq + r) \neq 2rq + (r+1)$ and $\sigma_{\alpha_2}(2rq + (r+1)) \neq 2rq + r$. We first suppose that $b_{r-M} = 2rq + (r+1)$ and $a_{j_{r-M}} = 2rq + r$ for some $q \in \{0, \ldots, \frac{s}{2} - 1\}$. We note that $R = A_q$ and $a_{j_\delta} \leq 2rq$ whenever $1 \leq \delta \leq r - M - 1$. Furthermore, since $b_{r-M} = 2rq + (r+1)$ is not fixed by σ_{α_2}, σ_{α_2} fixes an element of $R - \{2rq + (r+1)\} = \{2rq + 1, \ldots, 2rq + (r-1)\}$. That σ_{α_2} is acceptable is similar to the case that $1 < M < r - 1$, $b_{r-M} = 2rq + (r+1)$ and $a_{j_{r-M}} = 2rq + r$. If $b_1 = 2rq + r$ and $a_{j_1} = 2rq + (r+1)$, σ_{α_2} is similarly acceptable. In either case, there exists a colouring pattern $Q \in \mathcal{P}_{c,1}$ such that position z of Q has symbol c if and only if $z \in R$. Since $\sigma_{\alpha_2}(R) = \{a_1, \ldots, a_r\}$, then σ_{α_2} maps those positions of Q which contain symbol c onto the positions of P that contain c, and hence $P \in \mathcal{P}_{c,\alpha_2}$.

Next, suppose $M = 1$ and for each $q \in \{0, 1, \ldots, \frac{\varsigma}{2} - 1\}$, it is not true that both $b_{r-M} = 2rq + (r+1)$ and $a_{j_{r-M}} = 2rq + r$, and it is not true that both $b_1 = 2rq + r$ and $a_{j_1} = 2rq + (r+1)$. Let $q \in \{0, 1, \ldots, \frac{\varsigma}{2} - 1\}$ such that $R \in \{A_q, B_q\}$. We note that $\sigma_{\alpha_1}(2rq + r) \neq 2rq + (r+1)$ and $\sigma_{\alpha_1}(2rq + (r+1)) = 2rq + r$. First, consider the case that $R = A_q$. Assume that the permutation σ_{α_1} is not acceptable.

As the first of three subcases, suppose that $\sigma_{\alpha_1}(\mathcal{D}_1)$ has a $(2r)$-cycle on vertex set $\{x_{2rq+1,i}, x_{2rq+1,j}, x_{2rq+2,i}, x_{2rq+2,j}, \ldots, x_{2rq+r,i}, x_{2rq+r,j}\}$. If σ_{α_1} fixes some element of $\{2rq + 1, \ldots, 2rq + (r-1)\}$, then it must be that σ_{α_1} maps $2rq + (r+1)$ into $\{2rq + 1, \ldots, 2rq + r\}$, necessitating $\sigma_{\alpha_1}(2rq + (r+1)) = 2rq + r$, which is a contradiction. If σ_{α_1} fixes no element of $\{2rq + 1, \ldots, 2rq + (r-1)\}$, then $2rq + (r+1)$ must be fixed by σ_{α_1}, and we must have $a_{i_1} = 2rq + (r+1)$. Furthermore, $2rq+r$ cannot be mapped into $\{2rq+1, \ldots, 2rq+(r-1)\}$ since this would imply that $\sigma_{\alpha_1}^{-1}(2rq+r) \in \{2rq+1, \ldots, 2rq+(r-1)\}$, which would mean that $\sigma_{\alpha_1}(\mathcal{D}_1)$ could have no $(2r)$-cycle on $\{x_{2rq+1,i}, x_{2rq+1,j}, x_{2rq+2,i}, x_{2rq+2,j}, \ldots, x_{2rq+r,i}, x_{2rq+r,j}\}$ (as $\mathcal{D}_1$ contains no $(2r)$-cycle on vertex set $\{x_{\mu_1,i}, x_{\mu_1,j}, x_{\mu_2,i}, x_{\mu_2,j}, \ldots, x_{\mu_r,i}, x_{\mu_r,j}\}$ where $2qr + r, \sigma^{-1}(2qr + r) \in \{\mu_1, \ldots, \mu_r\}$ in such a case). Thus, $2rq + r$ must be fixed by σ_{α_1}, and it must be that $\sigma_{\alpha_1}(\{2rq+(r+2), \ldots, 2rq+2r\}) = \{2rq+1, \ldots, 2rq+(r-1)\}$ and hence that $\{a_1, \ldots, a_r\} = \{2rq + (r+1), 2rq + (r+2), \ldots, 2rq + 2r\}$, which is a contradiction to the hypothesis of the lemma.

Now suppose that the $(2r)$-cycle decomposition $\sigma_{\alpha_1}(\mathcal{D}_1)$ has a $(2r)$-cycle on vertex set $\{x_{2rq+(r+1),i}, x_{2rq+(r+1),j}, x_{2rq+(r+2),i}, x_{2rq+(r+2),j}, \ldots, x_{2rq+2r,i}, x_{2rq+2r,j}\}$. If σ_{α_1} fixes some element of $\{2rq + (r+2), \ldots, 2rq + 2r\}$, it must be that σ_{α_1} maps $2rq + r$ into $\{2rq + (r+1), \ldots, 2rq + 2r\}$, necessitating $\sigma_{\alpha_1}(2rq + r) = 2rq + (r+1)$, which is a contradiction. If σ_{α_1} fixes no element of $\{2rq + (r+2), \ldots, 2rq + 2r\}$, then $2rq + r$ must be fixed by σ_{α_1}, and it must be that $\{2rq + (r+2), \ldots, 2rq + 2r\} =$

$\{a_{j_1}, \ldots, a_{j_{r-1}}\}$. Furthermore, $2rq + (r+1)$ cannot be mapped into $\{2rq + (r+2), \ldots, 2rq + 2r\}$ since this would imply that $\sigma_{\alpha_1}^{-1}(2rq + (r+1)) \in \{2rq+(r+2), \ldots, 2rq+2r\}$, which would mean that $\sigma_{\alpha_1}(\mathcal{D}_1)$ could have no $(2r)$-cycle on $\{x_{2rq+(r+1),i}, x_{2rq+(r+1),j}, x_{2rq+(r+2),i}, x_{2rq+(r+2),j}, \ldots, x_{2rq+2r,i}, x_{2rq+2r,j}\}$. implying $a_{i_1} = 2rq + (r+1)$, and hence that $\{a_1, \ldots, a_r\} = \{2rq + (r+1), \ldots, 2rq + 2r\}$, which is a contradiction to the hypothesis of the lemma.

So it must be true that for some $\bar{q} \in \{0, \ldots, 2q-1\} \cup \{2q+2, \ldots, \varsigma - 1\}$, $\sigma_{\alpha_1}(\mathcal{D}_1)$ contains a $(2r)$-cycle C on $\{x_{r\bar{q}+1,i}, x_{r\bar{q}+1,j}, x_{r\bar{q}+2,i}, x_{r\bar{q}+2,j}, \ldots, x_{r\bar{q}+r,i}, x_{r\bar{q}+r,j}\}$. Necessarily, there exists an element $\phi \in \{1, \ldots, r-1\}$ such that $a_{j_\phi} \in \{r\bar{q}+1, \ldots, r\bar{q}+r\}$. Moreover, $\{r\bar{q}+1, \ldots, r\bar{q}+r\}$ contains at least one element, f, which is fixed by σ_{α_1}. Since $\sigma_{\alpha_1}^{-1} = \sigma_{\alpha_1}$, $\sigma_{\alpha_1}(C)$ is a $(2r)$-cycle of $\mathcal{D}_1$ which contains both $x_{f,i}$ and $x_{\sigma_{\alpha_1}(a_{j_\phi}),i} = x_{b_\psi,i}$ for some $\psi \in \{1, \ldots, r-1\}$, which is a contradiction because $\mathcal{D}_1$ has no $(2r)$-cycle containing both vertices $x_{f,i}$ and $x_{b_\psi,i}$ (since $f \in \{r\bar{q}+1, \ldots, r\bar{q}+r\}$, and either $b_\psi \in \{r(2q)+1, \ldots, r(2q)+r\}$ or $b_\psi = 2rq + (r+1) \in \{r(2q+1)+1, \ldots, r(2q+1)+r\}$). Hence σ_{α_1} is acceptable.

In the case that $R = B_q$, that σ_{α_1} is acceptable may be shown in a similar manner. In either case, there exists a colouring pattern $Q \in \mathcal{P}_{c,1}$ such that position z of Q has symbol c if and only if $z \in R$. Since $\sigma_{\alpha_1}(R) = \{a_1, \ldots, a_r\}$, then σ_{α_1} maps those positions of Q which contain symbol c onto the positions of P that contain c, and hence $P \in \mathcal{P}_{c,\alpha_1}$.

Finally, suppose $M = 0$, and consider the permutation σ_{α_3}. Note that σ_{α_3} fixes δ whenever $2r+1 \leq \delta \leq \varsigma r$, and so $\sigma_{\alpha_3}(\mathcal{D}_1)$, like $\mathcal{D}_1$, contains no $(2r)$-cycle on vertex set $\{x_{r\bar{q}+1,i}, x_{r\bar{q}+1,j}, x_{r\bar{q}+2,i}, x_{r\bar{q}+2,j}, \ldots, x_{r\bar{q}+r,i}, x_{r\bar{q}+r,j}\}$ for any $\bar{q} \in \{2, 3, \ldots, \varsigma - 1\}$. Furthermore, $\mathcal{D}_{\alpha_3}$ contains no $(2r)$-cycle on vertex set $\{x_{r+1,i}, x_{r+1,j}, x_{r+2,i}, x_{r+2,j}, \ldots, x_{2r,i}, x_{2r,j}\}$ (since $\mathcal{D}_1$ has no $(2r)$-cycle on vertex set

$\{x_{a_r,i}, x_{a_r,j}, x_{r+2,i}, x_{r+2,j}, x_{r+3,i}, x_{r+3,j}, \ldots, x_{2r,i}, x_{2r,j}\}$), and no $(2r)$-cycle on vertex set $\{x_{1,i}, x_{1,j}, x_{2,i}, x_{2,j}, \ldots, x_{r,i}, x_{r,j}\}$ (since $\mathcal{D}_1$ has no $(2r)$-cycle on vertex set $\{x_{a_1,i}, x_{a_1,j}, x_{a_2,i}, x_{a_2,j}, \ldots, x_{a_{r-1},i}, x_{a_{r-1},j}, x_{r,i}, x_{r,j}\}$). Thus, σ_{α_3} is acceptable. Consider the pattern $Q = c^{r-1} * c^{*\ell-(r+1)} \in \mathcal{P}_{c,1}$. The permutation σ_{α_3} maps those positions of Q which contain symbol c onto the positions of P that contain c, and hence $P \in \mathcal{P}_{c,\alpha_3}$. □

Lemma 2.3.7. *Let $i \in \{1, 2, \ldots, t\}$. Any $(k-1)$-colouring of X_i matches a pattern in $\mathcal{P}_{c,\alpha}$ for some $c \in \{1, 2, \ldots, k-1\}$, $\alpha \in \{1, 2, \ldots, p\}$.*

Proof. Consider a fixed $(k-1)$-colouring of X_i. As a consequence of Lemma 2.3.6, it need only be shown that r vertices $x_{a_1,i}, x_{a_2,i}, \ldots, x_{a_r,i}$ have the same colour, where $\{a_1, a_2, \ldots, a_r\} \neq \{rq+1, rq+2, \ldots, rq+r\}$ for each $q \in \{0, 1, \ldots, \varsigma - 1\}$.

Since $\ell > (r-1)(k-1)$, in any $(k-1)$-colouring of X_i, there must be at least r vertices of the same colour. As a first case, suppose that for some $c \in \{1, 2, \ldots, k-1\}$, there are $r+1$ vertices, say $x_{a_1,i}, x_{a_2,i}, \ldots, x_{a_{r+1},i}$, which all have colour c. If, for each $q \in \{0, 1, \ldots, \varsigma - 1\}$, $\{a_1, a_2, \ldots, a_r\} \neq \{rq+1, rq+2, \ldots, rq+r\}$, then we are done. If, on the other hand, $\{a_1, \ldots, a_r\} = \{r\bar{q}+1, \ldots, r\bar{q}+r\}$ for some $\bar{q} \in \{0, 1, \ldots, \varsigma-1\}$, then $\{a_1, a_2, \ldots, a_{r-1}, a_{r+1}\} \neq \{rq+1, \ldots, rq+r\}$ for each $q \in \{0, 1, \ldots, \varsigma - 1\}$.

Next, suppose that no $r+1$ vertices all have the same colour. Considering the case $\varsigma \leq 0$, let vertices $x_{a_1,i}, x_{a_2,i}, \ldots, x_{a_r,i}$ all have the same colour. Since $\varsigma \leq 0$, Lemma 2.3.6 is satisfied vacuously. In the case that $\varsigma > 0$, we note that $\ell = (r-1)(k-1) + (r-1+s)$, so there are at least $r-1+s$ disjoint sets of r vertices of the same colour. But $\left|\{\{rq+1, \ldots, rq+r\} : 0 \leq q \leq \varsigma - 1\}\right| = \varsigma < r-1+s$, so since $r \geq 3$, there must exist r vertices $x_{a_1,i}, x_{a_2,i}, \ldots, x_{a_r,i}$ of the same colour such that for any $q \in \{0, 1, \ldots, \varsigma - 1\}$, $\{a_1, \ldots, a_r\} \neq \{rq+1, rq+2, \ldots, rq+r\}$. □

Now let $t = \left(\frac{\ell}{r}(k-1)+1\right)^p$. We will take t sets S_i, $i \in \{1, 2, \ldots, t\}$, and use the p decompositions $\mathcal{D}_1$, $\mathcal{D}_2$, $\ldots$, $\mathcal{D}_p$ to place a $(2r)$-cycle decomposition of $K_{h,h}$ between each pair of sets S_i and S_j. We proceed in p steps to form the required $(2r)$-cycle decompositions.

Step 1. We take $\gamma = \frac{\ell}{r}(k-1)+1$ sets $S_1, S_2, \ldots, S_\gamma$. For any $1 \leq i < j \leq \gamma$, place $\mathcal{D}_1$ between X_i and X_j. If $h > \ell$, complete the decomposition of the $K_{h,h}$ between S_i and S_j by using any $(2r)$-cycle decomposition of $K_{2r,2r}$ between $S_i - X_i$ and $S_j - X_j$ and any $(2r)$-cycle decomposition of $K_{2r,\ell}$ between X_i and $S_j - X_j$ and between X_j and $S_i - X_i$. (Such decompositions exist by Theorem 2.2.1.) Let R_1 denote the resulting configuration.

Step n ($2 \leq n \leq p$). Suppose we have completed Step $(n-1)$. We now take γ copies of R_{n-1}, so that we now have γ^n sets of vertices, with sets $S_{(i-1)\gamma^{n-1}+1}$, $S_{(i-1)\gamma^{n-1}+2}$, $\ldots$, $S_{(i-1)\gamma^{n-1}+\gamma^{n-1}}$ being in the i^{th} copy of R_{n-1}, $i \in \{1, 2, \ldots, \gamma\}$. The $(2r)$-cycle decomposition of $K_{h,h}$ between $S_{(i-1)\gamma^{n-1}+d_1}$ and $S_{(i-1)\gamma^{n-1}+d_2}$, where $i \in \{1, 2, \ldots, \gamma\}$, $d_1, d_2 \in \{1, 2, \ldots, \gamma^{n-1}\}$ and $d_1 \neq d_2$, is inherited from R_{n-1} and therefore makes use of one of the decompositions $\mathcal{D}_1$, $\mathcal{D}_2$, $\ldots$, $\mathcal{D}_{n-1}$. Place $\mathcal{D}_n$ between any pair of sets $X_{(i-1)\gamma^{n-1}+d_1}$ and $X_{(j-1)\gamma^{n-1}+d_2}$, where $i, j \in \{1, 2, \ldots, \gamma\}$ such that $i < j$ and $d_1, d_2 \in \{1, 2, \ldots, \gamma^{n-1}\}$. If $h > \ell$, then for any pair $S_{(i-1)\gamma^{n-1}+d_1}$ and $S_{(j-1)\gamma^{n-1}+d_2}$ where $i, j \in \{1, 2, \ldots, \gamma\}$ such that $i \neq j$ and $d_1, d_2 \in \{1, 2, \ldots, \gamma^{n-1}\}$, complete the $(2r)$-cycle decomposition of $K_{h,h}$ between $S_{(i-1)\gamma^{n-1}+d_1}$ and $S_{(j-1)\gamma^{n-1}+d_2}$ using any $(2r)$-cycle decomposition of $K_{2r,2r}$ between $S_{(i-1)\gamma^{n-1}+d_1} - X_{(i-1)\gamma^{n-1}+d_1}$ and $S_{(j-1)\gamma^{n-1}+d_2} - X_{(j-1)\gamma^{n-1}+d_2}$, and any $(2r)$-cycle decomposition of $K_{2r,\ell}$ between $S_{(i-1)\gamma^{n-1}+d_1} - X_{(i-1)\gamma^{n-1}+d_1}$ and $X_{(j-1)\gamma^{n-1}+d_2}$ and between $S_{(j-1)\gamma^{n-1}+d_2} - X_{(j-1)\gamma^{n-1}+d_2}$ and $X_{(i-1)\gamma^{n-1}+d_1}$.

After completing Step p, we now have t sets S_i, $i \in \{1, 2, \ldots, t\}$, and for any $1 \leq i < j \leq t$, the copy of $K_{h,h}$ between S_i and S_j has been given a $(2r)$-cycle decomposition. Finally, to complete the $(2r)$-cycle system, for any $i \in \{1, 2, \ldots, t\}$, we place on $\{\infty\} \cup S_i$ any $(2r)$-cycle system of order $h + 1$. (Such a system exists by [34, 44].)

Lemma 2.3.8. *The $(2r)$-cycle system S which we have constructed is not $(k-1)$-colourable.*

Proof. Assume that S is $(k-1)$-colourable, and assign an arbitrary $(k-1)$-colouring.

For each $i, j \in \{1, \ldots, \gamma\}$ such that $i < j$, and for each $d_1, d_2 \in \{1, 2, \ldots, \gamma^{p-1}\}$, $X_{(i-1)\gamma^{p-1}+d_1}$ and $X_{(j-1)\gamma^{p-1}+d_2}$ have decomposition $\mathcal{D}_p$ between them. Thus, for any given colour $c \in \{1, 2, \ldots, k-1\}$, and any given pattern P in $\mathcal{P}_{c,p}$, there is at most one $i \in \{1, \ldots, \gamma\}$ such that the colouring of $X_{(i-1)\gamma^{p-1}+d}$ matches P for some $d \in \{1, 2, \ldots, \gamma^{p-1}\}$. It follows, since $\gamma = |\mathcal{P}_{c,p}|(k-1)+1$ for each $c \in \{1, \ldots, k-1\}$, that there exists an $i_p \in \{1, 2, \ldots, \gamma\}$ such that for every $d \in \{1, 2, \ldots, \gamma^{p-1}\}$, the colouring of $X_{(i_p-1)\gamma^{p-1}+d}$ matches no pattern in $\mathcal{P}_{c,p}$ for any $c \in \{1, 2, \ldots, k-1\}$. Without loss of generality, it may be assumed that $i_p = 1$.

Continue in an inductive fashion. Suppose that, for some $n \in \{3, 4, \ldots, p\}$, we have determined that for any $d \in \{1, 2, \ldots, \gamma^{n-1}\}$, the colouring of X_d matches no pattern in $\mathcal{P}_{c,q}$ for any $c \in \{1, 2, \ldots, k-1\}$, $q \in \{n, n+1, \ldots, p\}$. Note that for each $i, j \in \{1, 2, \ldots, \gamma\}$ such that $i < j$, and any $d_1, d_2 \in \{1, 2, \ldots, \gamma^{n-2}\}$, $X_{(i-1)\gamma^{n-2}+d_1}$ and $X_{(j-1)\gamma^{n-2}+d_2}$ have decomposition $\mathcal{D}_{n-1}$ between them. Thus, for any given colour $c \in \{1, \ldots, k-1\}$ and any given pattern P in $\mathcal{P}_{c,n-1}$, there is at most one $i \in \{1, \ldots, \gamma\}$ such that the colouring of $X_{(i-1)\gamma^{n-2}+d}$ matches P for some $d \in \{1, 2, \ldots, \gamma^{n-2}\}$. It follows that there exists an $i_{n-1} \in \{1, 2, \ldots, \gamma\}$ such that for every $d \in \{1, 2, \ldots, \gamma^{n-2}\}$, the colouring of $X_{(i_{n-1}-1)\gamma^{n-2}+d}$ matches no pattern in

$\mathcal{P}_{c,n-1}$ for any $c \in \{1, 2, \ldots, k-1\}$, and hence no pattern in $\mathcal{P}_{c,q}$ for any $c \in \{1, 2, \ldots, k-1\}$, $q \in \{n-1, n, n+1, \ldots, p\}$. Without loss of generality, it may be assumed that $i_{n-1} = 1$.

We continue until we have determined that for any $d \in \{1, 2, \ldots, \gamma\}$, the colouring of X_d matches no pattern in $\mathcal{P}_{c,q}$ for any $c \in \{1, 2, \ldots, k-1\}$, $q \in \{2, 3, \ldots, p\}$. Note that for each $i, j \in \{1, 2, \ldots, \gamma\}$ such that $i < j$, X_i and X_j have decomposition $\mathcal{D}_1$ between them. Thus, for any given colour $c \in \{1, 2, \ldots, k-1\}$ and any given pattern P in $\mathcal{P}_{c,1}$, there is at most one $i \in \{1, \ldots, \gamma\}$ such that the colouring of X_i matches the pattern P. It follows that there exists an $i_1 \in \{1, 2, \ldots, \gamma\}$ such that the colouring of X_{i_1} matches no pattern in $\mathcal{P}_{c,1}$ for any colour $c \in \{1, 2, \ldots, k-1\}$, and hence no pattern in $\mathcal{P}_{c,q}$ for any $c \in \{1, 2, \ldots, k-1\}$, $q \in \{1, 2, \ldots, p\}$, which contradicts Lemma 2.3.7. □

Lemma 2.3.9. *The $(2r)$-cycle system S which we have constructed is k-colourable.*

Proof. We consider the following cases.

Case 1. $s \leq 0$.

Give ∞ colour 1. For each $i \in \{1, 2, \ldots, t\}$, give vertices $x_{1,i}, x_{2,i}, \ldots, x_{r-1,i}$ colour 1. Note that $\ell - (r-1) = (r-1)k + s - (r-1) \leq (r-1)(k-1)$, so the $\ell - (r-1)$ vertices $x_{r,i}, x_{r+1,i}, \ldots, x_{\ell,i}$ may be coloured with colours $2, 3, \ldots, k$ such that no r of them have the same colour. Finally, if $h > \ell$, give vertices $x_{\ell+1,i}$ and $x_{\ell+2,i}$ colour 1, and colour the remaining vertices $x_{\ell+3,i}, x_{\ell+4,i}, \ldots, x_{h,i}$ with colours 2 and 3, each used exactly $(r-1)$ times.

Note that for each $i \in \{1, 2, \ldots, t\}$, no $(2r)$ vertices in $S_i \cup \{\infty\}$ have the same colour, as $r \geq 3$ and there are at most $r+2$ vertices of colour 1, at most $2r-2$ vertices of colour 2, at most $2r-2$ vertices of colour 3 and at most $r-1$ vertices of any other

colour. Thus, no $(2r)$-cycle in the $(2r)$-cycle system on $S_i \cup \{\infty\}$ has a monochromatic cycle. Let $i, j \in \{1, 2, \ldots, t\}$, $i < j$. No r vertices in X_i have the same colour, so no $(2r)$-cycle in the decomposition $\mathcal{D}_\alpha$ between X_i and X_j is monochromatic. If $h > \ell$, then no $(2r)$-cycle of the $(2r)$-cycle decomposition between X_i and $S_j - X_j$, the $(2r)$-cycle decomposition between X_j and $S_i - X_i$ or the $(2r)$-cycle decomposition between $S_i - X_i$ and $S_j - X_j$ is monochromatic, as no r vertices in $S_i - X_i$ have the same colour, and no r vertices in $S_j - X_j$ have the same colour. Thus no $(2r)$-cycle in S is monochromatic.

Case 2. $s > 0$.

We note that if $k \leq r$, then k must be even; otherwise, if $k \leq r$ and k is odd, then $3 \leq k \leq r$ implies $r(k-1)-2r < r(k-1)+(1-r) \leq r(k-1)+(1-k) = (r-1)(k-1) \leq r(k-1)-2 < r(k-1)$ and so $\ell = r(k-1)$ and $s = k-r \leq 0$. Furthermore, if $3 \leq k \leq r$, we have that $rk-2r < rk-2r+1 \leq rk-r-k+1 = (r-1)(k-1) \leq rk-r-2 < rk$, and so $\ell = rk$ and $s = k$. Also, if $k > r$, then $k \geq r+1 \geq s$. So, in any event, we have $k \geq s$.

For each $i \in \{1, 2, \ldots, t\}$, we colour the vertices of S_i in the following way. For each $q \in \{0, 1, \ldots, s-1\}$, colour vertices $x_{rq+1,i}, x_{rq+2,i}, \ldots, x_{rq+r,i}$ with colour $(q+1)$. Colour the remaining $(k-s)(r-1)$ vertices of X_i with colours $s+1, s+2, \ldots, k$ such that each colour is assigned to exactly $(r-1)$ vertices. If $h > \ell$, give vertices $x_{\ell+1,i}$ and $x_{\ell+2,i}$ colour 1, and colour the remaining vertices $x_{\ell+3,i}, x_{\ell+4,i}, \ldots, x_{h,i}$ with colours 2 and 3, each used exactly $(r-1)$ times.

Let $i, j \in \{1, 2, \ldots, t\}$, $i < j$. The existence of a monochromatic $(2r)$-cycle in the decomposition $\mathcal{D}_\alpha$ of the $K_{\ell,\ell}$ between X_i and X_j would imply that $\mathcal{D}_\alpha$ contains a $(2r)$-cycle on the vertex set $\{x_{rq+1,i}, x_{rq+1,j}, x_{rq+2,i}, x_{rq+2,j}, \ldots, x_{rq+r,i}, x_{rq+r,j}\}$ for some $q \in \{0, 1, \ldots, s-1\}$, which would contradict that the permutation σ_α is acceptable.

If $h > \ell$, no $(2r)$-cycle of the $(2r)$-cycle decomposition of $K_{2r,\ell}$ between $S_i - X_i$ and X_j, the $(2r)$-cycle decomposition of $K_{2r,\ell}$ between $S_j - X_j$ and X_i, or the $(2r)$-cycle decomposition of $K_{2r,2r}$ between $S_i - X_i$ and $S_j - X_j$ is monochromatic, as no r vertices of $S_i - X_i$ have the same colour, and no r vertices of $S_j - X_j$ have the same colour.

It only remains to colour vertex ∞ and to show that, for each $i \in \{1, 2, \ldots, t\}$, no $(2r)$-cycle in the $(2r)$-cycle system on vertex set $S_i \cup \{\infty\}$ is monochromatic. We consider separately the cases $r = 3$ and $r \neq 3$. First, suppose $r = 3$. We note that $k \neq 3$ (since if $r = k = 3$, then $s = 0$). If $k = 4$, then $\ell = h = 12$ and $s = 4$. In this case, we give ∞ colour 1. For any $i \in \{1, \ldots, t\}$, $S_i \cup \{\infty\}$ has 4 vertices of colour 1 and 3 vertices of each of colours 2, 3 and 4. Hence no $2r = 6$ vertices of $S_i \cup \{\infty\}$ have the same colour, and so no $(2r)$-cycle in the $(2r)$-cycle system on $S_i \cup \{\infty\}$ can be monchromatic. If $r = 3$ and $k > 4$, we give ∞ colour k, so that for each $i \in \{1, \ldots, t\}$, $S_i \cup \{\infty\}$ has at most 5 vertices of each of colours 1, 2 and 3, and at most 4 vertices of any other colour. Again, no $2r = 6$ vertices of $S_i \cup \{\infty\}$ have the same colour.

Now, suppose $r > 3$, and give ∞ colour 1. There are at most $r + 3$ vertices of colour 1, at most $2r - 1$ vertices of colour 2, at most $2r - 1$ vertices of colour 3 and at most r vertices of any other colour. Again, no $2r$ vertices in $S_i \cup \{\infty\}$ have the same colour. □

Lemmas 2.3.8 and 2.3.9 show that the $(2r)$-cycle system S which we have constructed is indeed k-chromatic. We have thus completed the proof of Theorem 2.3.5.

We finish this section by proving an extension of Lemma 2.2.11 for m-cycle systems for which m is a multiple of 4. We first require the following result, due to Milici and Tuza.

Lemma 2.3.10. *[40] Let $m > 3$ be even. There exists a 2-chromatic m-cycle system of order $2m+1$ which admits a 2-colouring with colour classes of sizes m and $m+1$.*

Lemma 2.3.11. *Suppose $m \equiv 0 \pmod 4$. If there exists a k-chromatic m-cycle system of order n, then there exists a k-chromatic m-cycle system of order $n + 2m$.*

Proof. Let S denote a k-chromatic m-cycle system of order n on vertex set $\{y_1, y_2, \ldots, y_n\}$. We add $2m$ vertices $x_1, \ldots, x_{2m}$. We will decompose the copy of K_{2m+1} with vertex set $\{y_1, x_1, x_2, \ldots, x_{2m}\}$ and the copy of $K_{2m,n-1}$ between $\{x_1, \ldots x_{2m}\}$ and $\{y_2, \ldots, y_n\}$ into m-cycles; together with the m-cycles of S, the m-cycles in these decompositions will form an m-cycle system of order $n + 2m$.

First, using Lemma 2.3.10, we form an m-cycle system, T, on vertex set $\{y_1, x_1, x_2, \ldots, x_{2m}\}$ which admits a 2-colouring with colour classes $C_1 = \{y_1, x_1, x_3, \ldots, x_{2m-1}\}$ and $C_2 = \{x_2, x_4, \ldots, x_{2m}\}$. Next, we form the required m-cycle decomposition of $K_{2m,n-1}$ between $\{x_1, \ldots, x_{2m}\}$ and $\{y_2, \ldots, y_n\}$ as follows. For each $i \in \{0, 1, 2, 3\}$, we form an m-cycle decomposition of the $K_{\frac{m}{2},n-1}$ between $\{x_{\frac{im}{2}+1}, x_{\frac{im}{2}+2}, \ldots, x_{\frac{im}{2}+\frac{m}{2}}\}$ and $\{y_2, \ldots, y_n\}$ (such an m-cycle decomposition exists by Theorem 2.2.1); these four decompositions together form an m-cycle decomposition of the $K_{2m,n-1}$ between $\{x_1, \ldots, x_{2m}\}$ and $\{y_2, \ldots, y_n\}$. By creating these decompositions, we have now constructed an m-cycle system S' of order $n + 2m$.

Noting that S' contains the k-chromatic system S, it is clear that S' has chromatic number at least k. To complete the proof that S' is indeed k-chromatic, we will assign a k-colouring, as follows. Colour the vertices $y_1, y_2, \ldots, y_n$ with k colours such that y_1 has colour 1 and no m-cycle in S is monochromatic. For $i \in \{1, \ldots, 2m\}$, colour x_i with colour 1 if i is odd, and with colour 2 if i is even. Then no m-cycle in T is monochromatic. Any other m-cycle in S' occurs the m-cycle decomposition between

$\{x_{\frac{im}{2}+1}, x_{\frac{im}{2}+2}, \ldots, x_{\frac{im}{2}+\frac{m}{2}}\}$ and $\{y_2, \ldots, y_n\}$ for some $i \in \{0, 1, 2, 3\}$; any such cycle contains two vertices x_{j_1} and x_{j_2} such that j_1 and j_2 have different parity, and thus cannot be monochromatic. $\square$

Chapter 3

Invariants and Enumeration

Two cycle systems $S_1 = (X_1, \mathcal{C}_1)$ and $S_2 = (X_2, \mathcal{C}_2)$ are said to be *isomorphic* if there is a bijection $f : X_1 \longrightarrow X_2$ such that f maps each cycle in $\mathcal{C}_1$ to a cycle in $\mathcal{C}_2$. An *invariant* for a cycle system S is a function I such that $I(S') = I(S)$ for any cycle system S' which is isomorphic to S. A simple example of an invariant for a cycle system is the number of cycles in the system; any m-cycle system of order n has $\frac{n(n-1)}{2m}$ cycles, so clearly any two isomorphic cycle systems have the same number of cycles. Another example of a cycle system invariant is its chromatic number, which is defined in Chapter 2. Any two isomorphic cycle systems have the same chromatic number; however, as the results of Chapter 2 show, there may exist m-cycle systems of order n which have different chromatic numbers.

Invariants for cycle systems can be useful tools for distinguishing nonisomorphic systems; if the value of an invariant evaluated on two cycle systems is different for each, they cannot be isomorphic. One measure of an invariant's effectiveness in this regard, proposed in [42], is its *sensitivity*, defined by the number of different values of an invariant over a sample of cycle systems divided by the total number of

nonisomorphic systems in the sample. Typically, the sample would be all pairwise nonisomorphic m-cycle systems of a given order n; we will assume this to be the case unless otherwise specified. An invariant's sensitivity is thus a rational number in the interval $(0, 1]$; the closer the value is to 1, the more effective the invariant. An invariant I that has the property that two m-cycle systems S_1 and S_2 of order n are isomorphic if and only if $I(S_1) = I(S_2)$ is called a *complete invariant*; such an invariant clearly has sensitivity 1. In this chapter, we review some known results regarding the number of pairwise nonisomorphic m-cycle systems of a given order, and discuss the use of invariants in distinguishing nonisomorphic cycle systems. We also present some new enumerative results.

3.1 Invariants for cycle systems

In this section, we describe some known invariants for cycle systems and introduce a new invariant. Many of these invariants will be revisited in Section 3.2 with respect to their use in enumeration.

3.1.1 Bicolour vectors and sum-bicolour sequences

Dejter, Rivera-Vega and Rosa [19] proposed several invariants for 2-factorizations and cycle systems of the complete graph, which they employed in the enumeration of the 4-cycle systems and 6-cycle systems of order 9. Among these are bicolour vectors and sum-bicolour sequences, which we describe here. Let $(X, \mathcal{C})$ be an m-cycle system of order n, and let $\mathcal{C} = \{C_1, C_2, \ldots, C_{n(n-1)/(2m)}\}$. For $i \in \{3, 4, \ldots, n\}$ and $a, b \in \{1, 2, \ldots, \frac{n(n-1)}{2m}\}$ such that $a \neq b$, let $\alpha[a, b]_i$ be the number of i-cycles in K_n which have exactly one edge in C_a and $i - 1$ edges in C_b.

Based on this concept, several invariants can be formed. First, if $i \in \{1, 2, \ldots, n\}$, then for each $j \in \{0, 1, \ldots, m\}$, let $t_{i,j}$ denote the number of ordered pairs of intgers (a, b) for which $\alpha[a, b]_i = j$. The vector $T_i = (t_{i,0}, t_{i,1}, \ldots, t_{i,m})$, called the *bicolour vector of rank i* for the cycle system S, is an invariant for S.

Consider any 4-cycle system $S = (X, \mathcal{C})$ of order n, where $\mathcal{C} = \{C_1, \ldots, C_{n(n-1)/8}\}$. A 3-cycle C in K_n is of type $\alpha[a, b]_3$ if and only if two of its edges are in C_b and the other, in C_a, is a diagonal of C_b; since the two diagonals of C_b cannot both be contained in the same 4-cycle in $\mathcal{C}$, it follows that $t_{3,2} = \frac{n(n-1)}{4}$ and $t_{3,j} = 0$ if $j \in \{1, 3, 4\}$. Thus any two 4-cycle systems of order n have the same bicolour vector of rank 3. Furthermore, any two 4-cycle systems have the same bicolour vector of rank $i \geq 4$, as $\alpha[a, b]_i = 0$ for any ordered pair (a, b). Thus, the bicolour vector is ineffective in distinguishing nonisomorphic 4-cycle systems. Nevertheless, it can have nonzero sensitivity for m-cycle systems where $m > 4$ (see [19] for a discussion of this invariant with respect to 6-cycle systems of order 9 and 9-cycle systems of order 9).

Rather than the bicolour vector, we will employ in Section 3.2 a related invariant for cycle systems, also proposed in [19]. For each $i \in \{1, 2, \ldots, n\}$ and $a \in \{1, 2, \ldots, \frac{n(n-1)}{2m}\}$, let $s_{a,i} = \sum_{x \in \{1,2,\ldots,\frac{n(n-1)}{2m}\}, x \neq a} \alpha[a, x]_i$. For a fixed $i \in \{1, 2, \ldots, n\}$, the *sum-bicolour sequence of rank i* for the cycle system S is defined as the sequence S_i of numbers $s_{a,i}$, $a \in \{1, 2, \ldots, \frac{n(n-1)}{2m}\}$, arranged in nondecreasing order, and is also an invariant for S.

3.1.2 Neighbourhood graphs and related invariants

In their investigation of the 4-cycle systems of order 9, Dejter, Rivera-Vega and Rosa [19] employed an invariant based on the neighbourhood graphs of vertices. In this section we describe this invariant and then propose a relaxation of it, which is

generalized to an invariant for even cycle systems.

First, let S be a 4-cycle system of order n, and let a be any of its vertices. Any cycle (a, b, c, d) of S which contains a induces a 2-path through vertices b, c and d, as illustrated in Figure 3.1.

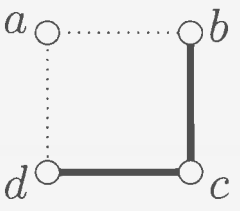

Figure 3.1: Induced path for vertex a in cycle (a, b, c, d)

For a given vertex a, there are $\frac{n-1}{2}$ such induced 2-paths. Taking the union of these $\frac{n-1}{2}$ paths yields a subgraph of K_n, called the *neighbourhood graph* of a in S, and denoted $N(a, S)$. Supposing there are t possible nonisomorphic neighbourhood graphs, $N_1, N_2, \ldots, N_t$, let n_i denote the number of vertices of S whose neighbourhood graph is isomorphic to N_i. The vector $(n_1, n_2, \ldots, n_t)$ is an invariant for the cycle system S.

Dejter, Rivera-Vega and Rosa [19] found that for $n = 9$, there are exactly three nonisomorphic neighbourhood graphs, N_1, N_2 and N_3, which are shown in Figure 3.2.

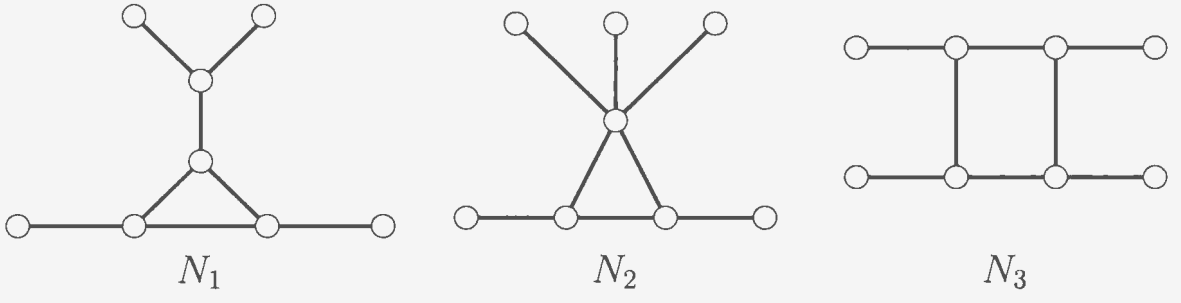

Figure 3.2: Possible neighbourhood graphs in a 4-cycle system of order 9

Example 3.1.1. *Consider the 4-cycle system S of order 9 with 4-cycles* $(1,2,3,4)$, $(1,3,6,5)$, $(1,6,2,7)$, $(1,8,2,9)$, $(2,4,7,5)$, $(3,5,8,7)$, $(3,8,6,9)$, $(4,5,9,8)$ *and* $(4,6,7,9)$. *If* $a \in \{3,4,5,7\}$, $N(a,S) = N_1$, *if* $a \in \{1,2,8,9\}$, $N(a,S) = N_2$ *and* $N(6,S) = N_3$; *thus S has neighbourhood graph invariant* $(4,4,1)$.

The neighbourhood graph invariant was used in [19] as a means of distinguishing nonisomorphic 4-cycle systems of order 9. There are eight such systems, and this invariant has sensitivity $\frac{7}{8}$.

To successfully use the neighbourhood graph invariant on 4-cycle systems of order $n > 9$ would require that the possible nonisomorphic neighbourhood graphs of vertices in a 4-cycle system of order n be determined. Rather than undertaking this task, we instead focus on the degree of each vertex in the neighbourhood graphs. Consider a 4-cycle system S of order $n \geq 9$, and let a be any vertex in S. Any vertex $b \neq a$ is adjacent to a in exactly one 4-cycle of S; the edges of this cycle (call it (a,b,c,d)) contribute 1 to the degree of b in $N(a,S)$, as the edge $\{b,c\}$ is in $N(a,S)$, while $\{a,b\}$ is not. Now, consider the cycles containing b other than (a,b,c,d). Any 4-cycle of S containing b but not a does not affect $N(a,S)$; any 4-cycle of S containing both a and b, but in which a and b are not adjacent, has b diagonally opposite to a, and contributes 2 to the degree of b in $N(a,S)$. It follows that the degree of b in $N(a,S)$ corresponds with the number of times b is diagonally opposite a in a 4-cycle of S; more specifically, b is diagonally opposite a in a 4-cycle of S q times if and only if b has degree $2q+1$ in $N(a,S)$.

We thus propose the following invariant for even cycle systems, which we will refer to as the *cycle diagonal invariant.* Let $S = (X, \mathcal{C})$ be a $(2r)$-cycle system of order n. For each vertex $a \in X$, form a vector $V_a = (v_1, v_2, \ldots, v_{\frac{n-1}{2}})$, where v_i is the number of vertices which are diagonally opposite a in exactly i of the

$(2r)$-cycles in $\mathcal{C}$. Note that any vector that we obtain in this manner has entries satisfying $v_1 + 2v_2 + \cdots + \frac{n-1}{2}v_{\frac{n-1}{2}} = \frac{n-1}{2}$, as a occurs in $\frac{n-1}{2}$ cycles in $\mathcal{C}$ and each vertex diagonally opposite a exactly i times accounts for i of these cycles. We can thus find a superset of the set of vectors $\{V_a : a \in X\}$ by solving the equation $x_1 + 2x_2 + \cdots + \frac{n-1}{2}x_{\frac{n-1}{2}} = \frac{n-1}{2}$, where x_j is a nonnegative integer for each $j \in \{1, 2, \ldots, \frac{n-1}{2}\}$; this is just an integer partitioning problem. Supposing there are t solution vectors, $W_1, W_2, \ldots, W_t$, we form an invariant vector $I = (i_1, i_2, \ldots, i_t)$, where i_j is the number of vertices a such that $V_a = W_j$.

We expect that this invariant will have somewhat lower sensitivity among 4-cycle systems than the neighbourhood graph invariant. As previously alluded to, for a 4-cycle system S and vertex a, the vector V_a corresponds with the degree sequence of the neighbourhood graph $N(a, S)$, yet among the 4-cycle systems of order 9, two of the three nonisomorphic neighbourhood graphs have the same degree sequence. We found among the eight pairwise nonisomorphic 4-cycle systems of order 9 that the cycle diagonal invariant has sensitivity 0.5, compared with 0.875 for the neighbourhood graph invariant. However, in practice, computation of the cycle diagonal invariant for 4-cycle systems of order $n > 9$ involves the solution of an integer partitioning problem and does not require computing in advance the possible pairwise nonisomorphic neighbourhood graphs. It is this property which makes the cycle diagonal invariant attractive, as the solution of the integer partitioning problem is easily computable.

Example 3.1.2. *Consider a 4-cycle system $(X, \mathcal{C})$ of order 9. The nonnegative integer solutions to the equation $x_1 + 2x_2 + 3x_3 + 4x_4 = 4$ are $W_1 = (4, 0, 0, 0)$, $W_2 = (2, 1, 0, 0)$, $W_3 = (1, 0, 1, 0)$, $W_4 = (0, 2, 0, 0)$ and $W_5 = (0, 0, 0, 1)$. Of these five vectors, we know that only two of them could be obtained as a vector V_a for some*

$a \in X$, as there are only two distinct neighbourhood graph degree sequences; these two are W_1 (which corresponds to the degree sequence of N_1 and N_3) and W_2 (which corresponds with the degree sequence of N_2).

The 4-cycle system of order 9 with 4-cycles $(1,2,3,4)$, $(1,3,6,5)$, $(1,6,2,7)$, $(1,8,2,9)$, $(2,4,7,5)$, $(3,5,8,7)$, $(3,8,6,9)$, $(4,5,9,8)$ *and* $(4,6,7,9)$ *has cycle diagonal invariant* $(5,4,0,0,0)$, *as* $V_3 = V_4 = V_5 = V_6 = V_7 = W_1$ *and* $V_1 = V_2 = V_8 = V_9 = W_2$.

3.1.3 Cycle structure

The cycle structure invariant was proposed for Steiner triple systems by Cole [14] and Cummings [16], and has since been employed by other authors to distinguish nonisomorphic triple systems [12, 29, 42]. The cycle structure invariant has been generalized for the Steiner system $S(t, t+1, n)$ (see [13]); we present a generalization of this invariant for cycle systems.

Let $(X, \mathcal{C})$ be an m-cycle system of order n, where $m \geq 3$. For each $x \in X$, let G_x be the graph induced by the set of edges $\{y, z\}$ such that x is adjacent to both y and z in some cycle $C \in \mathcal{C}$. Since every element of $X - \{x\}$ is adjacent to x exactly once in the cycles of $\mathcal{C}$, the edges of G_x form a 1-factor of K_{n-1} defined on vertex set $X - \{x\}$.

Given $x, y \in X$, we identify the vertex y in G_x (respectively x in G_y) with a new element ∞. The edges of G_x and G_y now form two 1-factors of K_{n-1} on vertex set $(X - \{x, y\}) \cup \{\infty\}$. Taking their union, and allowing multiple edges, we obtain a graph $G_{x,y}$ on vertex set $(X - \{x, y\}) \cup \{\infty\}$. Every vertex in $G_{x,y}$ has degree 2, and the connected components of $G_{x,y}$ are all even-length cycles of length at least 2, where we view two vertices joined by a pair of edges as a cycle of length 2.

Recording the length of each cycle in $G_{x,y}$, we form a vector $V_{x,y}$, called a *cycle list*, consisting of the cycle lengths in $G_{x,y}$ in nondecreasing order (where i occurs in $V_{x,y}$ exactly the number of times $G_{x,y}$ contains an i-cycle). Each such vector $V_{x,y}$ corresponds with a partition of the integer $n-1$ into nonnegative even integers. We form an invariant vector, each coordinate of which corresponds with exactly one of the vectors which is obtained in this manner. The value of the V-coordinate of the invariant vector is the number of times V is obtained as a vector $V_{x,y}$ for some pair $\{x,y\} \subseteq X$.

As an invariant for triple systems, Gibbons [26] suggested the *fragment vector*, whereby only 4-cycles in each cycle list are considered. This invariant, which reduces the space required in computation, is a complete invariant for Steiner triple systems of order at most 15 [26, 48].

3.1.4 Automorphism group order

Let $S = (X, \mathcal{C})$ be an m-cycle system of order n. An isomorphism $\phi : X \longrightarrow X$ which maps each cycle of $\mathcal{C}$ to a cycle of $\mathcal{C}$ is called an *automorphism*. The set of automorphisms of S, with the operation of set composition, forms a group called the *automorphism group* of S, denoted $\mathrm{Aut}(S)$. The order of this group is a well-known invariant for S.

3.2 Enumeration of cycle systems of small order

Let $N_m(n)$ denote the number of pairwise nonisomorphic m-cycle systems of order n. Known nontrivial values of $N_m(n)$ are summarized in Table 3.1. In this section, we describe the algorithm that we have employed to generate pairwise nonisomorphic

m	n	$N_m(n)$	Reference
3	7	1	
	9	1	
	13	2	[53, 18, 6, 14]
	15	80	[15, 52, 29]
	19	11,084,874,829	[32]
4	9	8	[19]
	17	$\geq$ 22,727,480	Section 3.2.2
5	11	$\geq$ 12,482,276	Section 3.2.3
6	9	640	[19]
	13	$\geq$ 27,834,268	Section 3.2.4
7	7	2	[9]
9	9	122	[9]
11	11	22,691,203	Section 3.2.5

Table 3.1: Number of nonisomorphic m-cycle systems of order n

cycle systems, and report on our results. In particular, we have used this algorithm to determine the value of $N_{11}(11)$, which we discuss in Section 3.2.5.

3.2.1 A method for enumeration

We employ a backtracking algorithm in the generation of nonisomorphic cycle systems. We exhaustively search for m-cycle systems of a given order containing the cycle $(1, 2, \ldots, m)$, pruning the search tree by eliminating partial m-cycle systems of order n isomorphic to any partial system that has been previously encountered. Once we have found a complete cycle system, we compute its invariant. (The invariant used may be a single invariant or a compound invariant created by combining two or more invariants, and is dependent on the value of m.) If the value computed for the system's invariant has not previously been encountered, we know that the cycle system is not isomorphic to any found previously. Otherwise, we test among the cycle systems which have been already found to have the same

invariant to determine if the newly found system is isomorphic to any of them; if so, we discard the system, while if not, it is stored. A program used to implement this search heuristic for Hamilton cycle systems may be found in Appendix 1.

To test isomorphism of cycle systems with the same invariant, we employ a related graph, inspired by the incidence graph of a design, which is described in [4, Section 5]. If $D = (V, \mathcal{B})$ is a balanced incomplete block design, its incidence graph $\mathrm{In}(D)$ has vertex set $V \cup \mathcal{B}$ and edge set $\{\{x, B\} : x \in V, B \in \mathcal{B}, x \in B\}$. Two designs D_1 and D_2 are isomorphic if and only if their respective incidence graphs are isomorphic [37]. If we were to define an incidence graph for a cycle system $S = (X, \mathcal{C})$ in a similar manner, that is, a graph with vertex set $X \cup \mathcal{C}$ and edge set $\{\{x, C\} : x \in X, C \in \mathcal{C}, x \in V(C)\}$, the same property would not necessarily hold. (It is easy to see, for example, that any two n-cycle systems of order n would have the same incidence graph.) We thus modify the definition for cycle systems, producing a graph which we will refer to as a *cycle incidence graph.*

Let $S = (X, \mathcal{C})$ be an m-cycle system of order n, where $X = \{x_1, \ldots, x_n\}$ and $\mathcal{C} = \{C_1, \ldots, C_{\frac{n(n-1)}{2m}}\}$, and for each $i \in \left\{1, 2, \ldots, \frac{n(n-1)}{2m}\right\}$, suppose $C_i = (x_{1,i}, x_{2,i}, \ldots, x_{m,i})$. We define the cycle incidence graph $\mathrm{CI}(S)$ as follows. Let $V(\mathrm{CI}(S)) = \left\{v_1, v_2, \ldots, v_{n+\frac{n(n-1)}{2}}\right\}$. Vertices $v_1, \ldots, v_n$ will correspond with the vertices of S, so that $x_i = v_i$ for each $i \in \{1, \ldots, n\}$, while vertices $v_{n+1}, \ldots, v_{n+\frac{n(n-1)}{2}}$ will correspond with the cycles of S, with m vertices for each cycle. We define the edges of $\mathrm{CI}(S)$ as follows. For each $j \in \left\{0, 1, \ldots, \frac{n(n-1)}{2m} - 1\right\}$, let v_{n+jm+i} be adjacent to $v_{n+jm+(i+1)}$ for each $i \in \{1, 2, \ldots, m-1\}$, and let v_{n+jm+m} be adjacent to v_{n+jm+1}. If $i \in \{1, 2, \ldots, n\}$ and $j \in \left\{0, 1, \ldots, \frac{n(n-1)}{2m} - 1\right\}$, $k \in \{1, 2, \ldots, m\}$ then let v_i be adjacent to v_{n+jm+k} if and only if $x_{k,j+1} = x_i$. Observe that in $\mathrm{CI}(S)$, $\deg(v_i) = \frac{n-1}{2}$ if $i \in \{1, \ldots, n\}$ (since x_i occurs in exactly $\frac{n-1}{2}$ cycles in $\mathcal{C}$), and $\deg(v_i) = 3$ if $i \in \{n+1, \ldots, n+\frac{n(n-1)}{2}\}$.

An example of the cycle incidence graph for a cycle system is illustrated in Figure 3.3, which shows the cycle incidence graph of a 3-cycle system of order 7.

Let σ be a permutation of $\left\{1, 2, \ldots, \frac{n(n-1)}{2m}\right\}$, and consider the effects of forming the graph defined above based on the new ordering $C_{\sigma(1)}, C_{\sigma(2)}, \ldots, C_{\sigma(\frac{n(n-1)}{2m})}$. The graph obtained is clearly isomorphic to that defined above; that is, the order of cycles $C_1, C_2, \ldots, C_{\frac{n(n-1)}{2m}}$ is irrelevant. Furthermore, for any cycle C_i, writing the cycle with a different starting vertex or in the opposite orientation produces an isomorphic graph. Thus, for a given cycle system S, the corresponding graph $\mathrm{CI}(S)$ is well-defined.

Theorem 3.2.1. *Let $S_1 = (X, \mathcal{C})$ and $S_2 = (Y, \mathcal{D})$ be m-cycle systems of order n. Then $S_1 \cong S_2$ if and only if $\mathrm{CI}(S_1) \cong \mathrm{CI}(S_2)$.*

Proof. Suppose $X = \{x_1, \ldots, x_n\}$ and $\mathcal{C} = \{C_1, \ldots, C_{\frac{n(n-1)}{2m}}\}$, where $C_i = (x_{1,i}, x_{2,i}, \ldots, x_{m,i})$ for each $i \in \{1, 2, \ldots, \frac{n(n-1)}{2m}\}$. Let $Y = \{y_1, \ldots, y_n\}$, and let $V(\mathrm{CI}(S_1)) = \{v_1, \ldots, v_{n+\frac{n(n-1)}{2}}\}$ and $V(\mathrm{CI}(S_2)) = \{w_1, \ldots, w_{n+\frac{n(n-1)}{2}}\}$.

($\Longrightarrow$) Suppose the cycle systems S_1 and S_2 are isomorphic. It is not difficult to observe that $\mathrm{CI}(S_1) \cong \mathrm{CI}(S_2)$.

($\Longleftarrow$) Let $\psi : V(\mathrm{CI}(S_1)) \longrightarrow V(\mathrm{CI}(S_2))$ be a graph isomorphism. Recall that each of the n vertices $v_1, v_2, \ldots, v_n$ of $\mathrm{CI}(S_1)$ (respectively, $w_1, w_2, \ldots, w_n$ of $\mathrm{CI}(S_2)$) has degree $\frac{n-1}{2}$, and each other vertex has degree 3. We note that if $n = 7$, then there exists an m-cycle system of order 7 if and only if $m \in \{3, 7\}$. If S_1 and S_2 are 3-cycle systems of order 7, then $S_1 \cong S_2$, since any two 3-cycle systems of order 7 are isomorphic. Considering the case $m = n = 7$, a computer program employing *nauty* [37] verified that the two nonisomorphic 7-cycle systems of order 7 have nonisomorphic cycle incidence graphs, and so if $\mathrm{CI}(S_1) \cong \mathrm{CI}(S_2)$, then $S_1 \cong S_2$.

We now suppose $n \neq 7$, so that $\frac{n-1}{2} \neq 3$. It follows that ψ maps $\{v_1, v_2, \ldots, v_n\}$ onto $\{w_1, w_2, \ldots, w_n\}$ and $\{v_{n+1}, v_{n+2}, \ldots, v_{n+\frac{n(n-1)}{2}}\}$ onto $\{w_{n+1}, w_{n+2}, \ldots, w_{n+\frac{n(n-1)}{2}}\}$.

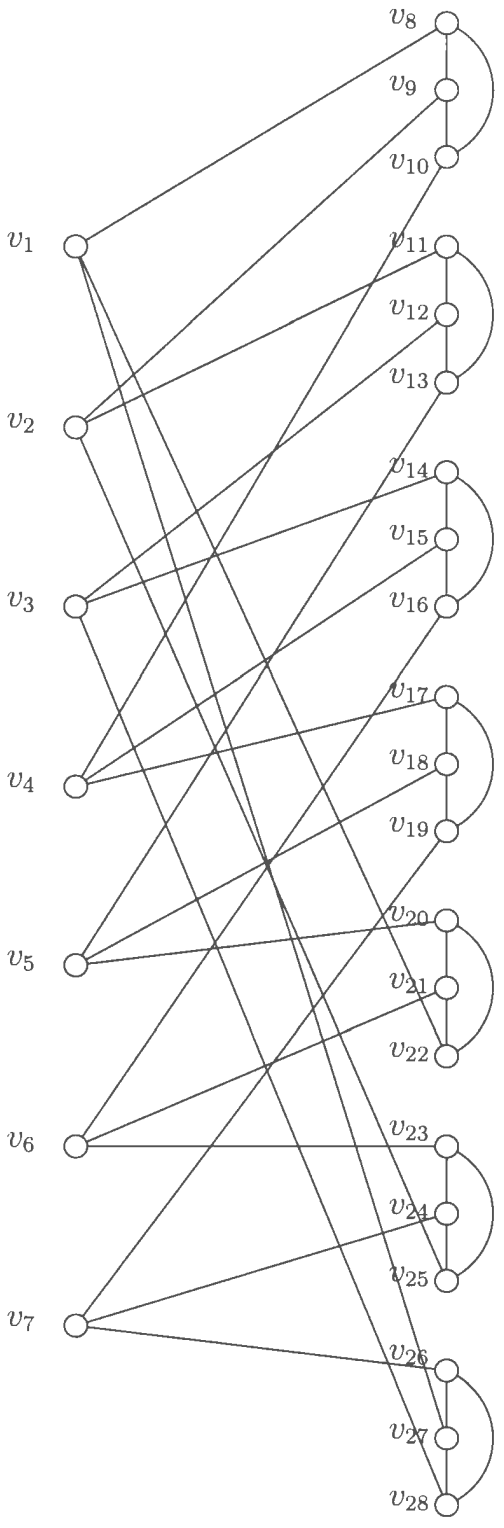

Figure 3.3: Cycle incidence graph for the 3-cycle system of order 7 with cycles (x_1, x_2, x_4), (x_2, x_3, x_5), (x_3, x_4, x_6), (x_4, x_5, x_7), (x_5, x_6, x_1), (x_6, x_7, x_2), (x_7, x_1, x_3)

Consider the function $\phi : X \longrightarrow Y$ defined by $\phi(x_i) = y_j$, where $w_j = \psi(v_i)$, for each $i \in \{1, 2, \ldots, n\}$. Observing that ϕ is the restriction of ψ to $\{v_1, \ldots, v_n\}$, it is clear that ϕ is a bijection between the sets X and Y. It remains to be shown that ϕ maps any m-cycle in $\mathcal{C}$ to an m-cycle in $\mathcal{D}$. Let $C_r = (x_{1,r}, \ldots, x_{m,r}) \in \mathcal{C}$, where $r \in \{1, \ldots, \frac{n(n-1)}{2m}\}$. Let $x_{t_i} = x_{i,r}$ for each $i \in \{1, \ldots, m\}$, and hence $v_{t_i} = x_{i,r}$ for each $i \in \{1, \ldots, m\}$. There exists $j \in \{0, 1, \ldots, \frac{n(n-1)}{2m} - 1\}$ such that for each $i \in \{1, \ldots, m\}$, v_{t_i} is adjacent to v_{n+jm+i}. Since ψ is an isomorphism, the m-cycle $(v_{n+jm+1}, v_{n+jm+2}, \ldots, v_{n+jm+m})$ is mapped by ψ to an m-cycle of $\mathrm{CI}(S_2)$. Since $\psi(v_{n+jm+i}) \in \{w_{n+1}, w_{n+2}, \ldots, w_{n+\frac{n(n-1)}{2}}\}$ for each $i \in \{1, 2, \ldots, m\}$, it follows that the m-cycle $(v_{n+jm+1}, v_{n+jm+2}, \ldots, v_{n+jm+m})$ is mapped under ψ to an m-cycle $(w_{n+km+1}, w_{n+km+2}, \ldots, w_{n+km+m})$ in $\mathrm{CI}(S_2)$ for some $k \in \{0, 1, \ldots, \frac{n(n-1)}{2m} - 1\}$. Since v_{t_i} is adjacent in $\mathrm{CI}(S_1)$ to v_{n+jm+i} for each $i \in \{1, 2, \ldots, m\}$, it follows that $\psi(v_{t_i})$ is adjacent to $\psi(v_{n+jm+i})$ for each $i \in \{1, 2, \ldots, m\}$. Because $(\psi(v_{n+jm+1}), \psi(v_{n+jm+2}), \ldots, \psi(v_{n+jm+m})) = (w_{n+km+1}, w_{n+km+2}, \ldots, w_{n+km+m})$, it follows that $(\psi(v_{t_1}), \psi(v_{t_2}), \ldots, \psi(v_{t_m}))$ is an m-cycle in $\mathcal{D}$, in other words, $(\phi(x_{t_1}), \phi(x_{t_2}), \ldots, \phi(x_{t_m})) \in \mathcal{D}$. □

To determine if two cycle systems are isomorphic, we determine if their respective cycle incidence graphs are isomorphic, by making use of the *nauty* software package [37], which calculates the automorphism group of a vertex-coloured graph; nauty is also able to determine a canonical labelling of a graph, which may be used for isomorphism testing. To determine isomorphism of partial triple systems, we build a partial cycle incidence graph, colouring the vertices corresponding to the vertices of the cycle system with one colour, and those corresponding to the cycles another colour; the coloured graphs are compared for isomorphism.

In addition to their use in determining whether two cycle systems are isomorphic,

cycle incidence graphs can also be used to determine properties of the automorphism groups of the cycle systems from which they are formed. So that we can explore the connection between the automorphism groups of cycle systems and their cycle incidence graphs, we first require some preliminary definitions.

Let $S = (X, \mathcal{C})$ be an m-cycle system of order n. Suppose $X = \{x_1, \ldots, x_n\}$ and $\mathcal{C} = \{C_1, \ldots, C_{\frac{n(n-1)}{2m}}\}$, where for each $i \in \{1, \ldots, \frac{n(n-1)}{2m}\}$, $C_i = (x_{1,i}, x_{2,i}, \ldots, x_{m,i})$. Let $G = \mathrm{CI}(S)$ denote the cycle incidence graph of S. Consider an automorphism ϕ of S. For each $i \in \{1, \ldots, \frac{n(n-1)}{2m}\}$, let $C_{\phi_i} = (x_{1,\phi_i}, \ldots, x_{m,\phi_i})$ be the cycle in $\mathcal{C}$ to which ϕ maps C_i. Note that if $i \neq j$, then $\phi_i \neq \phi_j$. If $i \in \{1, \ldots, \frac{n(n-1)}{2m}\}$, then let $\sigma_{\phi,i}$ be the permutation of $\{1, \ldots, m\}$ such that if $\phi(x_{p,i}) = x_{q,\phi_i}$, then $\sigma_{\phi,i}(p) = q$.

Given an automorphism ϕ of S, we now define a function

$$\psi_\phi : \{v_1, \ldots, v_{n+\frac{n(n-1)}{2}}\} \longrightarrow \{v_1, \ldots, v_{n+\frac{n(n-1)}{2}}\}$$

as follows. First, for each $i \in \{1, \ldots, n\}$, let $\psi_\phi(v_i) = v_j$, where $x_j = \phi(x_i)$. Next, let $i \in \{n+1, \ldots, n+\frac{n(n-1)}{2}\}$, and write $i = n+rm+s$ where $r \in \{0, \ldots, \frac{n(n-1)}{2m}-1\}$, $s \in \{1, \ldots, m\}$ (so v_i corresponds to $x_{s,r+1}$, recalling that $C_{r+1} = (x_{1,r+1}, x_{2,r+1}, \ldots, x_{m,r+1})$). Let $\psi_\phi(v_i) = v_{n+(\phi_{r+1}-1)m+\sigma_{\phi,r+1}(s)}$. Note that ψ_ϕ maps $\{v_1, \ldots, v_n\}$ onto $\{v_1, \ldots, v_n\}$ and $\{v_{n+1}, \ldots, v_{n+\frac{n(n-1)}{2}}\}$ onto $\{v_{n+1}, \ldots, v_{n+\frac{n(n-1)}{2}}\}$. Furthermore, note that the restriction of ψ_ϕ to $\{v_1, \ldots, v_n\}$ is just the function ϕ.

Lemma 3.2.2. *The function ψ_ϕ is an automorphism of G.*

Proof. Suppose that $\psi_\phi(v_i) = \psi_\phi(v_j)$ for some $i, j \in \{1, \ldots, n + \frac{n(n-1)}{2}\}$. Then either i and j are both in $\{1, \ldots, n\}$ or i and j are both in $\{n+1, \ldots, n + \frac{n(n-1)}{2}\}$. If $i, j \in \{1, \ldots, n\}$, then $\phi(x_i) = \phi(x_j)$ which implies $x_i = x_j$, and so $v_i = v_j$. Next, suppose $i, j \in \{n+1, \ldots, n + \frac{n(n-1)}{2}\}$, and write $i = n + rm + s$ and $j = n + tm + u$, where $r, t \in \{0, \ldots, \frac{n(n-1)}{2m} - 1\}$, $s, u \in \{1, \ldots, m\}$. So $\psi_\phi(v_i) = \psi_\phi(v_j)$ means that

$v_{n+(\phi_{r+1}-1)m+\sigma_{\phi,r+1}(s)} = v_{n+(\phi_{t+1}-1)m+\sigma_{\phi,t+1}(u)}$ and hence $n+(\phi_{r+1}-1)m+\sigma_{\phi,r+1}(s) = n+(\phi_{t+1}-1)m+\sigma_{\phi,t+1}(u)$. But since $\sigma_{\phi,r+1}(s), \sigma_{\phi,t+1}(u) \in \{1,\ldots,m\}$, it must be that $\sigma_{\phi,r+1}(s) = \sigma_{\phi,t+1}(u)$ and also $\phi_{r+1} = \phi_{t+1}$. Recalling that if $i \neq j$, then $\phi_i \neq \phi_j$, we see that $r+1 = t+1$, i.e. $r = t$. Now, $\sigma_{\phi,r+1}(s) = \sigma_{\phi,t+1}(u) = \sigma_{\phi,r+1}(u)$, so that $s = u$. Thus, $n+rm+s = n+tm+u$, and so $v_i = v_j$. Hence, ψ_ϕ is injective.

To show that ψ_ϕ is surjective, let $i \in \{1,\ldots,n+\frac{n(n-1)}{2}\}$. If $i \in \{1,\ldots,n\}$, then there exists $j \in \{1,\ldots,n\}$ such that $x_i = \phi(x_j)$; for the same j, $v_i = \psi_\phi(v_j)$. Now suppose $i \in \{n+1,\ldots,n+\frac{n(n-1)}{2}\}$ and write $i = n+rm+s$ where $r \in \{0,\ldots,\frac{n(n-1)}{2m}-1\}$, $s \in \{1,\ldots,m\}$. Since ϕ is a cycle system automorphism, there exists $t \in \{1,\ldots,\frac{n(n-1)}{2m}\}$ such that ϕ maps C_t onto C_{r+1}. So $\phi_t = r+1$. Choose $u \in \{1,\ldots,m\}$ such that $\phi(x_{u,t}) = x_{s,r+1}$, so $\sigma_{\phi,t}(u) = s$. Then $v_i = v_{n+rm+s} = v_{n+(\phi_t-1)m+\sigma_{\phi,t}(u)} = \psi_\phi(v_{n+(t-1)m+u})$. Hence ψ_ϕ is surjective.

To show that ψ_ϕ is an automorphism, let $\{v_i, v_j\} \in E(G)$; it must be shown that $\{\psi_\phi(v_i), \psi_\phi(v_j)\} \in E(G)$. Note that i and j cannot both be in $\{1,\ldots,n\}$, so there are two cases to consider:

Case 1. i and j are not both in $\{n+1,\ldots,n+\frac{n(n-1)}{2}\}$.

Without loss of generality, assume that $i \in \{1,\ldots,n\}$ and $j \in \{n+1,\ldots,n+\frac{n(n-1)}{2}\}$, and write $j = n+rm+s$, where $r \in \{0,\ldots,\frac{n(n-1)}{2m}-1\}$ and $s \in \{1,\ldots,m\}$. Since v_i and v_j are adjacent in G, $x_i = x_{s,r+1}$, so that $\psi_\phi(v_i) = \phi(x_i) = \phi(x_{s,r+1}) = x_{\sigma_{\phi,r+1}(s),\phi_{r+1}}$. Observe also that $\psi_\phi(v_j) = v_{n+(\phi_{r+1}-1)m+\sigma_{\phi,r+1}(s)}$. Therefore, $\psi_\phi(v_i)$ is adjacent to $\psi_\phi(v_j)$.

Case 2. $i, j \in \{n+1, \ldots, n+\frac{n(n-1)}{2}\}$.

Write $i = n + rm + s$ and $j = n + rm + u$ where $r \in \{0, \ldots, \frac{n(n-1)}{2m} - 1\}$ and $s, u \in \{1, \ldots, m\}$. Since v_i and v_j are adjacent in G, it must be that $x_{s,r+1}$ and $x_{u,r+1}$ are adjacent in C_{r+1}. Since ϕ is a cycle system automorphism, $\phi(x_{s,r+1})$ and $\phi(x_{u,r+1})$ are adjacent in $C_{\phi_{r+1}}$, i.e. $\{x_{\sigma_{\phi,r+1}(s),\phi_{r+1}}, x_{\sigma_{\phi,r+1}(u),\phi_{r+1}}\} \in E(C_{\phi_{r+1}})$, which means that $v_{n+(\phi_{r+1}-1)m+\sigma_{\phi,r+1}(s)}$ and $v_{n+(\phi_{r+1}-1)m+\sigma_{\phi,r+1}(u)}$ must be adjacent in G, in other words, $\{\psi_\phi(v_i), \psi_\phi(v_j)\} \in E(G)$.

So, for any $i, j \in \{1, \ldots, n+\frac{n(n-1)}{2}\}$, $\{v_i, v_j\} \in E(G)$ implies that $\{\psi_\phi(v_i), \psi_\phi(v_j)\} \in E(G)$, and hence ψ_ϕ is an automorphism. □

Theorem 3.2.3. *Let $S = (X, \mathcal{C})$ be an m-cycle system of order $n \neq 7$. Then the automorphism group of S is isomorphic to the automorphism group of its cycle incidence graph* $\mathrm{CI}(S)$.

Proof. Let $X = \{x_1, \ldots, x_n\}$ and $\mathcal{C} = \{C_1, \ldots, C_{\frac{n(n-1)}{2m}}\}$, where, for each $i \in \{1, \ldots, \frac{n(n-1)}{2m}\}$, $C_i = (x_{1,i}, \ldots, x_{m,i})$. Also, let $G = \mathrm{CI}(S)$.

Define $f : \mathrm{Aut}(S) \longrightarrow \mathrm{Aut}(G)$ by $f(\phi) = \psi_\phi$. It will be shown that f is an isomorphism.

First, suppose that $f(\phi) = f(\rho)$ for some automorphisms ϕ and ρ of S, so that $\psi_\phi = \psi_\rho$. Since $\phi = \psi_\phi\big|_{\{v_1,\ldots,v_n\}}$ and $\rho = \psi_\rho\big|_{\{v_1,\ldots,v_n\}}$, it is clear that $\phi = \rho$. So f is injective.

Now, let $\psi \in \mathrm{Aut}(G)$, and define $\phi : \{x_1, \ldots, x_n\} \longrightarrow \{x_1, \ldots, x_n\}$ by the rule that for each $i \in \{1, \ldots, n\}$, $\phi(x_i) = x_j$ where $v_j = \psi(v_i)$. It will be shown that $\phi \in \mathrm{Aut}(S)$ and $\psi = f(\phi)$.

First, recalling that $\deg(v_i) = 3$ if $i \in \{1, \ldots, n\}$ and $\deg(v_i) = \frac{n-1}{2}$ if $i \in \{n+1, \ldots, n+\frac{n(n-1)}{2}\}$, note that ψ maps $\{v_1, \ldots, v_n\}$ onto $\{v_1, \ldots, v_n\}$; it follows

that ϕ maps $\{x_1, \ldots, x_n\}$ onto $\{x_1, \ldots, x_n\}$, i.e. ϕ is surjective. Also, ϕ is injective since ψ is injective.

Now, let $i \in \{1, \ldots, \frac{n(n-1)}{2m}\}$ and consider the m-cycle C_i. The corresponding m-cycle $(v_{n+(i-1)m+1}, v_{n+(i-1)m+2}, \ldots, v_{n+(i-1)m+m})$ in G is mapped by ψ to an m-cycle C in G. Since ψ maps $\{v_{n+1}, \ldots, v_{n+\frac{n(n-1)}{2}}\}$ onto $\{v_{n+1}, \ldots, v_{n+\frac{n(n-1)}{2}}\}$, C must be of the form $(v_{n+(j-1)m+1}, \ldots, v_{n+(j-1)m+m})$ for some $j \in \{1, \ldots, \frac{n(n-1)}{2m}\}$. It will be shown that ϕ maps the m-cycle C_i onto the m-cycle C_j. For each $k \in \{1, \ldots, m\}$, let $k_i \in \{1, \ldots, n\}$ such that $x_{k_i} = x_{k,i}$. In G, v_{k_i} is adjacent to $v_{n+(i-1)m+k}$ and $\psi(x_{k,i}) = \psi(v_{k_i})$ is adjacent to $\psi(v_{n+(i-1)m+k})$. Since $(\psi(v_{n+(i-1)m+1}), \ldots, \psi(v_{n+(i-1)m+m})) = (v_{n+(j-1)m+1}, \ldots, v_{n+(j-1)m+m})$, it must be that $(\phi(x_{1,i}), \ldots, \phi(x_{m,i})) = C_j$. So ϕ is an automorphism of S.

It remains to be shown that $\psi = f(\phi)$, i.e. that $\psi = \psi_\phi$. Clearly $\psi(v_i) = \psi_\phi(v_i)$ for each $i \in \{1, \ldots, n\}$, so now suppose $i \in \{n+1, \ldots, n+\frac{n(n-1)}{2}\}$, and write $i = n+rm+s$, where $r \in \{0, \ldots, \frac{n(n-1)}{2m} - 1\}$, $s \in \{1, \ldots, m\}$. For each $u \in \{1, \ldots, m\}$, v_{n+rm+u} is adjacent to $v_{u_{r+1}}$, where $x_{u_{r+1}} = x_{u,r+1}$, and so $\psi(v_{n+rm+u})$ is adjacent to $\psi(v_{u_{r+1}}) = \phi(x_{u,r+1}) = x_{\sigma_{\phi,r+1}(u),\phi_{r+1}}$. It must be that $\{\psi(v_{n+rm+1}), \ldots, \psi(v_{n+rm+m})\} = \{v_{n+(\phi_{r+1}-1)m+1}, \ldots, v_{n+(\phi_{r+1}-1)m+m}\}$, and so, since $\psi(v_i) = v_{n+(\phi_{r+1}-1)m+\sigma_{\phi,r+1}(s)}$ is adjacent to $v_{(\sigma_{\phi,r+1}(s))_{\phi_{r+1}}} = x_{\sigma_{\phi,r+1}(s),\phi_{r+1}}$, it follows that $\psi(v_i) = v_{n+(\phi_{r+1}-1)m+\sigma_{\phi_{r+1}}(s)} = \psi_\phi(v_i)$. Therefore, $\psi = \psi_\phi = f(\phi)$. Hence f is surjective.

Now, let ϕ and ρ be automorphisms of S. It must be shown that $f(\phi \circ \rho) = f(\phi) \circ f(\rho)$. Let $i \in \{1, \ldots, n\}$. Then $\big(f(\phi \circ \rho)\big)(v_i) = \psi_{\phi\circ\rho}(v_i) = \big(\phi \circ \rho\big)(x_i) = \big(\psi_\phi \circ \psi_\rho\big)(v_i) = \big(f(\phi) \circ f(\rho)\big)(v_i)$. Now, suppose $i \in \{n+1, \ldots, n+\frac{n(n-1)}{2}\}$ and write $i = n + rm + s$ where $r \in \{0, \ldots, \frac{n(n-1)}{2m} - 1\}$ and $s \in \{1, \ldots, m\}$. Then

$$\big(f(\phi \circ \rho)\big)(v_i) = v_{n+(\phi_{\rho_{r+1}}-1)m+\sigma_{\phi,\rho_{r+1}}(\sigma_{\rho,r+1}(s))} = \big(f(\phi) \circ f(\rho)\big)(v_i).$$

Hence f is a homomorphism, and so $\mathrm{Aut}(S) \cong \mathrm{Aut}(G)$. $\square$

The following sections detail the results of our investigation into the number of pairwise nonisomorphic cycle systems of small order. We have completed the enumeration of the 11-cycle systems of order 11. We have also generated large numbers of 4-cycle systems of order 17, 5-cycle systems of order 11 and 6-cycle svstems of order 13, although the enumeration for these cycle systems is not yet complete. We will report on the sensitivity of various invariants for the cycle systems that we have generated.

3.2.2 4-cycle systems

The eight 4-cycle systems of order 9 were first enumerated in [19]. In this paper, the neighbourhood graph invariant and automorphism group order were examined. On these eight systems, we investigated combinations of the following four invariants: the cycle diagonal invariant, the sum-bicolour vector S_3 of rank 3, the automorphism group size and the cycle structure. Of these four invariants employed individually, none are complete; the one with greatest sensitivity is the cycle structure, with sensitivity 0.75. When combined to form a compound invariant, S_3 together with either the automorphism group size or the cycle structure forms a complete invariant. Only one combination of three invariants fails to be complete, namely the cycle diagonal invariant together with the automorphism group size and the cycle structure. For details on the effectiveness of each combination of invariants, refer to Table 3.2.

Using the algorithm described in Section 3.2.1, we have generated 22,727,480 pairwise nonisomorphic 4-cycle systems of order 17; the enumeration of the 4-cycle systems of order 17 is not yet complete. Refer to Table 3.3 for information on the sensitivity of invariants (rounded to six decimal places) on these 22,727,480 systems.

Invariant				Sensitivity	Maximum number of systems per invariant value
Cycle diagonal invariant	S_3	Automorphism group order	Cycle structure invariant		
x				0.5	4
	x			0.5	3
		x		0.625	3
			x	0.75	2
x	x			0.625	2
x		x		0.875	2
x			x	0.75	2
	x	x		1	1
	x		x	1	1
		x	x	0.875	2
x	x	x		1	1
x	x		x	1	1
x		x	x	0.875	2
	x	x	x	1	1
x	x	x	x	1	1

Table 3.2: Sensitivity of invariants for 4-cycle systems of order 9

Invariant				Sensitivity	Maximum number of systems per invariant value
Cycle diagonal invariant	S_3	Automorphism group order	Cycle structure invariant		
x				0.001553	86,314
	x			0.000379	229,490
		x		0.000000	22,722,851
			x	0.984499	54
x	x	x	x	0.994913	8

Table 3.3: Sensitivity of invariants for 22,727,480 pairwise nonisomorphic 4-cycle systems of order 17

Invariant			Sensitivity	Maximum number of systems per invariant value
S_3	Cycle structure invariant	Automorphism group order		
x			0.000405	245,391
	x		0.002898	24,672
		x	0.000000	12,482,211
x	x	x	0.253192	599

Table 3.4: Sensitivity of invariants for 12,482,276 pairwise nonisomorphic 5-cycle systems of order 11

3.2.3 5-cycle systems

We have generated 12,482,276 pairwise nonisomorphic 5-cycle systems of order 11, although this enumeration is incomplete. Table 3.4 contains information on the sensitivity of invariants on these systems, rounded to six decimal places.

3.2.4 6-cycle systems

The 6-cycle systems of order 9, like the 4-cycle systems of order 9, were first enumerated in [19]; there are 640 pairwise nonisomorphic 6-cycle systems of order 9. In [19], bicolour vectors and sum-bicolour sequences for 6-cycle systems were considered, along with the automorphism group. We recalculated the automorphism group order[1] and sum-bicolour sequences S_3 and S_4, and considered them along with the cycle diagonal invariant and cycle structure. The effectiveness of various combinations of these invariants for 6-cycle systems of order 9 is summarized in

[1]In [19], three of the 6-cycle systems of order 9 with automorphism group order 1 were reported as having automorphism group order 2. These are: the system with 6-cycles $(1,2,3,4,5,6)$, $(1,3,5,2,7,4)$, $(1,5,8,2,4,9)$, $(1,7,6,3,9,8)$, $(2,6,8,3,7,9)$, $(4,6,9,5,7,8)$; the system with 6-cycles $(1,2,3,4,5,6)$, $(1,3,5,2,6,7)$, $(1,4,7,2,8,9)$, $(1,5,9,6,3,8)$, $(2,4,8,5,7,9)$, $(3,7,8,6,4,9)$; and the system with 6-cycles $(1,2,3,4,5,6)$, $(1,3,5,2,6,7)$, $(1,4,8,5,7,9)$, $(1,5,9,6,3,8)$, $(2,4,7,3,9,8)$, $(2,7,8,6,4,9)$.

Table 3.5. Of the five invariants applied alone, the cycle structure invariant, with sensitivity of approximately 34.5%, has the highest sensitivity. Combined with the sum-bicolour vector of rank 3, the sensitivity of cycle structure increases to over 88%, while the compound invariant consisting of cycle structure and both the sum-bicolour vectors of ranks 3 and 4 has sensitivity over 98%.

We have generated 27,834,268 pairwise nonisomorphic 6-cycle systems of order 13, but the enumeration is not yet complete.

3.2.5 11-cycle systems

Colbourn [9] studied Hamilton decompositions of the complete graph, and enumerated the 7-cycle systems of order 7 and 9-cycle systems of order 9. Although the complete enumeration of pairwise nonisomorphic 11-cycle systems of order 11 was not completed in [9], the number of pairwise nonisomorphic 11-cycle systems of order 11 with nontrivial automorphism group was shown to be 3,140, and the number of 11-cycle systems of order 11 with each of the four automorphism group orders greater than 1 was determined. Using the algorithm described in Section 3.2.1, we have determined that there are precisely 22,691,203 pairwise nonisomorphic 11-cycle systems of order 11. The computer program with which we implemented our search can be found in Appendix 1, and a complete list of these systems may be found online at `http://www.math.mun.ca/~burgess/11cs.html`.

Table 3.6 shows the sensitivity of combinations of six invariants (cycle structure, the sum-bicolour sequences of ranks 3, 4, 5 and 6, and the automorphism group order) on the 11-cycle systems of order 11, rounded to six decimal places. Of these, cycle structure is the most effective, with a sensitivity of approximately 0.005390, while the automorphism group order is the least. (As there

Invariants					Sensitivity	Maximum number of systems per invariant value
Cycle diagonal invariant	S_3	S_4	Automorphism group order	Cycle structure invariant		
x					0.0390625	155
	x				0.0828125	80
		x			0.0765625	110
			x		0.009375	594
				x	0.3453125	13
x	x				0.365625	9
x		x			0.296875	32
x			x		0.0640625	147
x				x	0.75625	6
	x	x			0.5015625	26
	x		x		0.10625	80
	x			x	0.884375	3
		x	x		0.0984375	106
		x		x	0.85	6
			x	x	0.3796875	13
x	x	x			0.7375	8
x	x		x		0.3921875	29
x	x			x	0.9640625	3
x		x	x		0.3203125	32
x		x		x	0.95	3
x			x	x	0.7703125	6
	x	x	x		0.515625	25
	x	x		x	0.9828125	2
	x		x	x	0.8890625	3
		x	x	x	0.85625	6
x	x	x	x		0.74375	8
x	x	x		x	0.990625	2
x	x		x	x	0.9640625	3
x		x	x	x	0.95	3
	x	x	x	x	0.9828125	2
x	x	x	x	x	0.990625	2

Table 3.5: Invariants for 6-cycle systems of order 9

are only five different automorphism group orders, and given the relatively small number of systems whose automorphism group is nontrivial, the inefficiency of the automorphism group order as an invariant for these systems is not unexpected.) Combining with any one of the sum-bicolour sequences increases the cycle structure's sensitivity to approximately 0.25, while a compound invariant formed from the cycle structure along with any two of the sum-bicolour sequences has sensitivity over 0.9. The compound invariant formed from cycle structure and each of the four sum-bicolour vectors has extremely high sensitivity (approximately 0.999979); adding the automorphism group order yields no improvement in this case.

Invariant						Sensitivity	Maximum number of systems per invariant value
Cycle structure	S_3	S_4	S_5	S_6	Automorphism group order		
x						0.005390	15,877
	x					0.000077	747,788
		x				0.000097	928,627
			x			0.000086	930,781
				x		0.000089	791,489
					x	0.000000	22,688,063
x	x					0.258462	596
x		x				0.241241	657
x			x			0.240530	641
x				x		0.256611	655
x					x	0.005473	15,877
	x	x				0.018661	32,863
	x		x			0.019870	36,084
	x			x		0.021666	31,315
	x				x	0.000083	747,788
		x	x			0.019669	42,349
		x		x		0.020840	36,740
		x			x	0.000129	928,593
			x	x		0.020081	36,696
			x		x	0.000116	930,747
				x	x	0.000116	791,456
x	x	x				0.917995	35
x	x		x			0.915508	33
x	x			x		0.921727	31
x	x				x	0.258505	596
x		x	x			0.909062	32
x		x		x		0.914863	35
x		x			x	0.241293	657
x			x	x		0.916101	33
x			x		x	0.240587	641
x				x	x	0.256675	655
	x	x	x			0.400021	1,882
	x	x		x		0.416152	1,847
	x	x			x	0.018759	32,863
	x		x	x		0.415646	1,883

Invariant						Sensitivity	Maximum number of systems per invariant value
Cycle structure	S_3	S_4	S_5	S_6	Automorphism group order		
	x		x		x	0.019967	36,084
	x			x	x	0.021770	31,315
		x	x	x		0.393614	2,016
		x	x		x	0.019780	42,348
		x		x	x	0.020960	36,740
			x	x	x	0.020200	36,696
x	x	x	x			0.998752	4
x	x	x		x		0.998819	5
x	x	x			x	0.917996	35
x	x		x	x		0.998786	5
x	x		x		x	0.915509	33
x	x			x	x	0.921728	31
x		x	x	x		0.998695	6
x		x	x		x	0.909064	32
x		x		x	x	0.914866	35
x			x	x	x	0.916103	33
	x	x	x	x		0.927703	240
	x	x	x		x	0.400048	1,882
	x	x		x	x	0.416180	1,847
	x		x	x	x	0.415678	1,883
		x	x	x	x	0.393649	2,016
x	x	x	x	x		0.999979	4
x	x	x	x		x	0.998752	4
x	x	x		x	x	0.998819	4
x	x		x	x	x	0.998786	5
x		x	x	x	x	0.998695	6
	x	x	x	x	x	0.927705	240
x	x	x	x	x	x	0.999979	4

Table 3.6: Invariants for 11-cycle systems of order 11

Chapter 4

Summary and Open Problems

This thesis has examined two aspects of cycle systems: colourings of cycle systems and invariants for cycle systems. We now summarize the main results presented in this thesis and state some open problems which arise from them.

In Chapter 2, we explored vertex colourings of cycle systems in which no cycle is monochromatic. Our focus in this chapter was on even cycle systems. Most of the previous work on weak colouring of cycle systems has focussed on 3-cycle systems. The best known result prior to the writing of this thesis on the existence of m-cycle systems with chromatic number $k > 2$, where $m > 3$, was that there exists a non-2-chromatic m-cycle system for any integer $m > 3$ [39]. The systems considered in [39] had large orders, and the exact value of their chromatic numbers was not determined. In the case of 4-cycle systems, we have constructed a 3-chromatic 4-cycle system of order 49; this result dramatically lowers the order of a known 4-cycle system with chromatic number greater than 2, as each non-2-chromatic 4-cycle system constructed in [39] has order at least $1+8\cdot 257^{28}$. Our major result regarding 4-cycle systems is an analogue of a theorem of de Brandes, Phelps and Rödl [17] for 3-cycle systems; we have proven that for any integer $k \geq 2$, there is an integer w_k such that for any admissible $n \geq n_4(k)$, there is a k-chromatic

4-cycle system of order n. Furthermore, letting $n_4(k)$ represent the smallest admissible such w_k, $n_4(k)$ is the smallest integer for which there exists a k-chromatic 4-cycle system. Turning our attention to even cycle systems in general, we have proven by constructive methods that for any integers $r \geq 2$ and $k \geq 3$, there is a k-chromatic $(2r)$-cycle system.

The results presented in Chapter 2 lead to some natural questions, which we now state.

Open Problem 1. For integers $k \geq 2$ and $m \geq 3$, does there exist a k-chromatic, uniquely k-colourable m-cycle system?

Open Problem 2. In Section 2.2.2, it was determined that $n_4(2) = 9$, $17 \leq n_4(3) \leq 49$ and $17 \leq n_4(k) \leq hk^{\ell(\ell-2)(\ell-2)!+1} + 1$ for any $k > 3$, where ℓ is the least even integer greater than or equal to k and h is the least multiple of 8 greater than or equal to ℓ. What is the exact value of $n_4(k)$ for $k \geq 3$?

Open Problem 3. Let $r \geq 3$ be an integer. For an integer $k \geq 2$, is there an admissible order $n_{2r}(k)$ such that for any admissible $n \geq n_{2r}(k)$ there is a k-chromatic $(2r)$-cycle system of order n?

Open Problem 4. For each integer $k \geq 3$ and each integer $r \geq 2$, does there exist a k-chromatic $(2r+1)$-cycle system?

Chapter 3 dealt with invariants for cycle systems and their application in the enumeration of pairwise nonisomorphic cycle systems of small order. We began by describing some invariants for cycle systems. We then discussed a method by which invariants assist in the generation of pairwise nonisomorphic cycle systems. In subsequent sections, we presented the results of our application of this algorithm

and discussed the sensitivity of invariants among various groups of cycle systems as a measure of their effectiveness in distinguishing nonisomorphic systems. In particular, we determined the exact number of pairwise nonisomorphic 11-cycle systems of order 11; this value was previously unknown. We finish by stating an open problem which arises from the results of Chapter 3.

Open Problem 5. Recall from Section 3.2 that $N_m(n)$ denotes the number of pairwise nonisomorphic m-cycle systems of order n. We have generated large numbers of 4-cycle systems of order 17, 5-cycle systems of order 11 and 6-cycle systems of order 13. What are the exact values of $N_4(17)$, $N_5(11)$ and $N_6(13)$?

Bibliography

[1] B. Alspach, and H. Gavlas, Cycle decompositions of K_n and $K_n - I$, *J. Combin. Theory Ser. B* 81 (2001), 77–99.

[2] B. Alspach and B.N. Varma. Decomposing complete graphs into cycles of length $2p^e$. Combinatorics 79 (Proc. Colloq., Univ. Montréal, Montréal, Qué., 1979), Part II. *Ann. Discrete Math.* 9 (1980), 155–162.

[3] I. Anderson. *Combinatorial Designs and Tournaments*, Oxford Lecture Series in Mathematics and its Applications 6, Clarendon Press, Oxford University Press, New York, 1997.

[4] M. Behzad and E.S. Mahmoodian. Graphs versus designs - a quasisurvey. In Y. Alavi et al., Eds. *Graph Theory, Combinatorics and Applications, Vol. 1*, Proceedings of the Sixth Quadrennial International Conference on the Theory and Applications of Graphs, Wiley, New York, 1991, pp. 125–142.

[5] A. Bruen, L. Haddad, and D. Wehlau. Caps and colouring Steiner triple systems. *Des. Codes Cryptogr.* 13 (1998), 51–55.

[6] G. Brunel. Sur les deux systèmes de triades de treize elements. *J. Math. Soc. Sci. Phys. nat. Bordeaux (6)* 2 (1902), 1–23.

[7] D. Bryant, C.D. Leach and C.A. Rodger. Hamilton decompositions of complete bipartite graphs with a 3-factor leave. *Australas. J. Combin.* 31 (2005), 331–336.

[8] A.C. Burgess and D.A. Pike. Colouring 4-cycle systems. *J. Combin. Des.*, to appear.

[9] C.J. Colbourn. Hamiltonian decompositions of complete graphs. *Ars Combin.* 14 (1982), 261–269.

[10] C.J. Colbourn, J. Dinitz and A. Rosa. Bicoloring Steiner triple systems. *Electron. J. Combin.* 6 (1999), Research paper 25.

[11] C.J. Colbourn and A. Rosa. *Triple Systems.* Clarendon Press, Oxford University Press, New York, 1999.

[12] M.J. Colbourn. An analysis technique for Steiner triple systems. *Proceedings of the Tenth Southeastern Conference on Combinatorics, Graph Theory and Computing*, 1979, 289–303.

[13] M.J. Colbourn and R.A. Mathon. On cyclic Steiner 2-designs. In Topics on Steiner Systems, *Ann. Discrete Math.* 7, North Holland, Amsterdam, 1980, pp. 215–253.

[14] F.N. Cole. The triad systems of thirteen letters. *Trans. Amer. Math. Soc.* 14 (1913), 1–5.

[15] F.N. Cole, L.D. Cummings and H.S. White. The complete enumeration of triad systems in 15 elements. *Proc. Nat. Acad. Sci. U.S.A.* 3 (1917), 197–199.

[16] L.D. Cummings. On a method of comparison for triple systems. *Trans. Amer. Math. Soc.* 15 (1914), 311–237.

[17] M. de Brandes, K.T. Phelps and V. Rödl. Coloring Steiner triple systems. *SIAM J. Algebraic Discrete Methods* 3 (1982), 241–249.

[18] V. De Pasquale. Sui sistemi ternari di 13 elementi. *Rend. R. Ist. Lombardo Sci. Lett (2)* 32 (1899), 213–221.

[19] I.J. Dejter, P.I. Rivera-Vega, and A. Rosa. Invariants for 2-factorizations and cycle systems. *J. Combin. Math. Combin. Comput.* 16 (1994), 129–152.

[20] G.A. Dirac. On Hamilton circuits and Hamilton paths. *Math. Ann.* 197 (1972), 57–70.

[21] Z. Dvořák et al. On pattern coloring of cycle systems. DIMACS Technical Report 2002-06, February 2002.

[22] P. Erdős and A. Hajnal. On the chromatic number of graphs and set-systems. *Acta Math. Acad. Sci. Hungar.* 17 (1966), 61–99.

[23] A.D. Forbes, M.J. Grannell, and T.S. Griggs. Independent sets in Steiner triple systems. *Ars Combin.* 72 (2004), 161–169.

[24] A.D. Forbes, M.J. Grannell, and T.S. Griggs. On colourings of Steiner triple systems. *Discrete Math.* 261 (2003), 255–276.

[25] J. Fugère, L. Haddad, and D. Wehlau. 5-chromatic Steiner triple systems. *J. Combin. Des.* 2 (1994), 287–299.

[26] P.B. Gibbons. *Computing Techniques for the Construction and Analysis of Block Designs*, Ph.D. Thesis, University of Toronto, 1976.

[27] M. Gionfriddo and G. Quattrocchi. Colouring 4-cycle systems with equitably coloured blocks. *Discrete Math.* 284 (2004), 137–148.

[28] L. Haddad. On the chromatic numbers of Steiner triple systems. *J. Combin. Des.* 7 (1999), 1–10.

[29] M. Hall, Jr. and J.D. Swift. Determination of Steiner triple systems of order 15. *Math. Tables Aids Comput.* 9 (1955), 146–152.

[30] P. Horák. On the chromatic number of Steiner triple systems of order 25. *Discrete Math.*, to appear.

[31] B.W. Jackson. Some cycle decompositions of complete graphs. *J. Combin. Inform. System Sci.* 13 (1988), 20–32.

[32] P. Kaski and P.R.J. Östergård. The Steiner triple systems of order 19. *Math. Comp.* 73 (2004), 2075–2092.

[33] T.P. Kirkman. On a problem in combinations. *Cambr. and Dublin Math. J.* 2 (1847), 191–204.

[34] A. Kotzig. On decompositions of the complete graph into $4k$-gons. *Mat.-Fyz. Časopis Sloven. Akad. Vied* 15 (1965), 229–233.

[35] C.C. Lindner, and C.A. Rodger. Decompositions into cycles II: Cycle systems. In J. Dinitz and D. Stinson, Eds. *Contemporary design theory: A collection of surveys*, Wiley, New York, 1992, pp. 325–369.

[36] R.A. Mathon, K.T. Phelps, and A. Rosa, Small Steiner triple systems and their properties, *Ars Combin.* 15 (1983), 3–110 and 16 (1983), 286.

[37] B.D. McKay. *nauty* user's guide. `http://cs.anu.edu.au/~bdm/nauty/nug.pdf`.

[38] S. Milici, A. Rosa and V. Voloshin. Colouring Steiner triple systems with specified block colour patterns. *Discrete Math.* 240 (2001), 145–160.

[39] S. Milici and Zs. Tuza. Cycle systems without 2-colourings. *J. Combin. Des.* 2 (1996), 135–142.

[40] S. Milici and Zs. Tuza. Disjoint blocking sets in cycle systems. *Discrete Math.* 208/209 (1999), 451–462.

[41] J. Pelikán. Properties of balanced incomplete block designs, *Combinatorial Theory and its Applications*, III, Proc. Colloq. Balatonfüred 1969, North-Holland, Amsterdam, 1970, pp. 869–889.

[42] L.P. Petrenjuk and A.J. Petrenjuk. An enumeration method for nonisomorphic combinatorial designs. In C.C. Lindner and A. Rosa, Eds. Topics on Steiner Triple Systems, *Ann. Discrete Math.* 7, North Holland, Amsterdam, 1980, pp. 265–276.

[43] G. Quattrocchi. Colouring 4-cycle systems with specified block colour patterns: the case of embedding P_3-designs. *Electron. J. Combin.* 8 (2001), Research paper 24.

[44] A. Rosa. On cyclic decompositions of the complete graph into $(4m + 2)$-gons. *Mat.-Fyz. Časopis Sloven. Akad. Vied* 16 (1966), 349–352.

[45] A. Rosa. Steiner triple systems and their chromatic number. *Acta Fac. Rerum Natur. Univ. Comenian. Math.* 24 (1970), 159–174.

[46] M. Šajna, Cycle decompositions III: Complete graphs and fixed length cycles, *J. Combin. Des.* 10 (2002), 27–78.

[47] D. Sotteau. Decompositions of $K_{m,n}$ ($K^*_{m,n}$) into cycles (circuits) of length $2k$. *J. Combin. Theory Ser. B* 30 (1981), 75–81.

[48] D.R. Stinson. A comparison of two invariants for Steiner triple systems: fragments and trains. *Ars Combin.* 16 (1983), 69–76.

[49] C. Treash. The completion of finite incomplete Steiner triple systems with applications to loop theory. *J. Combin. Theory Ser. A* 10 (1971), 259–265.

[50] M. Walecki. Reported in É. Lucas. *Récréations Mathématiques*, Vol. 2., Gauthier-Villars et Fils, Paris, 1896.

[51] D.B. West. *Introduction to Graph Theory*, 2nd edition, Prentice-Hall, Inc., Upper Saddle River, NJ, 2001.

[52] H.S. White, F.N. Cole and L.D. Cummings. Complete classification of the triad systems on fifteen elements. *Memoirs Nat. Acad. Sci. U.S.A.* 14 (1919), 1–89.

[53] K. Zulauf. *Über Tripelsysteme von 13 Elementen*, Dissertation Giessen, Wintersche Buchdruckerei, Darmstadt, 1897.

Appendix 1

A program for generation of n-cycle systems of order n

```
/*
 *   This program, given an odd integer n>=5, finds all non-isomorphic
 *   n-cycle systems of order n.
 *
 *   Authors:  A.C. Burgess and D.A. Pike
 *
 *   Compiled using: gcc -lm ham_c_s.c nauty.c nautil.c naugraph.c
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>


#define WK_SP_SZ 1000
#include "nauty.h"
#define Pseudo_Byte_Length 6



typedef struct path_node     // Structure used to find Hamiltonian paths.
   {
   int vertex;
   struct path_node *next;
   } path_node;

typedef struct cycle_sys    // An array of arrays to store cycle systems.
   {
   char *cycle;
   } cycle_sys;

typedef struct cycle_node        // A tree of cycle systems.
```

```
   {
   struct cycle_sys *system;
   struct cycle_node *left;
   struct cycle_node *right;
   } cycle_node;

typedef struct vector_node
   {
   int *vect;
   struct vector_node *next;
   } vector_node;

typedef struct inv_tree_node       // Used to form the invariant tree.
   {
   int *inv;                          // The cycle structure invariant.
   int groupsize;                     // The automorphism group order.
   int *S_3;                          // Sum-bicolour vector of rank 3.
   int *S_4;                          // Sum-bicolour vector of rank 4.
   int *S_5;                          // Sum-bicolour vector of rank 5.
   int *S_6;                          // Sum-bicolour vector of rank 6.
   struct cycle_node *cycle_systems;  // Systems with a given invariant.
   struct inv_tree_node *left;
   struct inv_tree_node *right;
   } inv_tree_node;

typedef struct vector_array              // Stores vectors.
   {
   int *vec;
   } vector_array;

typedef struct partial_system_tree      // Stores partial systems of
                                        // cycle systems, used to discard
                                        // isomorphic partial systems.
   {
   struct cycle_sys *partial_system;
   struct partial_system_tree *left;
   struct partial_system_tree *right;
   } partial_system_tree;



/* Global variables */

int cycle_number;
int total_cycles;               // Number of cycles in the system.
cycle_sys *cycle_system;        // The cycle system.
int cycle_system_count;         // The number of nonisomorphic systems.

int vector_count;
int B_n;
int id_length;
```

```
char *id;                          // Identifier for a cycle system.
char *id2;
int invariant_count;               // The number of distinct invariants.
int sum_vertices;                  // Used in finding cycle structure.

double groupsize_dbl;
int groupsize_int;




// Subroutines to find and store the possible vectors which will be used
// to form the invariant for a cycle system.

void generate_vectors(vector_node **, vector_node **, int, int);
void store_vector(vector_node **, vector_node **, int *, int);
void print_vectors(vector_node *, int);

// Subroutines used in finding the cycle systems.

char what_is (char *, int, int, int);
void set_bit (char *, int, int, int, char);
void build_Ham_path (char *, int, int, int, path_node **, int *);
void retreat_Ham_path (char *, int, int, int, path_node **, int *, int *);

void is_decomposable(char *, int, vector_array *, inv_tree_node **,
  inv_tree_node **, cycle_node **, int *, char *, partial_system_tree **,
  char *, int *, int *, int *, int *, int *, int *);



// Subroutines used in finding the cycle structure once a cycle system has been
// found.

void find_invariant(cycle_sys *, int *, char *, int *, int, vector_array *);
int find_vector(int *, vector_array *, int);

int find_cycle_length(char, char, char, cycle_sys *, char *, int);
void put_in_order(int *, int);
int find_edge(char, char, cycle_sys *, int);


// Subroutines used for finding the sum-bicolour sequences(s).

void find_S_3(cycle_sys *, int *, int);
void find_S_4(cycle_sys *, int *, int);
void find_S_5(cycle_sys *, int *, int);
void find_S_6(cycle_sys *, int *, int);
void order_array(int *);
```

```
// Subroutines to find and compare the automorphism group size.

void find_group_size(cycle_sys *, int, char *);
int compare_integers(int, int);


// Subroutines for comparing invariants and storing the invariants, along
// with their associated cycle systems, in a tree.

int compare_invariant(int, int *, int *, int *, int *, int *, int *, int *,
   int *, int *, int *, int);
int compare_integer_array(int *, int *, int);
int is_new_CS(cycle_sys *, int *, inv_tree_node **, inv_tree_node **, int,
   char *, int *, int *, int *, int *);


// Subroutines for determining isomorphism and storing nonisomorphic
// cycle systems.

int is_new_graph(char *, int *, cycle_sys *, int, inv_tree_node **,
   inv_tree_node **, int, int *, int *, int *, int *);
void get_id(char *, int, char *, int);
int insert_system(int *, cycle_sys *, int, char *, char *,
   inv_tree_node **, inv_tree_node **, int, char *, int *, int *,
   int *, int *);


// Subroutines used to analyse the invariant.

int analyse_inv_tree(inv_tree_node *, int, int, int *);
int find_max(int *, int);
int count_cycle_systems(cycle_node *);


// Subroutines for testing partial system isomorphism.

int is_new_partial_system(cycle_sys *, int, char *, partial_system_tree **, int);
int insert_partial_system(partial_system_tree **, char *, cycle_sys *, int, char
   *, int, int);




int main (argc, argv)
   int argc;
   char *argv[];
   {
   char *A;                                    // Adjacency matrix.
   int i, j;                                   // Counters.
```

```
int n;                                      // The order of the system, and
                                            // the length of cycles in the
                                            // system.

partial_system_tree **partial_system_trees;     // Stores partial systems for
                                                // isomorphism comparison.

char *B;                                    // Adjacency matrix for
                                            // the graph associated
                                            // with a cycle system.

vector_node *vector_list;        // Used to store vectors, once found.
vector_node *vector_list_current;

inv_tree_node *inv_tree;         // A tree which stores the cycle
                                 // systems with their invariants.
inv_tree_node *tree_current;

cycle_node *cycle_current;

int *invariant;                  // Used to find the invariant of a
                                 // given cycle system.

vector_array *vector_list_array;  // A list of vectors used in finding
                                  // invariants
vector_node *temp_node;

int *c_s_cnt_by_inv;             // An array to be used to keep track
                                 // of the number of cycle systems
                                 // associated with each invariant.

int max;                         // The maximum number of cycle systems
                                 // associated with a given invariant.

int inv_number;                  // Used to find the number of
                                 // invariants that have a particular
                                 // number of systems.

char *vertices;                  // Arrays used in finding the
int *cycle_lengths;              // cycle structure invariant.

int *vertices_used_in_path;      // An array used in finding
                                 // Hamilton paths.

int *S_3;
int *S_4;
int *S_5;
int *S_6;


vector_count = 0;
```

```
cycle_system_count = 0;

if(argc !=2)
   {
   printf("\nUsage: %s n\n", argv[0]);
   exit(0);
   }

n = atoi (argv[1]);

if((n%2==0)||(n<5))
   {
   printf("Please enter an odd integer greater than 4.\n");
   exit(0);
   }

printf("%d\n", n);


// Initialize lists and trees.

vector_list=NULL;
if ((vector_list_current = (vector_node *) malloc (1 * sizeof
   (vector_node))) == NULL)
   {
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
   }

inv_tree=NULL;


// Find the vectors which will be used to find invariants and count
// them.

generate_vectors(&vector_list, &vector_list_current, n, (n-1)/2);

// Store the vectors in an array, so they will not have to be all gone
// through when finding the invariant.

if((vector_list_array = (vector_array *) malloc (vector_count * sizeof
     (vector_array))) == NULL)
   {
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
   }

temp_node = vector_list;

for(i=0;i<vector_count;i++)
   {
```

```
   if(( (vector_list_array[i]).vec = (int *) malloc ( ((n-1)/2) *
         sizeof (int))) == NULL)
      {
      printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
      exit(1);
      }

   for(j=0;j<(n-1)/2;j++)
      ((vector_list_array[i]).vec)[j] = (temp_node->vect)[j];

   free(temp_node->vect);
   temp_node = temp_node->next;
   free(vector_list);
   vector_list = temp_node;
   }


// We know how much memory to set aside for the invariant vector.

if((invariant = (int *) malloc (vector_count * sizeof (int))) == NULL)
   {
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
   }

for(i=0;i<vector_count;i++)
   invariant[i]=0;


// Initialize the vertices and cycle_lengths arrays, which will be
// used in finding the cycle structure invariant.

if((vertices = (char *) malloc (n * sizeof (char))) == NULL)
   {
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
   }

if((cycle_lengths = (int *) malloc (((n-1)/2) * sizeof(int))) == NULL)
   {
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
   }

// Initialize the adjacency matrices.

if ((A = (char *) malloc ( (n*(n-1))/2 * sizeof (char))) == NULL)
   {
   printf ("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit (1);
   }
```

```
for(i=0; i<(n*(n-1))/2; i++)
   A[i]=1;

B_n = n + ((n*(n-1))/2);

if((B = (char *) malloc ( ((B_n*B_n-B_n)/2) * sizeof(char))) == NULL)
   {
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
   }

for(i=0;i<((B_n*B_n-B_n)/2);i++)
   B[i]=0;


// Find id length and allocate memory for ids.

id_length= (B_n*(B_n-1))/2 / Pseudo_Byte_Length;
if (( (B_n*(B_n-1)/2) % Pseudo_Byte_Length)>0)
   id_length++;
id_length++;


if((id = (char *) malloc (id_length * sizeof(char)))==NULL)
   {
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
   }

for(i=0;i<id_length;i++)
   id[i]=0;

if((id2 = (char *) malloc (id_length * sizeof(char)))==NULL)
   {
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
   }

for(i=0;i<id_length;i++)
   id2[i]=0;


// Find the number of n-cycles in the system.

total_cycles = (n*(n-1))/(2*n);


// Allocate memory for the partial id trees and the cycle system.

if (((partial_system_trees) = (partial_system_tree **) malloc (total_cycles *
```

```
  sizeof(partial_system_tree *))) == NULL)
   {
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
   }

for(i=0;i<total_cycles;i++)
   {
   partial_system_trees[i]=NULL;
   }

if ((cycle_system = (cycle_sys *) malloc (total_cycles * sizeof
  (cycle_sys))) == NULL)
   {
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
   }

for(i=0;i<total_cycles;i++)
   {
   if (( ((cycle_system[i]).cycle) = (char *) malloc (n * sizeof
     (char))) == NULL)
      {
      printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
      exit(1);
      }

   for(j=0;j<n;j++)
      ((cycle_system[i]).cycle)[j] = 0;
   }


// Without loss of generality, we may assume that the first cycle in
// the system is (1,2,...,n).

for(i=0;i<n;i++)
   ((cycle_system[0]).cycle)[i] = i+1;

for(i=1;i<n;i++)
   set_bit(A,n,i,i+1,0);
set_bit(A,n,1,n,0);



// Allocate memory for the S_3, S_4, S_5 and S_6 invariant vectors.

if ((S_3 = (int *) malloc (total_cycles * sizeof (int))) == NULL)
   {
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
   }
```

```
for(i=0;i<total_cycles;i++)
   S_3[i]=0;


if ((S_4 = (int *) malloc (total_cycles * sizeof (int))) == NULL)
   {
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
   }

for(i=0;i<total_cycles;i++)
   S_4[i]=0;

if ((S_5 = (int *) malloc (total_cycles * sizeof (int))) == NULL)
   {
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
   }

for(i=0;i<total_cycles;i++)
   S_5[i]=0;


if ((S_6 = (int *) malloc (total_cycles * sizeof (int))) == NULL)
   {
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
   }

for(i=0;i<total_cycles;i++)
   S_6[i]=0;



cycle_number=2;
invariant_count = 0;


// Initialize the vertices_used_in_path array, to be used in finding
// Hamilton cycles.

if (( vertices_used_in_path = (int *) malloc (n * sizeof (int))) == NULL)
   {
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
   }

// Find the n-cycle decompositions of K_n.

is_decomposable(A, n, vector_list_array, &inv_tree,
```

```
  &tree_current, &cycle_current, invariant, B, partial_system_trees,
  vertices, cycle_lengths, vertices_used_in_path, S_3, S_4, S_5, S_6);


printf("\nFound %d non-isomorphic cycle systems.\n",
  cycle_system_count);
fflush(stdout);

if((c_s_cnt_by_inv = (int *) malloc (invariant_count * sizeof (int)))
  == NULL)
   {
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
   }

analyse_inv_tree(inv_tree, n, 0, c_s_cnt_by_inv);
printf("\n");


// Find the maximum number of systems per invariant.

max = find_max (c_s_cnt_by_inv, invariant_count);

// Output the number of invariants with i cycle system,
// for each i between 1 and max.

for(i=1; i<=max; i++)
   {
   inv_number=0;
   for(j=0;j<invariant_count;j++)
      {
      if(c_s_cnt_by_inv[j]==i)
         inv_number++;
      }
   printf("There are %d invariants with exactly %d cycle systems.\n",
     inv_number, i);
   }

printf("\nFound %d non-isomorphic cycle_systems.\n",
  cycle_system_count);
printf("There are %d different invariants.\n", invariant_count);
printf("The maximum number of systems per invariant is %d\n", max);

free(cycle_system);
free(A);
free(vector_list_array);
free(id);
free(id2);
free(B);
free(invariant);
free(vertices);
```

```
   free(cycle_lengths);
   free(vertices_used_in_path);

   return(0);
   }


// This subroutine finds the n-cycle systems of order n.

void is_decomposable (A, n, vector_list_array, inv_tree, tree_current,
cycle_current, invariant, B, partial_system_trees, vertices, cycle_lengths,
vertices_used_in_path, S_3, S_4, S_5, S_6)
   char *A;
   int n;
   vector_array *vector_list_array;
   inv_tree_node **inv_tree;
   inv_tree_node **tree_current;
   cycle_node **cycle_current;
   int *invariant;
   char *B;
   partial_system_tree **partial_system_trees;
   char *vertices;
   int *cycle_lengths;
   int *vertices_used_in_path;
   int *S_3;
   int *S_4;
   int *S_5;
   int *S_6;
   {
   char decomposable = 0;
   path_node *Ham_path, *temp1, *temp2;
   int index1, i, j;
   int neighbour = 0;
   int neighbour_cnt = 0;

   /* For each vertex adjacent to vertex 1, we search for
    * a Hamilton path from vertex 1 to its neighbour in the hopes that
    * such a cycle will enable us to decompose the graph... actually
    * it is only necessary to check 1 or 2 neighbours, depending upon
    * whether the graph has even or odd degree, for if we find that
    * the edge (1, neighbour) for some neighbour is not in a Hamilton
    * cycle, then either there is no Hamilton decomposition or this
    * edge must be in the 1-factor that remains when we remove all
    * Hamilton cycles from a graph of odd degree.
    */


   for (index1 = 2 ; ((index1 <= n) && (!decomposable)
                      && (neighbour_cnt < 1))
                  ; index1 ++)
      if (what_is (A, n, 1, index1))
```

```
{
neighbour_cnt ++;
neighbour = index1;
Ham_path = NULL;

// Find a Hamilton path from 1 to neighbour.

build_Ham_path (A, n, 1, neighbour, &Ham_path,
  vertices_used_in_path);
while (Ham_path != NULL)
   {
   // Remove the edges in the Hamilton cycle from A.

   set_bit (A, n, 1, neighbour, 0);
   for (temp1 = Ham_path ; temp1->next != NULL ; temp1 = temp1->next)
      set_bit (A, n, temp1->vertex, temp1->next->vertex, 0);

   temp2 = Ham_path;
   for(i=0;i<n;i++)
      {
      ((cycle_system[cycle_number-1]).cycle)[i] = (temp2->vertex);
      temp2 = temp2->next;
      }


   if(cycle_number==total_cycles)
      {
      // We have a complete cycle system.  Determine if it is
      // indeed a new system.

      decomposable = 1;

      find_invariant(cycle_system, invariant, vertices,
        cycle_lengths, n, vector_list_array);

      find_group_size(cycle_system, n, B);

      find_S_3(cycle_system, S_3, n);

      if(n>=7)
         find_S_4(cycle_system, S_4, n);

      if(n>=9)
         find_S_5(cycle_system, S_5, n);

      if(n>=11)
         find_S_6(cycle_system, S_6, n);

      if(is_new_CS(cycle_system, invariant, inv_tree,
        tree_current, n, B, S_3, S_4, S_5, S_6))
         {
```

```
            cycle_system_count++;
            for(i=0;i<total_cycles;i++)
               {
               for(j=0;j<n;j++)
                  {
                  printf("%d ", ((cycle_system[i]).cycle[j]) );
                  }
               }
            printf("\n");
            fflush(stdout);
            }

         }

      // If we have added a cycle, but the system is not complete,
      // determine if the resulting partial system is new.

      else if (is_new_partial_system (cycle_system, n, B,
        partial_system_trees, cycle_number))
         {
         cycle_number++;

         // Try to decompose what remains of the graph.

         is_decomposable(A, n, vector_list_array,
           inv_tree, tree_current, cycle_current, invariant, B,
           partial_system_trees, vertices, cycle_lengths,
           vertices_used_in_path, S_3, S_4, S_5, S_6);
         cycle_number--;
         }

      /* put the cycle back in A */
      set_bit (A, n, 1, neighbour, 1);
      for (temp1 = Ham_path ; temp1->next != NULL ; temp1 = temp1->next)
         set_bit (A, n, temp1->vertex, temp1->next->vertex, 1);

      build_Ham_path (A, n, 1, neighbour, &Ham_path,
        vertices_used_in_path);
      }  /* end while */
   }

/* free up any memory used by the last path */
while (Ham_path != NULL)
   {
   temp1 = Ham_path;
   Ham_path = Ham_path->next;
   free (temp1);
   }
return;
}
```

```
// This subroutine tests to see whether the edge between vertices i and j
// remains, returning 1 if i and j are adjacent and 1 otherwise.

char what_is (A, n, i, j)
   char *A;
   int n;
   int i;
   int j;
   {
   if (i > j)
      return what_is (A, n, j, i);
   else if (i == j)
      return 0;
   else
      return A [ (i-1)*n - (i*(i-1))/2 + (j-i) - 1 ];
   }


// This subroutine takes away or puts back an edge in the adjacency
// matrix, as required.

void set_bit (A, n, i, j, value)
   char *A;
   int n;
   int i;
   int j;
   char value;
   {
   if (i > j)
      set_bit (A, n, j, i, value);
   else if (i == j)
      {
      if (value)
         {
         printf ("\n\nTrying to set A [%d, %d]\n\n", i, j);
         exit (1);
         }
      }
   else
      A [ (i-1)*n - (i*(i-1))/2 + (j-i) - 1 ] = value;
   }


// This subroutine, given two vertices x and y, builds a Hamilton path,
// if possible, from x to y.

void build_Ham_path (A, n, x, y, Ham_path, vertices_used_in_path)
   char *A;
   int n;
   int x;
```

```
int y;
path_node **Ham_path;
int *vertices_used_in_path;
{
int path_length;
path_node *temp;
int index;
int have_added;

if (*Ham_path == NULL)
   {
   if ((*Ham_path = (path_node *) malloc (sizeof (path_node))) == NULL)
      {
      printf ("\n\nAllocation of memory failed...line %d\n", __LINE__);
      exit (1);
      }
   (*Ham_path)->vertex = x;
   (*Ham_path)->next = NULL;
   path_length = 1;
   }
else
   {
   // Find the path length.

   path_length = 0;
   for (temp = *Ham_path ; temp != NULL ; temp = temp->next)
      path_length ++;
   }

if (path_length == n)
   retreat_Ham_path (A, n, x, y, &(*Ham_path), &path_length,
   vertices_used_in_path);

while ((path_length < n) && (*Ham_path != NULL))
   {
   for (index = 1 ; index <= n ; index ++)
      vertices_used_in_path [index - 1] = 0;
   for (temp = *Ham_path ; temp->next != NULL ; temp = temp->next)
      vertices_used_in_path [temp->vertex - 1] = 1;
   have_added = 0;

   // Try to find a vertex that can be added to the path.

   for (index = 1 ; ((index <= n) && (!have_added)) ; index ++)
      {
      /* make sure vertex index is adjacent to vertex temp->vertex */
      if (what_is (A, n, temp->vertex, index))
         {
         /* make sure vertex index isn't already used in the path */
         if (!(vertices_used_in_path [index - 1]))
            {
```

```
            /* if index == y, we can only add it if it is to be the last
             * vertex in the path
             */
            if ((index != y) || ((index == y) && (path_length == (n-1))))
               {
               /* add the vertex to the path */
               if ((temp->next = (path_node *)
                                 malloc (sizeof (path_node))) == NULL)
                  {
                  printf ("\n\nAllocation of memory failed...line %d\n",
                          __LINE__);
                  exit (1);
                  }
               temp->next->vertex = index;
               temp->next->next = NULL;
               have_added = 1;
               path_length ++;
               }
            }
         }
      }

   if (!have_added)
      retreat_Ham_path (A, n, x, y, &(*Ham_path), &path_length,
        vertices_used_in_path);
   }  /* end while */
}



void retreat_Ham_path (A, n, x, y, Ham_path, path_length,
vertices_used_in_path)
   char *A;
   int n;
   int x;
   int y;
   path_node **Ham_path;
   int *path_length;
   int *vertices_used_in_path;
   {
   path_node *temp;
   int dropped_vertex;
   int index;
   char have_retreated = 0;

   for (temp = *Ham_path ; temp->next != NULL ; temp = temp->next);
   dropped_vertex = temp->vertex;

   if (dropped_vertex == x)
      /* we are completely unable to proceed to a next valid path,
       * so return a NULL path
```

```
    */
   {
   free (*Ham_path);
   *Ham_path = NULL;
   *path_length = 0;
   }
else
   {
   for (index = 1 ; index <= n ; index ++)
      vertices_used_in_path [index - 1] = 0;

   for (temp = *Ham_path ; temp->next->vertex != dropped_vertex
                         ; temp = temp->next)
      vertices_used_in_path [temp->vertex - 1] = 1;

   for (index = dropped_vertex + 1 ; ((index <= n) && (!have_retreated))
                                   ; index ++)
      {
      /* is vertex index is adjacent to vertex temp->vertex ? */
      if (what_is (A, n, temp->vertex, index))
         {
         /* make sure vertex index isn't already used in the path ? */
         if (!(vertices_used_in_path [index - 1]))
            {
            /* if index == y, we can't add it as a vertex here */
            if (index != y)
               {
               /* add the vertex to the path */
               temp->next->vertex = index;
               have_retreated = 1;
               }
            }
         }
      }
   if (!have_retreated)
      {
      /* we were unable to replace the last vertex with a new vertex,
       * so we must delete the last vertex in the path and attempt to
       * replace the 2nd last vertex.
       */
      free (temp->next);
      temp->next = NULL;
      (*path_length) --;
      retreat_Ham_path (A, n, x, y, &(*Ham_path), &(*path_length),
        vertices_used_in_path);
      }
   }
}
```

```
// This subroutine generates the possible vectors to be used for finding
// the invariant of a cycle system, each representing a partition of n-1
// into nonnegative even integers.

void generate_vectors(vector_list, vector_list_current, n, length)
   vector_node **vector_list;
   vector_node **vector_list_current;
   int n;
   int length;
   {
   int i,j;
   int sum;
   int store;
   int *vector;

   // Allocate memory for the vector.

   if((vector = (int *) malloc ((length) * sizeof (int))) == NULL)
      {
      printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
      exit(1);
      }

   // Initialize the vector.  The first solution is (n-1, 0, ...,0).
   // Store this vector.

   vector[0]=n-1;
   for(i=1;i<length;i++)
      vector[i]=0;

   store_vector((vector_list), (vector_list_current), vector,
      length);
   vector_count=1;

   // Find and store the other vectors.  Each vector will have entries in
   // nonincreasing order.

   while(vector[length-1]==0)
      {
      for(i=length-2;i>=0;i--)
         {
         if(vector[i]>2)
            {
            vector[i]-=2;
            sum=0;
            for(j=0;j<=i;j++)
               {
               sum+=vector[j];
               }
            vector[i+1]=n-1-sum;
            for(j=i+2;j<length;j++)
```

```
            vector[j]=0;

         store=1;
         for(j=0;j<length-1;j++)
            {
            if(vector[j]<vector[j+1])
               {
               store=0;
               break;
               }
            }
         if(store==1)
            {
            store_vector((vector_list), (vector_list_current),
              vector, length);
            vector_count++;
            }

         break;
         }
      }
   }

return;
}


// This subroutine stores each vector, once found, in an ordered list.

void store_vector(vector_list, vector_list_current, vector, length)
   vector_node **vector_list;
   vector_node **vector_list_current;
   int *vector;
   int length;
   {
   int i;

   if(*vector_list==NULL)
      {
      if (( ((*vector_list_current)->vect) = (int *) malloc (length *
            sizeof (int))) == NULL)
         {
         printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
         exit(1);
         }

      for(i=0;i<length;i++)
         {
         ((*vector_list_current)->vect)[i]=vector[i];
         }
      (*vector_list_current) -> next=NULL;
```

```
      *vector_list=*vector_list_current;
      }

   else
      {
      if(( (*vector_list_current)->next = (vector_node *) malloc (1 *
           sizeof (vector_node))) == NULL)
         {
         printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
         exit(1);
         }
      (*vector_list_current)=(*vector_list_current)->next;

      if(( (*vector_list_current)->vect = (int *) malloc (length * sizeof
           (int))) == NULL)
         {
         printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
         exit(1);
         }

      for(i=0;i<length;i++)
         ((*vector_list_current)->vect)[i]=vector[i];

      (*vector_list_current) -> next=NULL;
      }

   }


// This subroutine, given a cycle system and the vector list found above,
// finds the cycle system's invariant.

void find_invariant(cycle_system, cycle_structure, vertices,
cycle_lengths, n, vector_list_array)
   cycle_sys *cycle_system;
   int *cycle_structure;
   char *vertices;
   int *cycle_lengths;
   int n;
   vector_array *vector_list_array;
   {
   int i,j,k;
   int cycle_length;                    // The length of a given cycle.
   char next_vertex;                    // The next vertex in the cycle.

   int cycle_lengths_count=0;           // A counter to record the
                                        // position in cycle_lengths to
                                        // store the next entry.

   int coordinate;                      // The coordinate of the
                                        // cycle_structure vector to be
```

```
                                        // incremented.
sum_vertices=n;


// Initialize the arrays.

for(i=0;i<vector_count;i++)
   {
   cycle_structure[i]=0;
   }

for(i=0;i<n;i++)
   vertices[i]=1;

for(i=0;i<(n-1)/2;i++)
   cycle_lengths[i]=0;

// For each pair {i,j} of vertices, find the lengths of the cycles in
// the double neighbourhood.  An edge {x,y} which occurs in both G_i
// and G_j will be thought of as a cycle of length 2.

for(i=1;i<n;i++)
   {
   for(j=i+1;j<=n;j++)
      {
      // First find the length of the cycle containing the element
      // corresponding to i and j.

      next_vertex=j;

      cycle_length = find_cycle_length(i, j, next_vertex,
        cycle_system, vertices, n);

      cycle_lengths[0]=cycle_length;
      cycle_lengths_count++;

      // Find the lengths of all other cycles.  Each time a vertex is
      // used in a cycle, its entry in the array vertices is set to 0.
      // Thus, when no vertices remain, every entry in this array will
      // be 0.

      while(sum_vertices>0)
         {
         // Find the next unused vertex.

         for(k=0;k<n;k++)
            {
            if(vertices[k]==1)
               {
               next_vertex=k+1;
               break;
```

```
                }
              }

            // Find and record the length of the cycle containing this
            // vertex.

           cycle_length = find_cycle_length(i, j,
              next_vertex, cycle_system, vertices, n);

            cycle_lengths[cycle_lengths_count] = cycle_length;
            cycle_lengths_count++;
            } // end while

         // Now that all lengths have been found, arrange them in
         // nonincreasing order.

         put_in_order(cycle_lengths, (n-1)/2);

         // Increment the coordinate of cycle_structure corresponding to
         // the cycle_lengths array.

         coordinate = find_vector(cycle_lengths, vector_list_array,
           (n-1)/2);

         cycle_structure[coordinate]++;

         // Reset values.
         for(k=0;k<n;k++)
            vertices[k]=1;
         sum_vertices=n;
         for(k=0;k<(n-1)/2;k++)
            cycle_lengths[k]=0;
         cycle_lengths_count=0;
         }
      }

   }


// This subroutine finds the length of the cycle containing vertex
// next_vertex in the double neighbourhood of vertices nbhd_vertex and
// other_vertex.

int find_cycle_length(nbhd_vertex, other_vertex, next_vertex,
cycle_system, vertices, n)
   char nbhd_vertex;
   char other_vertex;
   char next_vertex;
   cycle_sys *cycle_system;
   char *vertices;
   int n;
```

```
{
int i;
int temp;
int current_cycle;           // The current cycle in the cycle system.
int cycle_length=0;

// If this is the cycle containing the element corresponding to i and
// j, we must ensure that both of these values will be set to 0 in
// the array vertices.

if(next_vertex==other_vertex)
   {
   vertices[nbhd_vertex-1]=0;
   sum_vertices--;
   }

// We traverse the cycle until we find a repeated vertex, keeping
// track of the length along the way.

while(vertices[next_vertex-1]!=0)
   {
   // Find which cycle in the cycle system contains the edge
   // {nbhd_vertex, next_vertex}.  The other vertex in this cycle
   // adjacent to nbhd_vertex is the next vertex in the double
   // neighbourhood cycle.

   current_cycle = find_edge(nbhd_vertex, next_vertex, cycle_system, n);

   vertices[next_vertex-1]=0;
   sum_vertices--;

   // Find the next vertex in the double neighbourhood cycle.

   if( ((cycle_system[current_cycle]).cycle)[0] == nbhd_vertex)
      {
      if( ((cycle_system[current_cycle]).cycle)[1] == next_vertex)
         next_vertex = ((cycle_system[current_cycle]).cycle)[n-1];
      else
         next_vertex = ((cycle_system[current_cycle]).cycle)[1];
      }

   else if ( ((cycle_system[current_cycle]).cycle)[n-1] == nbhd_vertex)
      {
      if( ((cycle_system[current_cycle]).cycle)[0] == next_vertex)
         next_vertex = ((cycle_system[current_cycle]).cycle)[n-2];
      else
         next_vertex = ((cycle_system[current_cycle]).cycle)[0];
      }

   else
      {
```

```
         for(i=1;i<n-1;i++)
            {
            if( ((cycle_system[current_cycle]).cycle)[i] == nbhd_vertex)
               {
               if ( ((cycle_system[current_cycle]).cycle)[i+1] == next_vertex)
                  next_vertex = ((cycle_system[current_cycle]).cycle)[i-1];
               else
                  next_vertex = ((cycle_system[current_cycle]).cycle)[i+1];
               }
            }
         }

      cycle_length++;

      // The next vertex in the cycle will be in the cycle (in the
      // cycle system) containing the edge {next_vertex, other_vertex}.
      // Switch the values of nbhd_vertex and other_vertex.

      temp = nbhd_vertex;
      nbhd_vertex = other_vertex;
      other_vertex = temp;

      }

   return(cycle_length);
   }


// This subroutine, given an integer array and its length, puts the
// entries of the array in nonincreasing order.

void put_in_order(array, length)
   int *array;
   int length;
   {
   int i,j,temp;

   for(i=length-1;i>=0;i--)
      {
      for(j=i-1;j>=0;j--)
         {
         if(array[j]<array[i])
            {
            temp=array[j];
            array[j]=array[i];
            array[i]=temp;
            }
         }
      }
   return;
   }
```

```
// Find the vector in the vector list that is equal to a given vector and
// return the appropriate coordinate value.

int find_vector(vector, vector_list_array, length)
   int *vector;
   vector_array *vector_list_array;
   int length;
   {
   int compare;                  // Value will be set as 1, 0, or 2,
                                 // respectively, depending on whether the
                                 // vector to be tested is greater than,
                                 // equal to, or less than the middle one
                                 // in the vector array.

   int mid;                      // The coordinate of the middle vector in
                                 // the section of the array where the vector
                                 // is located.

   int left,right;               // The left and right coordinates,
                                 // respectively, of the section of the array
                                 // in which the vector is located.

   left=0;
   right=vector_count-1;

   while(right-left>0)
      {
      mid=(left+right)/2;
      compare = compare_integer_array(vector,
        (vector_list_array[mid]).vec, length);
      if(compare==0)
         {
         left=mid;
         right=mid;
         }
      else if(compare==1)
         {
         right=mid-1;
         }
      else
         {
         left=mid+1;
         }
      }

   return(left);
   }
```

```
// Compare componentwise two given integer arrays.
// Return 0 if they are equal, 1 if arr1 ">" arr2, or 2 if arr1 "<" arr2.

int compare_integer_array(arr1, arr2, length)
   int *arr1;
   int *arr2;
   int length;
   {
   int i;
   int return_value=0;

   for(i=0;i<length;i++)
      {
      if(arr1[i]>arr2[i])
         {
         return_value=1;
         break;
         }
      else if(arr1[i]<arr2[i])
         {
         return_value=2;
         break;
         }
      }

   return(return_value);
   }


// Check to see if the cycle system which has just been found is a new
// system, first by comparing invariants.  If the invariant is not yet in
// the tree, the system is new, so store it.  If the invariant is in the
// tree, test among the known systems with that invariant to see if the
// new system is isomorphic to any of them.

int is_new_CS(cycle_system, invariant, inv_tree, tree, n, B, S_3, S_4,
S_5, S_6)
   cycle_sys *cycle_system;
   int *invariant;
   inv_tree_node **inv_tree;
   inv_tree_node **tree;
   int n;
   char *B;
   int *S_3;
   int *S_4;
   int *S_5;
   int *S_6;
   {
   int return_value=1;
   int compare_value=0;
```

```
// If there are no entries yet in the tree, the system is the first
// found.  Store it.


if(*inv_tree==NULL)
   {
   invariant_count++;
   return_value = is_new_graph(B, invariant, cycle_system, n,
                  &(*inv_tree), &(*tree), compare_value, S_3, S_4,
                  S_5, S_6);
   }


// Otherwise, compare the system's invariant with those in the tree.

else
   {
   *tree = *inv_tree;
   while(1)
      {
      compare_value = compare_invariant (n, invariant, (*tree)->inv,
        S_3, (*tree)->S_3, S_4, (*tree)->S_4, S_5, (*tree)->S_5,
        S_6, (*tree)->S_6, (*tree)->groupsize);

      if(compare_value == 0)
         {
         return_value = is_new_graph(B, invariant, cycle_system, n,
            &(*inv_tree), &(*tree), compare_value, S_3, S_4, S_5,
            S_6);
         break;
         }

      else if(compare_value==1)
         {
         if( ((*tree)->right) != NULL)
            {
            *tree = (*tree)->right;
            }
         else
            {
            invariant_count++;
            return_value = is_new_graph (B, invariant, cycle_system, n,
               &(*inv_tree), &(*tree), compare_value, S_3, S_4, S_5,
               S_6);
            break;
            }
         }

      else if(compare_value==2)
         {
         if( ((*tree)->left) != NULL)
```

```
                {
                *tree = (*tree)->left;
                }
             else
                {
                invariant_count++;
                return_value = is_new_graph (B, invariant, cycle_system, n,
                   &(*inv_tree), &(*tree), compare_value, S_3, S_4, S_5,
                   S_6);
                break;
                }

             }
          }
       }

   return(return_value);
   }


// A subroutine which prints the invariants along with their associated
// cycle systems.  In addition, the number of cycle systems associated
// with each invariant is counted and the number is stored in the array
// c_s_cnt_by_inv.

int analyse_inv_tree(inv_tree, n, posn, c_s_cnt_by_inv)
   inv_tree_node *inv_tree;
   int n;
   int posn;                    // Keeps track of the position in the
                               // c_s_cnt_by_inv vector to be updated.
   int *c_s_cnt_by_inv;
   {
   int c_s_count=0;              // Counts the number of cycle systems
                                 // associated with a particular
                                 // invariant.

   if(inv_tree!=NULL)
      {
      posn=analyse_inv_tree((inv_tree)->left,n,posn,c_s_cnt_by_inv);

      // Count the number of systems associated with the current
      // invariant and store it in the vector.

      c_s_count=count_cycle_systems((inv_tree)->cycle_systems);

      c_s_cnt_by_inv[posn]=c_s_count;
      posn++;

      posn=analyse_inv_tree((inv_tree)->right,n,posn,c_s_cnt_by_inv);
      }
```

```
   return(posn);
   }


// This subroutine tests to see if a cycle system is isomorphic to any
// already found, by forming a unique identifier based on the canonical
// labelling of the system's associated graph.

int is_new_graph(B, invariant, cycle_system, n, inv_tree, curr_inv_node,
compare_value, S_3, S_4, S_5, S_6)
   char *B;
   int *invariant;
   cycle_sys *cycle_system;
   int n;
   inv_tree_node **inv_tree;
   inv_tree_node **curr_inv_node;
   int compare_value;
   int *S_3;
   int *S_4;
   int *S_5;
   int *S_6;
   {
   int is_new;
   int i;
   int cycle_number;

   // Get the adjacency matrix of the graph associated with the cycle
   // system.

   for(i=0;i<(B_n*B_n-B_n)/2;i++)
      B[i]=0;

   for(cycle_number=0; cycle_number<total_cycles; cycle_number++)
      {
      for(i=1;i<n;i++)
         set_bit(B,B_n,n+n*cycle_number+i,n+n*cycle_number+(i+1),1);
      set_bit(B,B_n,n+n*cycle_number+n,n+n*cycle_number+1,1);

      for(i=0;i<n;i++)
         set_bit(B, B_n, n+n*cycle_number+(i+1),
           ((cycle_system[cycle_number]).cycle)[i], 1);
      }

   for(i=0;i<id_length;i++)
      id[i]=0;

   // Find the system's identifier.

   get_id(B,n,id,B_n);

   // If the system is new, insert it and its id into the invariant tree.
```

```
   is_new = insert_system (invariant, cycle_system, n, id, id2,
                       &(*inv_tree), &(*curr_inv_node),
                       compare_value, B, S_3, S_4, S_5, S_6);

   return(is_new);
   }


// This subroutine uses nauty to find the canonical labelling of the graph
// associated with a given cycle system and then builds the system's id.

void get_id(B,n,id,number_of_vertices)
   char *B;
   int n;
   char *id;
   int number_of_vertices;      // Number of vertices in the graph.
   {
   int i,j,m;
   int u,v;
   int bit_cnt;

   DYNALLSTAT(graph,g,g_sz);
   DYNALLSTAT(graph,canong,canong_sz);
   DYNALLSTAT(int,lab,lab_sz);
   DYNALLSTAT(int,ptn,ptn_sz);
   DYNALLSTAT(int,orbits,orbits_sz);

   static DEFAULTOPTIONS(options);
   statsblk(stats);
   setword workspace[WK_SP_SZ];
   set *gv;

   m = (number_of_vertices+WORDSIZE-1)/WORDSIZE;
   nauty_check (WORDSIZE, m, number_of_vertices, NAUTYVERSIONID);

   if(WK_SP_SZ < 50 * m)
      {
      printf("WK_SP_SZ=%d is smaller than 50*m=%d\n", WK_SP_SZ, 50*m);
      exit(1);
      }

   DYNALLOC2(graph,g,g_sz,m,number_of_vertices,"malloc");
   DYNALLOC2(graph,canong,canong_sz,m,number_of_vertices,"malloc");
   DYNALLOC1(int,lab,lab_sz,number_of_vertices,"malloc");
   DYNALLOC1(int,ptn,ptn_sz,number_of_vertices,"malloc");
   DYNALLOC1(int,orbits,orbits_sz,number_of_vertices,"malloc");

   options.writeautoms = FALSE;
   options.writemarkers = FALSE;
   options.digraph = FALSE;
```

```
options.defaultptn=FALSE;
options.tc_level=0;
options.getcanon=TRUE;



for(i=1;i<=number_of_vertices;i++)
   {
   gv = GRAPHROW(g,i-1,m);
   EMPTYSET(gv,m);
   for(j=1;j<=number_of_vertices;j++)
      {
      if(what_is (B,B_n,i,j))
         {
         ADDELEMENT(gv,j-1);
         }
      }
   }


for(i=0;i<number_of_vertices;i++)
   {
   lab[i]=i;
   ptn[i]=1;
   }
ptn[n-1]=0;
ptn[number_of_vertices-1]=0;

// Get the canonical labelling.

nauty (g, lab, ptn, NILSET, orbits, &options, &stats, workspace,
       WK_SP_SZ, m, number_of_vertices, canong);

// Use the canonical labelling to form the id.

i=0;
bit_cnt=0;
for(u=1;u<number_of_vertices;u++)
   for(v=u+1;v<=number_of_vertices;v++)
      {
      id[i]+= (what_is (B, B_n, 1+lab[u-1], 1+lab[v-1]))
                << (Pseudo_Byte_Length - bit_cnt);
      bit_cnt++;
      if(bit_cnt==Pseudo_Byte_Length)
         {
         id[i]+=1;
         i++;
         bit_cnt=0;
         }
     }
return;
```

```
   }


// If the system is new, this subroutine inserts the system and its id in
// the appropriate place in the invariant tree.

int insert_system(invariant, cycle_system, n, id1, id2, inv_tree,
curr_inv_node, compare_value, B, S_3, S_4, S_5, S_6)
   int *invariant;
   cycle_sys *cycle_system;
   int n;
   char *id1;
   char *id2;
   inv_tree_node **inv_tree;
   inv_tree_node **curr_inv_node;
   int compare_value;
   char *B;
   int *S_3;
   int *S_4;
   int *S_5;
   int *S_6;
   {
   inv_tree_node *temp_inv_node;
   cycle_node *temp_node, *curr_cycle_node;
   char found_duplicate=0;
   int insert=0;
   int compare_ids;
   int i,j;

   if( (compare_value==0) && (*inv_tree!=NULL) )
   // Check among other systems with the same invariant to see if an
   // isomorphic copy has already been found.
      {
      curr_cycle_node=(*curr_inv_node)->cycle_systems;
      while ( (!found_duplicate) && (!insert) )
         {
         for(i=0;i<(B_n*B_n-B_n)/2;i++)
            B[i]=0;

         for(i=0; i<total_cycles; i++)
            {
            for(j=1;j<n;j++)
               set_bit(B, B_n, n+n*i+j, n+n*i+(j+1), 1);
            set_bit(B, B_n, n+n*i+n, n+n*i+1, 1);

            for(j=0;j<n;j++)
               set_bit(B, B_n, n+n*i+(j+1),
                 (((curr_cycle_node->system)[i]).cycle)[j], 1);
            }

         for(i=0;i<id_length;i++)
```

```
         id2[i]=0;

      get_id(B,n,id2,B_n);

      compare_ids=strcmp(id,id2);

      if (compare_ids==0)
         found_duplicate=1;
      else if (compare_ids<0)
         {
         if(curr_cycle_node->left!=NULL)
            curr_cycle_node = curr_cycle_node->left;
         else
            insert=1;
         }
      else
         {
         if(curr_cycle_node->right!=NULL)
            curr_cycle_node = curr_cycle_node->right;
         else
            insert=2;
         }
      }
   }

else
// This is either the first system, or the first with its invariant.
// Allocate memory for a new invariant node in the appropriate place
// in the tree.
   {
   if(*inv_tree==NULL)
      {
      if((temp_inv_node = (inv_tree_node *) malloc (1 *
         sizeof(inv_tree_node)))==NULL)
         {
         printf("\n\nAllocation of memory failed...line %d\n",
           __LINE__);
         exit(1);
         }

      temp_inv_node->cycle_systems=NULL;
      temp_inv_node->left=NULL;
      temp_inv_node->right=NULL;
      *inv_tree=temp_inv_node;
      *curr_inv_node=*inv_tree;
      }

   else if (compare_value==1)
      {
      if(( (*curr_inv_node)->right = (inv_tree_node *) malloc (1 *
        sizeof(inv_tree_node)))==NULL)
```

```
      {
      printf("\n\nAllocation of memory failed...line %d\n",
        __LINE__);
      exit(1);
      }

   *curr_inv_node = (*curr_inv_node)->right;
   (*curr_inv_node)->left = NULL;
   (*curr_inv_node)->right = NULL;
   (*curr_inv_node)->cycle_systems = NULL;
   }

else if (compare_value==2)
   {
   if(( (*curr_inv_node)->left = (inv_tree_node *) malloc (1 *
     sizeof(inv_tree_node)))==NULL)
      {
      printf("\n\nAllocation of memory failed...line %d\n",
        __LINE__);
      exit(1);
      }

   *curr_inv_node = (*curr_inv_node)->left;
   (*curr_inv_node)->left = NULL;
   (*curr_inv_node)->right = NULL;
   (*curr_inv_node)->cycle_systems = NULL;
   }

if(( (*curr_inv_node)->inv = (int *) malloc (vector_count *
  sizeof(int))) == NULL)
   {
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
   }

for(i=0;i<vector_count;i++)
   {
   ((*curr_inv_node)->inv)[i]=invariant[i];
   }

if(( (*curr_inv_node)->S_3 = (int *) malloc (total_cycles * sizeof
(int))) == NULL)
   {
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
   }

for(i=0;i<total_cycles;i++)
   {
   ((*curr_inv_node)->S_3)[i] = S_3[i];
   }
```

```
if(n>=7)
   {
   if(( (*curr_inv_node)->S_4 = (int *) malloc (total_cycles * sizeof
   (int))) == NULL)
      {
      printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
      exit(1);
      }

   for(i=0;i<total_cycles;i++)
      {
      ((*curr_inv_node)->S_4)[i] = S_4[i];
      }
   }

if(n>=9)
   {
   if(( (*curr_inv_node)->S_5 = (int *) malloc (total_cycles * sizeof
   (int))) == NULL)
      {
      printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
      exit(1);
      }

   for(i=0;i<total_cycles;i++)
      {
      ((*curr_inv_node)->S_5)[i] = S_5[i];
      }
   }


if(n>=11)
   {
   if(( (*curr_inv_node)->S_6 = (int *) malloc (total_cycles * sizeof
   (int))) == NULL)
      {
      printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
      exit(1);
      }

   for(i=0;i<total_cycles;i++)
      {
      ((*curr_inv_node)->S_6)[i] = S_6[i];
      }
   }

(*curr_inv_node)->groupsize = groupsize_int;

}
```

```
// If the system is indeed new, store it in the invariant tree.

if(!found_duplicate)
   {
   if((temp_node = (cycle_node *) malloc (1 * sizeof(cycle_node))) ==
     NULL)
      {
      printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
      exit(1);
      }

   if(( (temp_node)->system = (cycle_sys *) malloc (total_cycles *
     sizeof(cycle_sys))) == NULL)
      {
      printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
      exit(1);
      }

   for(i=0;i<total_cycles;i++)
      {
      if (( (((temp_node)->system)[i]).cycle = (char *) malloc (n *
        sizeof (char))) == NULL)
         {
         printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
         exit(1);
         }

      for(j=0;j<n;j++)
         ((((temp_node)->system)[i]).cycle)[j] =
           ((cycle_system[i]).cycle)[j];
      }

   temp_node->left=NULL;
   temp_node->right=NULL;

   if(insert==1) // which means we came from the case where
                 // compare_value=0, inv_tree wasn't null upon entry
                 // to subroutine, and the system must be stored in
                 // the left cycle system subtree
      {
      curr_cycle_node->left=temp_node;
      }

   else if(insert==2) // the system must be stored in the right
                      // subtree.
      {
      curr_cycle_node->right=temp_node;
      }

   else
```

```
      // This is the first system to be stored attached to its invariant node.
         {
         (*curr_inv_node)->cycle_systems = temp_node;
         insert=1;
         }
      }

   return(insert);
   }


// This subroutine finds the maximum value of those stored in an integer
// array.  It is used to find the maximum number of cycle systems
// associated with an invariant.

int find_max(array, length)
   int *array;
   int length;
   {
   int i;
   int max = array[0];

   for(i=1;i<length;i++)
      {
      if(array[i]>max)
         max = array[i];
      }

   return(max);
   }


// This subroutine determines if a partial system of a cycle system is isomorphic
// to one which has already been tested.

int is_new_partial_system(cycle_system, n, B, partial_system_trees, cycle_number)
   cycle_sys *cycle_system;
   int n;
   char *B;
   partial_system_tree **partial_system_trees;
   int cycle_number;
   {
   int i,j;
   int number_of_vertices=n+n*cycle_number;
   int is_new;

   // let's try to reduce the amount of memory needed, by not
   // checking for isomorphisms at every level.
   if (cycle_number > total_cycles - 2)
      return 1;
```

```
   // Set the values for the adjacency matrix of the graph associated
   // with the partial system.

   for(i=0;i<(B_n*B_n-B_n)/2;i++)
      B[i]=0;

   for(i=0; i<cycle_number; i++)
      {
      for(j=1;j<n;j++)
         set_bit(B, B_n,n+n*i+j, n+n*i+(j+1), 1);
      set_bit(B, B_n, n+n*i+n, n+n*i+1, 1);

      for(j=0;j<n;j++)
         set_bit(B, B_n, n+n*i+(j+1), ((cycle_system[i]).cycle)[j], 1);
      }

   // Find the partial system's identifier.

   for(i=0;i<id_length;i++)
      id[i]=0;

   get_id(B,n,id,number_of_vertices);

   // If the partial system is new, insert its id into the appropriate tree
   // in the partial system_trees array.

   is_new = insert_partial_system (&(partial_system_trees[cycle_number]), id,
      cycle_system, cycle_number, B, n, number_of_vertices);

   return(is_new);
   }


// This subroutine tests to see if a partial cycle system is
// isomorphic to any already tested.  If not, the partial system
// is stored in an appropriate tree.

int insert_partial_system (tree, id, cycle_system, cycle_number, B, n,
number_of_vertices)
   partial_system_tree **tree;
   char *id;
   cycle_sys *cycle_system;
   int cycle_number;
   char *B;
   int n;
   int number_of_vertices;
   {
   partial_system_tree *temp_tree, *tree_curr;
   int inserted=0;
   char found_duplicate=0;
   int compare_id;
```

```
int i,j;

if(*tree==NULL)
// The partial system is new; store it.
   {
   if(( temp_tree = (partial_system_tree *) malloc (1 * sizeof
     (partial_system_tree))) == NULL)
     {
     printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
     exit(1);
     }

   if(( temp_tree->partial_system = (cycle_sys *) malloc (cycle_number *
     sizeof(cycle_sys))) == NULL)
      {
      printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
      exit(1);
      }

   for(i=0;i<cycle_number;i++)
      {
      if(( ((temp_tree->partial_system)[i]).cycle = (char *) malloc (n *
        sizeof(char))) == NULL)
         {
         printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
         exit(1);
         }

      for(j=0;j<n;j++)
         {
         (((temp_tree->partial_system)[i]).cycle)[j] =
           ((cycle_system[i]).cycle)[j];
         }
      }

   temp_tree->left=NULL;
   temp_tree->right=NULL;

   *tree = temp_tree;
   inserted=1;
   }

else
// Go through the tree to see if the partial system has already been entered.
// If not, insert it in the appropriate position in the tree.
   {
   tree_curr = *tree;

   while ((!inserted) && (!found_duplicate))
      {
      // Form the adjacency matrix for the graph associated with the
```

```
// partial system and get the partial system's id.

for(i=0;i<(B_n*B_n-B_n)/2;i++)
   B[i]=0;

for(i=0; i<cycle_number; i++)
   {
   for(j=1;j<n;j++)
      set_bit(B, B_n, n+n*i+j, n+n*i+(j+1), 1);
   set_bit(B, B_n, n+n*i+n, n+n*i+1, 1);

   for(j=0;j<n;j++)
      set_bit(B, B_n, n+n*i+(j+1),
        (((tree_curr->partial_system)[i]).cycle)[j], 1);
   }

for(i=0;i<id_length;i++)
   id2[i]=0;
get_id(B,n,id2,number_of_vertices);

// Compare the partial system's identifier (id) with that of the
// partial system in the current tree node (id2).  If they are
// equal, the partial systems are isomorphic.  If id < id2, search
// the left subtree, if possible.  Otherwise, search the right
// subtree, if possible.

compare_id = strcmp (id, id2);
if(compare_id==0)
   found_duplicate = 1;
else if (compare_id < 0)
   {
   if(tree_curr->left !=NULL)
      tree_curr = tree_curr->left;
   else
      {
      if((temp_tree = (partial_system_tree *) malloc (1 *
        sizeof(partial_system_tree))) == NULL)
         {
         printf("\n\nAllocation of memory failed...line %d\n",
           __LINE__);
         exit(1);
         }
      if((temp_tree->partial_system = (cycle_sys *) malloc
        (cycle_number * sizeof (cycle_sys))) == NULL)
         {
         printf("\n\nAllocation of memory failed...line %d\n",
           __LINE__);
         exit(1);
         }

      for(i=0;i<cycle_number;i++)
```

```
         {
         if (( (((temp_tree)->partial_system)[i]).cycle = (char *)
           malloc (n * sizeof (char))) == NULL)
            {
            printf("\n\nAllocation of memory failed...line %d\n",
              __LINE__);
            exit(1);
            }

         for(j=0;j<n;j++)
            ((((temp_tree)->partial_system)[i]).cycle)[j] =
              ((cycle_system[i]).cycle)[j];
         }

      temp_tree->left=NULL;
      temp_tree->right=NULL;

      tree_curr->left = temp_tree;

      inserted=1;
      } // end else
   } // end else

else
   {
   if(tree_curr->right !=NULL)
      tree_curr = tree_curr->right;
   else
      {
      if((temp_tree = (partial_system_tree *) malloc (1 *
        sizeof(partial_system_tree))) == NULL)
         {
         printf("\n\nAllocation of memory failed...line %d\n",
           __LINE__);
         exit(1);
         }

      if((temp_tree->partial_system = (cycle_sys *) malloc
        (cycle_number * sizeof (cycle_sys))) == NULL)
         {
         printf("\n\nAllocation of memory failed...line %d\n",
           __LINE__);
         exit(1);
         }

      for(i=0;i<cycle_number;i++)
         {
         if (( (((temp_tree)->partial_system)[i]).cycle = (char *)
           malloc (n * sizeof (char))) == NULL)
            {
            printf("\n\nAllocation of memory failed...line %d\n",
```

```
                    __LINE__);
                  exit(1);
                  }

               for(j=0;j<n;j++)
                  ((((temp_tree)->partial_system)[i]).cycle)[j] =
                    ((cycle_system[i]).cycle)[j];
               }

            temp_tree->left=NULL;
            temp_tree->right=NULL;

            tree_curr->right = temp_tree;


            inserted=1;
            } // end else
         } // end else

      } // end while
   } // end else

   return(inserted);
   }


// This subroutine counts the number of cycle systems in a tree.  It
// is used to count the number of cycle systems associated with a
// particular invariant.

int count_cycle_systems (cycle_system_tree)
   cycle_node *cycle_system_tree;
   {
   int count=0;

   if(cycle_system_tree!=NULL)
      {
      count+=count_cycle_systems(cycle_system_tree->left);
      count++;
      count+=count_cycle_systems(cycle_system_tree->right);
      }

   return(count);
   }


// This subroutine, given two vertices v1 and v2 of an n-cycle system of
// order n, finds the cycle which contains the edge {v1,v2}.

int find_edge(v1,v2,cycle_system,n)
   char v1;
```

```
char v2;
cycle_sys *cycle_system;
int n;
{
int i,j;
int return_value;

// Find the cycle with edge {v1,v2}.

for(i=0;i<total_cycles;i++)
   {
   if( ((cycle_system[i]).cycle)[0]==v1 )
      {
      if( (((cycle_system[i]).cycle)[1]==v2) ||
        (((cycle_system[i]).cycle)[n-1]==v2) )
         {
         return_value=i;
         break;
         }
      }

   else if( ((cycle_system[i]).cycle)[n-1]==v1)
      {
      if( (((cycle_system[i]).cycle)[n-2]==v2) ||
        (((cycle_system[i]).cycle)[0]==v2) )
         {
         return_value=i;
         break;
         }
      }

   else
      {
      for(j=1;j<n-1;j++)
         {
         if ( ((cycle_system[i]).cycle)[j]==v1)
            {
            if( (((cycle_system[i]).cycle)[j-1]==v2) ||
              (((cycle_system[i]).cycle)[j+1]==v2) )
               {
               return_value=i;
               break;
               }
            }
         }
      }

   }

return(return_value);
}
```

```
// This subroutine compares the invariants of two cycle systems.

int compare_invariant(n, cycle_structure1, cycle_structure2, S_3_1,
S_3_2, S_4_1, S_4_2, S_5_1, S_5_2, S_6_1, S_6_2, group2)
   int n;
   int *cycle_structure1;
   int *cycle_structure2;
   int *S_3_1;
   int *S_3_2;
   int *S_4_1;
   int *S_4_2;
   int *S_5_1;
   int *S_5_2;
   int *S_6_1;
   int *S_6_2;
   int group2;
   {
   int compare_value;

   compare_value = compare_integer_array(cycle_structure1,
     cycle_structure2, vector_count);
   if(compare_value!=0)
      return(compare_value);

   compare_value = compare_integer_array(S_3_1, S_3_2, total_cycles);
   if(compare_value!=0)
      return(compare_value);

   if(n>=7)
      {
      compare_value = compare_integer_array(S_4_1, S_4_2, total_cycles);
      if(compare_value!=0)
         return(compare_value);
      }

   if(n>=9)
      {
      compare_value = compare_integer_array(S_5_1, S_5_2, total_cycles);
      if(compare_value!=0)
         return(compare_value);
      }

   if(n>=11)
      {
      compare_value = compare_integer_array(S_6_1, S_6_2, total_cycles);
      if(compare_value!=0)
         return(compare_value);
      }
```

```
   compare_value = compare_integers(groupsize_int, group2);
   if(compare_value!=0)
      return(compare_value);

   return(0);
   }


// This subroutine finds the sum-bicolour vector of rank 3 for the cycle system.

void find_S_3(cycle_system, S_3, n)
   cycle_sys *cycle_system;
   int *S_3;
   int n;
   {
   int i,j;
   int y;

   for(i=0;i<total_cycles;i++)
      S_3[i]=0;

   for(i=0;i<total_cycles;i++)
      {
      for(j=0;j<n-2;j++)
         {
         y = find_edge( ((cycle_system[i]).cycle)[j],
             ((cycle_system[i]).cycle)[j+2], cycle_system, n);
         S_3[y]++;
         }

      y = find_edge( ((cycle_system[i]).cycle)[n-2],
          ((cycle_system[i]).cycle)[0], cycle_system, n);
      S_3[y]++;

      y = find_edge( ((cycle_system[i]).cycle)[n-1],
          ((cycle_system[i]).cycle)[1], cycle_system, n);
      S_3[y]++;
      }

/*
   printf("S_3:  ");
   for(i=0;i<total_cycles;i++)
      printf("%d ", S_3[i]);
   printf("\n");
*/
   order_array(S_3);

   return;
   }
```

```
// This subroutine finds the sum-bicolour vector of rank 4 for the system.

void find_S_4(cycle_system, S_4, n)
   cycle_sys *cycle_system;
   int *S_4;
   int n;
   {
   int i,j;
   int y;

   for(i=0;i<total_cycles;i++)
      S_4[i]=0;

   for(i=0;i<total_cycles;i++)
      {
      for(j=0;j<n-3;j++)
         {
         y = find_edge( ((cycle_system[i]).cycle)[j],
          ((cycle_system[i]).cycle)[j+3], cycle_system, n);
         S_4[y]++;
         }

      y = find_edge( ((cycle_system[i]).cycle)[n-3],
          ((cycle_system[i]).cycle)[0], cycle_system, n);
      S_4[y]++;

      y = find_edge( ((cycle_system[i]).cycle)[n-2],
          ((cycle_system[i]).cycle)[1], cycle_system, n);
      S_4[y]++;

      y = find_edge( ((cycle_system[i]).cycle)[n-1],
          ((cycle_system[i]).cycle)[2], cycle_system, n);
      S_4[y]++;
      }

/*
   printf("S_4:  ");
   for(i=0;i<total_cycles;i++)
      printf("%d ", S_4[i]);
   printf("\n");
*/

   order_array(S_4);
   }


// This subroutine finds the sum-bicolour vector of rank 5 for the system.

void find_S_5(cycle_system, S_5, n)
   cycle_sys *cycle_system;
   int *S_5;
```

```
   int n;
   {
   int i,j;
   int y;

   for(i=0;i<total_cycles;i++)
      S_5[i]=0;

   for(i=0;i<total_cycles;i++)
      {
      for(j=0;j<n-4;j++)
         {
         y = find_edge( ((cycle_system[i]).cycle)[j],
          ((cycle_system[i]).cycle)[j+4], cycle_system, n);
         S_5[y]++;
         }

      y = find_edge( ((cycle_system[i]).cycle)[n-4],
          ((cycle_system[i]).cycle)[0], cycle_system, n);
      S_5[y]++;

      y = find_edge( ((cycle_system[i]).cycle)[n-3],
          ((cycle_system[i]).cycle)[1], cycle_system, n);
      S_5[y]++;

      y = find_edge( ((cycle_system[i]).cycle)[n-2],
          ((cycle_system[i]).cycle)[2], cycle_system, n);
      S_5[y]++;

      y = find_edge( ((cycle_system[i]).cycle)[n-1],
          ((cycle_system[i]).cycle)[3], cycle_system, n);
      S_5[y]++;
      }

/*
   printf("S_5:  ");
   for(i=0;i<total_cycles;i++)
      printf("%d ", S_5[i]);
   printf("\n");
*/

   order_array(S_5);
   }


// This subroutine finds the sum-bicolour vector of rank 6 for the system.

void find_S_6(cycle_system, S_6, n)
   cycle_sys *cycle_system;
   int *S_6;
   int n;
```

```
   {
   int i,j;
   int y;

   for(i=0;i<total_cycles;i++)
      S_6[i] = 0;

   for(i=0;i<total_cycles;i++)
      {
      for(j=0;j<n-5;j++)
         {
         y = find_edge( ((cycle_system[i]).cycle)[j],
          ((cycle_system[i]).cycle)[j+5], cycle_system, n);
         S_6[y]++;
         }

      y = find_edge( ((cycle_system[i]).cycle)[n-5],
          ((cycle_system[i]).cycle)[0], cycle_system, n);
      S_6[y]++;

      y = find_edge( ((cycle_system[i]).cycle)[n-4],
          ((cycle_system[i]).cycle)[1], cycle_system, n);
      S_6[y]++;

      y = find_edge( ((cycle_system[i]).cycle)[n-3],
          ((cycle_system[i]).cycle)[2], cycle_system, n);
      S_6[y]++;

      y = find_edge( ((cycle_system[i]).cycle)[n-2],
          ((cycle_system[i]).cycle)[3], cycle_system, n);
      S_6[y]++;

      y = find_edge( ((cycle_system[i]).cycle)[n-1],
          ((cycle_system[i]).cycle)[4], cycle_system, n);
      S_6[y]++;
      }

   order_array(S_6);

/*
   printf("S_6:  ");
   for(i=0;i<total_cycles;i++)
      printf("%d ", S_6[i]);
   printf("\n");
*/

   }


// This subroutine, given the integer array S of length total_cycles,
// puts the elements of S in nondecreasing order.
```

```
void order_array(S)
   int *S;
   {
   int i,j;
   int temp;

   for(i=0;i<total_cycles;i++)
      {
      for(j=i+1;j<total_cycles;j++)
         {
         if(S[j]<S[i])
            {
            temp=S[j];
            S[j]=S[i];
            S[i]=temp;
            }
         }
      }

   return;
   }


// A subroutine to compare the value of two integers.

int compare_integers(int1, int2)
   int int1;
   int int2;
   {
   int return_value=0;

   if(int1>int2)
      return_value=1;
   else if (int1<int2)
      return_value=2;

   return(return_value);
   }


// This subroutine finds the automorphism group order of a cycle system.

void find_group_size(cycle_system, n, B)
   cycle_sys *cycle_system;
   int n;
   char *B;
   {
   int m;
   int i,j;
```

```
DYNALLSTAT(graph,g,g_sz);
DYNALLSTAT(graph,canong,canong_sz);
DYNALLSTAT(int,lab,lab_sz);
DYNALLSTAT(int,ptn,ptn_sz);
DYNALLSTAT(int,orbits,orbits_sz);

static DEFAULTOPTIONS(options);
statsblk(stats);
setword workspace[WK_SP_SZ];
set *gv;

m = (B_n+WORDSIZE-1)/WORDSIZE;
nauty_check (WORDSIZE, m, B_n, NAUTYVERSIONID);

if(WK_SP_SZ < 50 * m)
   {
   printf("WK_SP_SZ=%d is smaller than 50*m=%d\n", WK_SP_SZ, 50*m);
   exit(1);
   }

DYNALLOC2(graph,g,g_sz,m,B_n,"malloc");
DYNALLOC2(graph,canong,canong_sz,m,B_n,"malloc");
DYNALLOC1(int,lab,lab_sz,B_n,"malloc");
DYNALLOC1(int,ptn,ptn_sz,B_n,"malloc");
DYNALLOC1(int,orbits,orbits_sz,B_n,"malloc");

options.writeautoms = FALSE;
options.writemarkers = FALSE;
options.digraph = FALSE;
options.defaultptn=FALSE;
options.tc_level=0;
options.getcanon=TRUE;


// Find the adjacency matrix.

for(i=0;i<(B_n*B_n-B_n)/2;i++)
   B[i]=0;

for(i=0; i<total_cycles; i++)
   {
   for(j=0;j<n-1;j++)
      set_bit(B,B_n,n+n*i+(j+1),n+n*i+(j+2),1);
   set_bit(B,B_n,n+n*i+n,n+n*i+1,1);

   for(j=0;j<n;j++)
      set_bit(B, B_n, n+n*i+(j+1),((cycle_system[i]).cycle)[j],1);
   }


for(i=1;i<=B_n;i++)
```

```
      {
      gv = GRAPHROW(g,i-1,m);
      EMPTYSET(gv,m);
      for(j=1;j<=B_n;j++)
         {
         if(what_is (B,B_n,i,j))
            {
            ADDELEMENT(gv,j-1);
            }
         }
      }

   for(i=0;i<B_n;i++)
      {
      lab[i]=i;
      ptn[i]=1;
      }
   ptn[n-1]=0;
   ptn[B_n-1]=0;

   nauty (g, lab, ptn, NILSET, orbits, &options, &stats, workspace,
          WK_SP_SZ, m, B_n, canong);

   groupsize_dbl = stats.grpsize1 * pow(10,stats.grpsize2);

   groupsize_int = groupsize_dbl;
   if( (groupsize_dbl-groupsize_int) > 0.1)
      {
      groupsize_int++;
      }

/*
   if(groupsize_int!=1)
      printf("Groupsize=%d\n", groupsize_int);
*/

   return;
   }
```